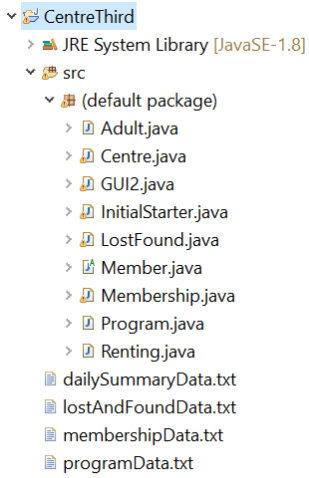
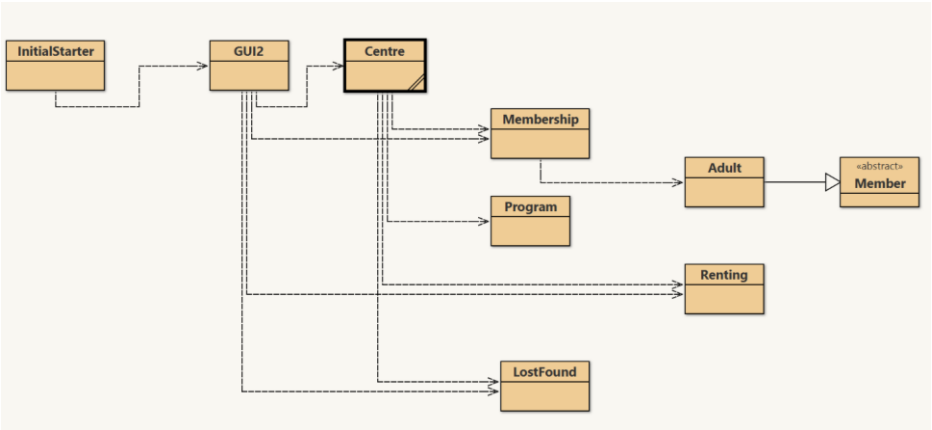
Appendix – Code Development

Class Organization





Source Code:

```
1.  import java.awt.EventQueue;
2.
3.  import javax.swing.JFrame;
4.  import javax.swing.JLabel;
5.  import javax.swing.JOptionPane;
6.
7.  import java.awt.Font;
8.  import javax.swing.JTextField;
9.  import javax.swing.JButton;
10. import java.awt.event.ActionListener;
11. import java.io.File;
12. import java.awt.event.ActionEvent;
13.
14. public class InitialStarter {
15.
16.
17.     //Instance fields declaration
18.
19.     private JFrame frame;
20.     private JTextField memFileNameT;
21.     private JTextField proFilenameT;
22.     private JTextField lofoFileNameT;
23.     private JTextField avWheelsT;
24.     private JTextField avStrollsT;
25.
26.     /**
27.      * Check if file exit in the directory
28.      *
29.      * @param name the name of the file
30.      * @return true if the file exist, false if not
31.      */
32.     public static boolean ifFileExit(String name)
33.     {
34.         File f = new File(name);
35.         if(f.exists() && !f.isDirectory()) return true;
```

```
36.         else return false;
37.     }
38.     /**
39.      * Launch the application.
40.      */
41.     public static void main(String[] args) {
42.         EventQueue.invokeLater(new Runnable() {
43.             public void run() {
44.                 try {
45.                     InitialStarter window = new InitialStarter();
46.                     window.frame.setVisible(true);
47.                 } catch (Exception e) {
48.                     e.printStackTrace();
49.                 }
50.             }
51.         });
52.     }
53. 
54.     /**
55.      * Create the application.
56.      */
57.     public InitialStarter() {
58.         initialize();
59.     }
60. 
61.     /**
62.      * Initialize the contents of the frame.
63.      */
64.     private void initialize() {
65.         frame = new JFrame();
66.         frame.setBounds(100, 100, 554, 409);
67.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68.         frame.getContentPane().setLayout(null);
69. 
70.         JLabel memFileL = new JLabel("Membership File Name: ");
71.         memFileL.setBounds(40, 85, 180, 20);
72.         frame.getContentPane().add(memFileL);
73. 
74.         JLabel lblProgramInitializer = new JLabel("Program Initializer");
75.         lblProgramInitializer.setFont(new Font("Tahoma", Font.PLAIN, 30));
76.         lblProgramInitializer.setBounds(153, 0, 261, 69);
77.         frame.getContentPane().add(lblProgramInitializer);
78. 
79.         JLabel proFileL = new JLabel("Program File Name:");
80.         proFileL.setBounds(40, 121, 170, 20);
81.         frame.getContentPane().add(proFileL);
82. 
83.         JLabel lofoFileL = new JLabel("Lost and Found File Name:");
84.         lofoFileL.setBounds(40, 157, 198, 20);
85.         frame.getContentPane().add(lofoFileL);
86. 
87.         JLabel lblNumberOfAvailable = new JLabel("Number of Available Wheelchair:");
88.         lblNumberOfAvailable.setBounds(40, 228, 241, 20);
89.         frame.getContentPane().add(lblNumberOfAvailable);
90. 
91.         JLabel lblNumberOfAvailable_1 = new JLabel("Number of Available Strollers:");
92.         lblNumberOfAvailable_1.setBounds(40, 264, 241, 20);
93.         frame.getContentPane().add(lblNumberOfAvailable_1);
94. 
95.         memFileNameT = new JTextField();
96.         memFileNameT.setText("membershipData.txt");
```

```
97.
98.         memFileNameT.setBounds(296, 82, 206, 26);
99.         frame.getContentPane().add(memFileNameT);
100.                memFileNameT.setColumns(10);
101.
102.                proFilenameT = new JTextField();
103.                proFilenameT.setBounds(296, 121, 206, 26);
104.                proFilenameT.setText("programData.txt");
105.
106.                frame.getContentPane().add(proFilenameT);
107.                proFilenameT.setColumns(10);
108.
109.                lofoFileNameT = new JTextField();
110.                lofoFileNameT.setBounds(296, 157, 206, 26);
111.
112.                lofoFileNameT.setText("lostAndFoundData.txt");
113.                frame.getContentPane().add(lofoFileNameT);
114.                lofoFileNameT.setColumns(10);
115.
116.                avWheelsT = new JTextField();
117.                avWheelsT.setBounds(296, 225, 206, 26);
118.                avWheelsT.setText("15");
119.
120.                frame.getContentPane().add(avWheelsT);
121.                avWheelsT.setColumns(10);
122.
123.                avStrollsT = new JTextField();
124.                avStrollsT.setBounds(296, 261, 206, 26);
125.                avStrollsT.setText("15");
126.                frame.getContentPane().add(avStrollsT);
127.                avStrollsT.setColumns(10);
128.
129.                JButton btnEnter = new JButton("Enter");
130.                btnEnter.addActionListener(new ActionListener() {
131.                    public void actionPerformed(ActionEvent e) {
132.                        //get all the texts
133.                        String strMemFilename = memFileNameT.getText();
134.                        //Check if file name is valid, if not, return message//
135.                        String strProFilename = proFilenameT.getText();
136.                        //Check if file name is valid, if not, return message//
137.                        String strLofoFilename = lofoFileNameT.getText();
138.                        //Check if file name is valid, if not, return message//
139.                        int numAvWheels, numAvStrolls;
140.
141.                        try {
142.                            numAvWheels = Integer.parseInt(avWheelsT.getText());
143.                            numAvStrolls = Integer.parseInt(avStrollsT.getText());
144.
145.                            if(!strMemFilename.equals("membershipData.txt"))
146.                            {
147.                                JOptionPane.showMessageDialog(null, "Invalid membership
    file name, use 'membershipData.txt'");
148.                            }
149.                            else if(!strProFilename.equals("programData.txt"))
150.                            {
151.                                JOptionPane.showMessageDialog(null, "Invalid program fil
    e name, use 'programData.txt'");
152.                            }
153.                            else if(!strLofoFilename.equals("lostAndFoundData.txt"))
154.                            {
```

```
155.                                 JOptionPane.showMessageDialog(null, "Invalid lost and fo
    und file name, use 'lostAndFoundData.txt'");
156.                             }
157.                             else if(numAvWheels < 0 || numAvStrolls < 0)
158.                             {
159.                                 JOptionPane.showMessageDialog(null, "please enter positi
    ve integer values");
160.                             }
161.                             else
162.                             {
163.                                 GUI2 acall = new GUI2();
164.                                 GUI2.main(strMemFilename, strProFilename, strLofoFilenam
    e, numAvWheels, numAvStrolls);
165.                                 frame.setVisible(false);
166.                             }
167.                         }
168.                         catch(NumberFormatException e1)
169.                         {
170.                             e1.printStackTrace();
171.                             JOptionPane.showMessageDialog(null, "File may be invalid, en
    ter correct information.");
172.                         }
173.                         //if everything is entered correctly, then call GUI2.
174.                         //MUST CHECK FIRST
175.
176.                     }
177.                 });
178.                 btnEnter.setBounds(219, 308, 115, 29);
179.                 frame.getContentPane().add(btnEnter);
180.             }
181.
182.         }
```

```
1.  import java.awt.EventQueue;
2.
3.  import javax.swing.JFrame;
4.  import java.awt.CardLayout;
5.  import javax.swing.JPanel;
6.  import javax.swing.ButtonGroup;
7.  import javax.swing.JButton;
8.  import java.awt.event.ActionListener;
9.  import java.io.IOException;
10. import java.util.ArrayList;
11. import java.awt.event.ActionEvent;
12. import javax.swing.JLabel;
13. import javax.swing.JOptionPane;
14.
15. import java.awt.Font;
16. import java.awt.Color;
17. import java.awt.Component;
18.
19. import javax.swing.JTable;
20. import javax.swing.JInternalFrame;
21. import javax.swing.JTextField;
22. import javax.swing.JComboBox;
23. import javax.swing.JScrollPane;
24. import javax.swing.SwingConstants;
25. import javax.swing.table.DefaultTableModel;
26. import javax.swing.JRadioButton;
27. import java.awt.event.MouseWheelListener;
28. import java.awt.event.WindowAdapter;
29. import java.awt.event.WindowEvent;
30. import java.awt.event.MouseWheelEvent;
31.
32. public class GUI2 {
33.     //Class fields declaration
34.     private static String MEMFILE = "membershipData.txt"; // initial initialization tha
    t sets as the base if client did not prepare any additional text
35.     private static String PROFILE = "programData.txt";
36.     private static String LOFOFILE = "lostAndFoundData.txt";
37.     private static int WHEELCHAIR_NUMBER = 15;
38.     private static int STROLLER_NUMBER = 15;
39.
40.     //Constant declaration
41.     private static final String[] LEVEL = {"Lunar","Planetary", "Stellar", "Gallactic",
    "Cosmic" };
42.     private static final String[] MEMTITLE = {"ID:", "Holders", "Adress", "contact numb
    er", "Membership Level", "Expiring date"};
43.     private static final String[] PROTITLE = {"Time:", "Location:", "Group size:", "Con
    tact", "Contact Phone"};
44.     private static final String[] RENTITLE = {"Renting Item:", "Renter:", "Contact numb
    er:"};
45.     private static final String[] LOFOTITLE = {"Date:", "Item Description", "Contact",
    "Contact Number:", "Lost?", "Found?", "Location"};
46.
47.     //instance field declaration
48.     private Centre centre;
49.     private JFrame frame;
50.     private JButton exitB;
51.     private JTable table;
52.     private JTable table_1;
53.     private JTable table_3;
54.     private JPanel mainMenuP;
55.     private JButton memB;
```

```
56.     private JButton proB;
57.     private JButton renB;
58.     private JButton lofoB;
59.     private JButton daiB;
60.     private JPanel memListP;
61.     private JPanel proListP;
62.     private JPanel renListP;
63.     private JPanel lofoListP;
64.     private JPanel daiListP;
65.     private JButton btnAdd;
66.     private JButton btnDelete;
67.     private JButton btnSearch;
68.     private JButton btnNewButton;
69.     private JButton button_1;
70.     private JButton button_2;
71.     private JButton button_3;
72.     private JButton button_5;
73.     private JButton button_6;
74.     private JButton button_8;
75.     private JButton button_9;
76.     private JPanel addMem;
77.     private JLabel lblFirstName;
78.     private JLabel lblLastName;
79.     private JLabel lblAdress;
80.     private JLabel lblMembershipLevel;
81.     private JLabel label;
82.     private JLabel lblPhoneNumber;
83.     private JLabel lblAddMembership;
84.     private JTextField fMemNameT;
85.     private JTextField lMemNameT;
86.     private JTextField memPhoneT;
87.     private JTextField memAdrT;
88.     private JScrollPane memTable;
89.     private JButton btnRefreshTable;
90.     private JButton button_10;
91.     private JButton button_11;
92.     private JButton button_12;
93.     private JButton button_13;
94.     private JScrollPane lofoTable;
95.     private JScrollPane proTable;
96.     private JButton btnBack;
97.     private JPanel searchMem;
98.     private JLabel label_1;
99.     private JLabel label_2;
100.            private JLabel label_3;
101.            private JLabel label_5;
102.            private JLabel label_6;
103.            private JLabel lblSearchMembership;
104.            private JTextField textField;
105.            private JTextField textField_1;
106.            private JTextField textField_2;
107.            private JTextField textField_3;
108.            private JButton button_15;
109.            private JLabel lblId;
110.            private JTextField idT;
111.            private JPanel addRen;
112.            private JPanel addLofo;
113.            private JLabel label_4;
114.            private JLabel label_7;
115.            private JLabel lblType;
116.            private JLabel label_12;
```

```
117.            private JLabel label_13;
118.            private JLabel lblAddRenting;
119.            private JTextField textField_10;
120.            private JTextField textField_11;
121.            private JTextField textField_12;
122.            private JButton button_17;
123.            private JButton button_18;
124.            private JLabel lblAddLostfound;
125.            private JButton button_21;
126.            private JButton button_22;
127.            private JLabel label_14;
128.            private JLabel label_18;
129.            private JTextField textField_16;
130.            private JTextField textField_17;
131.            private JLabel label_11;
132.            private JLabel label_17;
133.            private JTextField textField_14;
134.            private JTextField textField_18;
135.            private JRadioButton rdbtnLost;
136.            private JRadioButton rdbtnFound;
137.            private JLabel lblDescription;
138.            private JTextField textField_19;
139.            private JLabel lblsuccessadd;
140.            private JLabel lblinvalidadd;
141.            private Object[][] data;
142.            private JLabel lblRemainingStrollers;
143.            private JLabel lblNewLabel;
144.            private JLabel lblRemainingWheelchairs;
145.            private JLabel lblN;
146.            private JLabel lblSort;
147.            private JButton btnById;
148.            private JButton btnByHolder;
149.            private JButton btnTime;
150.            private JButton btnContact;
151.            private JLabel lblSort_1;
152.            private JButton btnByName;
153.            private JButton btnByType;
154.            private JLabel lblSort_2;
155.            private JLabel lblSort_3;
156.            private JButton btnByType_1;
157.            private JButton btnByContact_1;
158.            private JButton btnByDate;
159.            private JRadioButton rdbtnWheelChair;
160.            private JRadioButton rdbtnStroller;
161.            private final ButtonGroup buttonGroup = new ButtonGroup();
162.            private JButton btnDelete_1;
163.            private JButton btnByAdress;
164.            private JButton btnByPhone;
165.            private JButton btnByGroupSize;
166.            private JButton btnByPhone_1;
167.            private JButton btnByPhone_2;
168.            private JButton btnByDescription;
169.            private JButton btnByLocation;
170.            private JButton btnDelete_2;
171.            private JTable table_2;
172.            private JScrollPane scrollPane;
173.            private JTable table_4;
174.            private JPanel panel;
175.            private JScrollPane scrollPane_1;
176.            private JLabel lblNewMembers;
177.            private JTable table_6;
```

```
178.            private JScrollPane scrollPane_3;
179.            private JLabel lblRentingHistory;
180.            private JTable table_7;
181.            private JLabel lblLostAndFound_1;
182.            private JScrollPane scrollPane_4;
183.            private JButton btnSearch_1;
184.            private JButton btnSearch_2;
185.            private JButton btnSearch_3;
186.            private JButton btnSearch_4;
187.            private JButton btnSearch_5;
188.            private JButton btnSearch_6;
189.            private JButton btnSearch_7;
190.            private JButton btnSearch_8;
191.            private JPanel searchProP;
192.            private JLabel lblContactName;
193.            private JLabel lblContactPhone;
194.            private JLabel lblLocation;
195.            private JLabel label_15;
196.            private JLabel lblGroupSize;
197.            private JLabel lblSearchProgram;
198.            private JTextField textSeaProCon;
199.            private JTextField textSeaProPhone;
200.            private JTextField textSeaProGoup;
201.            private JTextField textSeaProLoc;
202.            private JButton bSearchProBack;
203.            private JButton bSearhProName;
204.            private JButton bSearhProPhone;
205.            private JButton bSearhProGroup;
206.            private JButton bSearhProLoc;
207.            private JPanel searchRenP;
208.            private JLabel label_8;
209.            private JLabel label_9;
210.            private JTextField textSearchRenName;
211.            private JTextField textSearchRenPhone;
212.            private JButton bSearchRenBack;
213.            private JButton bSearchRenName;
214.            private JButton bSearchRenPhone;
215.            private JLabel lblSearchRenting;
216.            private JPanel searchLofoP;
217.            private JLabel lblKeyword;
218.            private JLabel lblLocation_1;
219.            private JLabel label_21;
220.            private JLabel lblSearchLostAnd;
221.            private JTextField textSearchLofoKey;
222.            private JTextField textSearchLofoLoc;
223.            private JTextField textSearchLofoPhone;
224.            private JButton button;
225.            private JLabel lblName;
226.            private JTextField textSearchLofoName;
227.            private JButton button_4;
228.            private JButton button_7;
229.            private JButton button_14;
230.            private JButton button_16;
231.
232.
233.
234.            /*
235.             * static methods
236.             */
237.
238.
```

```
239.
240.        public static void main(String mem, String pro, String lofo, int w, int s) {
241.            MEMFILE = mem;
242.            PROFILE = pro;
243.            LOFOFILE = lofo;
244.            WHEELCHAIR_NUMBER = w;
245.            STROLLER_NUMBER = s;
246.            EventQueue.invokeLater(new Runnable() {
247.                public void run() {
248.                    try {
249.                        GUI2 window = new GUI2();
250.                        window.frame.setVisible(true);
251.                    } catch (Exception e) {
252.                        e.printStackTrace();
253.                    }
254.                }
255.            });
256.        }
257.
258.
259.
260.
261.        /**
262.         * Create the application.
263.         * @throws IOException
264.         */
265.        public GUI2() {
266.            initialize();
267.        }
268.
269.        /**
270.         * Set main menu to be visible, while all others are invisible
271.         */
272.        public void back()
273.        {
274.            mainMenuP.setVisible(true);
275.            memListP.setVisible(false);
276.            proListP.setVisible(false);
277.            renListP.setVisible(false);
278.            lofoListP.setVisible(false);
279.            daiListP.setVisible(false);
280.            addMem.setVisible(false);
281.            searchMem.setVisible(false);
282.            addRen.setVisible(false);
283.            addLofo.setVisible(false);
284.            searchProP.setVisible(false);
285.            searchRenP.setVisible(false);
286.            searchLofoP.setVisible(false);
287.        }
288.
289.        /**
290.         * Set daily summary to be visible, while all others are invisible
291.         */
292.        public void backDai()
293.        {
294.            mainMenuP.setVisible(false);
295.            memListP.setVisible(false);
296.            proListP.setVisible(false);
297.            renListP.setVisible(false);
298.            lofoListP.setVisible(false);
```

```
299.                daiListP.setVisible(true);
300.                addMem.setVisible(false);
301.                searchMem.setVisible(false);
302.                addRen.setVisible(false);
303.                addLofo.setVisible(false);
304.                searchProP.setVisible(false);
305.                searchRenP.setVisible(false);
306.                searchLofoP.setVisible(false);
307.            }
308.            /**
309.             * Set lost and found to be visible, while all others are invisible
310.             */
311.            public void backlofo()
312.            {
313.                mainMenuP.setVisible(false);
314.                memListP.setVisible(false);
315.                proListP.setVisible(false);
316.                renListP.setVisible(false);
317.                lofoListP.setVisible(true);
318.                daiListP.setVisible(false);
319.                addMem.setVisible(false);
320.                searchMem.setVisible(false);
321.                addRen.setVisible(false);
322.                addLofo.setVisible(false);
323.                searchProP.setVisible(false);
324.                searchRenP.setVisible(false);
325.                searchLofoP.setVisible(false);
326.            }
327.            /**
328.             * Set add lost and found to be visible, while all others are invisible
329.             */
330.            public void backLofoAdd()
331.            {
332.                mainMenuP.setVisible(false);
333.                memListP.setVisible(false);
334.                proListP.setVisible(false);
335.                renListP.setVisible(false);
336.                lofoListP.setVisible(false);
337.                daiListP.setVisible(false);
338.                addMem.setVisible(false);
339.                searchMem.setVisible(false);
340.                addRen.setVisible(false);
341.                addLofo.setVisible(true);
342.                searchProP.setVisible(false);
343.                searchRenP.setVisible(false);
344.                searchLofoP.setVisible(false);
345.            }
346.            /**
347.             * Set main membership to be visible, while all others are invisible
348.             */
349.            public void backMem()
350.            {
351.                mainMenuP.setVisible(false);
352.                memListP.setVisible(true);
353.                proListP.setVisible(false);
354.                renListP.setVisible(false);
355.                lofoListP.setVisible(false);
356.                daiListP.setVisible(false);
357.                addMem.setVisible(false);
358.                searchMem.setVisible(false);
359.                addRen.setVisible(false);
```

```
360.                addLofo.setVisible(false);
361.                searchProP.setVisible(false);
362.                searchRenP.setVisible(false);
363.                searchLofoP.setVisible(false);
364.            }
365.            /**
366.             * Set add membership to be visible, while all others are invisible
367.             */
368.            public void backMemAdd()
369.            {
370.                mainMenuP.setVisible(false);
371.                memListP.setVisible(false);
372.                proListP.setVisible(false);
373.                renListP.setVisible(false);
374.                lofoListP.setVisible(false);
375.                daiListP.setVisible(false);
376.                addMem.setVisible(true);
377.                searchMem.setVisible(false);
378.                addRen.setVisible(false);
379.                addLofo.setVisible(false);
380.                searchProP.setVisible(false);
381.                searchRenP.setVisible(false);
382.                searchLofoP.setVisible(false);
383.            }
384.            /**
385.             * Set search membership to be visible, while all others are invisible
386.             */
387.            public void backMemSearch()
388.            {
389.                mainMenuP.setVisible(false);
390.                memListP.setVisible(false);
391.                proListP.setVisible(false);
392.                renListP.setVisible(false);
393.                lofoListP.setVisible(false);
394.                daiListP.setVisible(false);
395.                addMem.setVisible(false);
396.                searchMem.setVisible(true);
397.                addRen.setVisible(false);
398.                addLofo.setVisible(false);
399.                searchProP.setVisible(false);
400.                searchRenP.setVisible(false);
401.                searchLofoP.setVisible(false);
402.            }
403.            /**
404.             * Set programs to be visible, while all others are invisible
405.             */
406.            public void backPro()
407.            {
408.                mainMenuP.setVisible(false);
409.                memListP.setVisible(false);
410.                proListP.setVisible(true);
411.                renListP.setVisible(false);
412.                lofoListP.setVisible(false);
413.                daiListP.setVisible(false);
414.                addMem.setVisible(false);
415.                searchMem.setVisible(false);
416.                addRen.setVisible(false);
417.                addLofo.setVisible(false);
418.                searchProP.setVisible(false);
419.                searchRenP.setVisible(false);
420.                searchLofoP.setVisible(false);
```

```
421.            }
422.            /**
423.             * Set Rentings to be visible, while all others are invisible
424.             */
425.            public void backRen()
426.            {
427.                mainMenuP.setVisible(false);
428.                memListP.setVisible(false);
429.                proListP.setVisible(false);
430.                renListP.setVisible(true);
431.                lofoListP.setVisible(false);
432.                daiListP.setVisible(false);
433.                addMem.setVisible(false);
434.                searchMem.setVisible(false);
435.                addRen.setVisible(false);
436.                addLofo.setVisible(false);
437.                searchProP.setVisible(false);
438.                searchRenP.setVisible(false);
439.                searchLofoP.setVisible(false);
440.            }
441.            /**
442.             * Set add rentings to be visible, while all others are invisible
443.             */
444.            public void backRenAdd()
445.            {
446.                mainMenuP.setVisible(false);
447.                memListP.setVisible(false);
448.                proListP.setVisible(false);
449.                renListP.setVisible(false);
450.                lofoListP.setVisible(false);
451.                daiListP.setVisible(false);
452.                addMem.setVisible(false);
453.                searchMem.setVisible(false);
454.                addRen.setVisible(true);
455.                addLofo.setVisible(false);
456.                searchProP.setVisible(false);
457.                searchRenP.setVisible(false);
458.                searchLofoP.setVisible(false);
459.            }
460.            /**
461.             * Set add search lost and found to be visible, while all others are invisib
    le
462.             */
463.            public void backSearchLofo()
464.            {
465.                mainMenuP.setVisible(false);
466.                memListP.setVisible(false);
467.                proListP.setVisible(false);
468.                renListP.setVisible(false);
469.                lofoListP.setVisible(false);
470.                daiListP.setVisible(false);
471.                addMem.setVisible(false);
472.                searchMem.setVisible(false);
473.                addRen.setVisible(false);
474.                addLofo.setVisible(false);
475.                searchProP.setVisible(false);
476.                searchRenP.setVisible(false);
477.                searchLofoP.setVisible(true);
478.            }
479.
480.            /**
```

```
481.             * Set add search program to be visible, while all others are invisible
482.             */
483.            public void backSearchPro()
484.            {
485.                mainMenuP.setVisible(false);
486.                memListP.setVisible(false);
487.                proListP.setVisible(false);
488.                renListP.setVisible(false);
489.                lofoListP.setVisible(false);
490.                daiListP.setVisible(false);
491.                addMem.setVisible(false);
492.                searchMem.setVisible(false);
493.                addRen.setVisible(false);
494.                addLofo.setVisible(false);
495.                searchProP.setVisible(true);
496.                searchRenP.setVisible(false);
497.                searchLofoP.setVisible(false);
498.            }
499.
500.            /**
501.             * Set add search renting to be visible, while all others are invisible
502.             */
503.            public void backSearchRen()
504.            {
505.                mainMenuP.setVisible(false);
506.                memListP.setVisible(false);
507.                proListP.setVisible(false);
508.                renListP.setVisible(false);
509.                lofoListP.setVisible(false);
510.                daiListP.setVisible(false);
511.                addMem.setVisible(false);
512.                searchMem.setVisible(false);
513.                addRen.setVisible(false);
514.                addLofo.setVisible(false);
515.                searchProP.setVisible(false);
516.                searchRenP.setVisible(true);
517.                searchLofoP.setVisible(false);
518.            }
519.
520.
521.            /*
522.             * Non Static methods
523.             */
524.
525.
526.            /**
527.             * Initialize the contents of the frame.
528.             * @throws IOException
529.             */
530.            private void initialize() {
531.
532.                frame = new JFrame();
533.                frame.setBounds(100, 100, 700, 600);
534.                frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
535.                frame.addWindowListener(new WindowAdapter(){
536.                    public void windowClosing(WindowEvent e) {
537.                        safeExit();
538.                    }
539.                });
540.                frame.getContentPane().setLayout(new CardLayout(0, 0));
541.
```

```
542.                //Main menu and panels
543.
544.                mainMenuP = new JPanel();
545.                frame.getContentPane().add(mainMenuP, "name_1072966911474402");
546.                mainMenuP.setLayout(null);
547.
548.                memListP = new JPanel();
549.                frame.getContentPane().add(memListP, "name_1072970042823651");
550.                memListP.setLayout(null);
551.                memListP.setVisible(false);
552.
553.                proListP = new JPanel();
554.                frame.getContentPane().add(proListP, "name_1072976183404740");
555.                proListP.setLayout(null);
556.                proListP.setVisible(false);
557.
558.                renListP = new JPanel();
559.                frame.getContentPane().add(renListP, "name_1073529008307145");
560.                renListP.setLayout(null);
561.                renListP.setVisible(false);
562.
563.                lofoListP = new JPanel();
564.                frame.getContentPane().add(lofoListP, "name_1073547485642318");
565.                lofoListP.setLayout(null);
566.                lofoListP.setVisible(false);
567.
568.                daiListP = new JPanel();
569.                frame.getContentPane().add(daiListP, "name_1073558454133617");
570.                daiListP.setLayout(null);
571.                daiListP.setVisible(false);
572.
573.                //connection to Centre
574.                centre = new Centre(MEMFILE, PROFILE, LOFOFILE, WHEELCHAIR_NUMBER, STROL
    LER_NUMBER);
575.
576.                memB = new JButton("Memberships");
577.                memB.setFont(new Font("Tahoma", Font.PLAIN, 20));
578.                memB.addActionListener(new ActionListener() {
579.                    public void actionPerformed(ActionEvent e) {
580.
581.                        backMem();
582.                    }
583.                });
584.                memB.setForeground(Color.WHITE);
585.                memB.setBackground(Color.RED);
586.                memB.setBounds(90, 138, 166, 86);
587.                mainMenuP.add(memB);
588.
589.                proB = new JButton("Programs");
590.                proB.setFont(new Font("Tahoma", Font.PLAIN, 20));
591.                proB.setForeground(Color.WHITE);
592.                proB.setBackground(Color.RED);
593.                proB.addActionListener(new ActionListener() {
594.                    public void actionPerformed(ActionEvent arg0) {
595.                        backPro();
596.                    }
597.                });
598.                proB.setBounds(415, 138, 166, 86);
599.                mainMenuP.add(proB);
600.
601.                renB = new JButton("Rentings");
```

```
602.                renB.addActionListener(new ActionListener() {
603.                    public void actionPerformed(ActionEvent e) {
604.                        backRen();
605.                    }
606.                });
607.                renB.setFont(new Font("Tahoma", Font.PLAIN, 20));
608.                renB.setForeground(Color.WHITE);
609.                renB.setBackground(Color.GREEN);
610.                renB.setBounds(90, 265, 166, 86);
611.                mainMenuP.add(renB);
612.
613.                lofoB = new JButton("Lost and Found");
614.                lofoB.addActionListener(new ActionListener() {
615.                    public void actionPerformed(ActionEvent e) {
616.                        backlofo();
617.                    }
618.                });
619.                lofoB.setFont(new Font("Tahoma", Font.PLAIN, 18));
620.                lofoB.setForeground(Color.WHITE);
621.                lofoB.setBackground(Color.GREEN);
622.                lofoB.setBounds(415, 267, 166, 85);
623.                mainMenuP.add(lofoB);
624.
625.                daiB = new JButton("Daily Summary");
626.                daiB.addActionListener(new ActionListener() {
627.                    public void actionPerformed(ActionEvent e) {
628.                        refreshDaiTable();
629.                        backDai();
630.                    }
631.                });
632.                daiB.setFont(new Font("Tahoma", Font.PLAIN, 18));
633.                daiB.setForeground(Color.WHITE);
634.                daiB.setBackground(new Color(30, 144, 255));
635.                daiB.setBounds(90, 393, 166, 86);
636.                mainMenuP.add(daiB);
637.
638.                exitB = new JButton("Exit and Save");
639.                exitB.setFont(new Font("Tahoma", Font.PLAIN, 20));
640.                exitB.addActionListener(new ActionListener() {
641.                    public void actionPerformed(ActionEvent e) {
642.                        try {
643.                            centre.writeMemFile("membershipData.txt");
644.                            centre.writeLofoFile("lostAndFoundData.txt");
645.                            centre.writedaiFile("dailySummaryData.txt");
646.                        } catch (IOException e1) {
647.                            // TODO Auto-generated catch block
648.                            e1.printStackTrace();
649.                        }
650.                        System.exit(0);
651.                    }
652.                });
653.                exitB.setForeground(Color.WHITE);
654.                exitB.setBackground(new Color(30, 144, 255));
655.                exitB.setBounds(415, 392, 166, 86);
656.                mainMenuP.add(exitB);
657.
658.                JLabel lblMainMenu = new JLabel("Main Menu: ");
659.                lblMainMenu.setFont(new Font("Tahoma", Font.PLAIN, 36));
660.                lblMainMenu.setBounds(55, 40, 213, 54);
661.                mainMenuP.add(lblMainMenu);
662.
```

```
663.                JLabel lblMemberships = new JLabel("Memberships");
664.                lblMemberships.setEnabled(false);
665.                lblMemberships.setFont(new Font("Tahoma", Font.PLAIN, 36));
666.                lblMemberships.setBounds(29, 37, 283, 44);
667.                memListP.add(lblMemberships);
668.
669.                data = centre.diaplayMembership();
670.                table = new JTable(data, MEMTITLE);
671.                memTable = new JScrollPane();
672.                memTable.setViewportView(table);
673.                memTable.addMouseWheelListener(new MouseWheelListener() {
674.                    public void mouseWheelMoved(MouseWheelEvent arg0) {
675.                    }
676.                });
677.                table.setDefaultEditor(Object.class, null);
678.                memTable.setBounds(29, 137, 621, 305);
679.                memListP.add(memTable);
680.
681.
682.                //DefaultTableModel model = new DefaultTableModel(data, MEMTITLE);
683.
684.                //model.setColumnIdentifiers(MEMTITLE);
685.                //memtable.setModel(model);
686.
687.
688.                btnAdd = new JButton("Add");
689.                btnAdd.addActionListener(new ActionListener() {
690.                    public void actionPerformed(ActionEvent e) {
691.                        //add a member by calling pop up window.
692.                        backMemAdd();
693.                    }
694.                });
695.                btnAdd.setBounds(273, 53, 71, 29);
696.                memListP.add(btnAdd);
697.
698.                btnDelete = new JButton("Delete");
699.                btnDelete.addActionListener(new ActionListener() {
700.                    public void actionPerformed(ActionEvent e) {
701.                    if(data.length == centre.getMemList().size())
702.                    {
703.                        int i = table.getSelectedRow();
704.                        if(i >= 0)
705.                        {
706.                            //you need to find the member what is corresponded to that r
    ow
707.                            Membership m = centre.getMemIndex(i);
708.                            centre.deleteMembership(m);
709.                            JOptionPane.showMessageDialog(null, "Success");
710.                        }
711.                        else
712.                        {
713.                            JOptionPane.showMessageDialog(null, "Sorry I didn't catch th
    at, select a membership account again to delete");
714.                        }
715.                        refreshMemTable(centre.diaplayMembership());
716.                    }
717.                    else
718.                    {
719.                        JOptionPane.showMessageDialog(null, "You cannot delete from sear
    ch, please refresh to return to membership list");
720.                    }
```

```
721.                 }
722.             });
723.             btnDelete.setBounds(359, 53, 86, 29);
724.             memListP.add(btnDelete);
725.
726.             btnSearch = new JButton("Search");
727.             btnSearch.addActionListener(new ActionListener() {
728.                 public void actionPerformed(ActionEvent e) {
729.                     backMemSearch();
730.                 }
731.             });
732.             btnSearch.setBounds(457, 53, 86, 29);
733.             memListP.add(btnSearch);
734.
735.             btnNewButton = new JButton("Back");
736.             btnNewButton.addActionListener(new ActionListener() {
737.                 public void actionPerformed(ActionEvent e) {
738.                     back();
739.                 }
740.             });
741.             btnNewButton.setBounds(564, 53, 86, 29);
742.             memListP.add(btnNewButton);
743.
744.             btnRefreshTable = new JButton("Refresh Table");
745.             btnRefreshTable.addActionListener(new ActionListener() {
746.                 public void actionPerformed(ActionEvent e) {
747.                     refreshMemTable(centre.diaplayMembership());
748.                 }
749.             });
750.             btnRefreshTable.setBounds(467, 93, 183, 29);
751.             memListP.add(btnRefreshTable);
752.
753.             lblSort = new JLabel("Sort:");
754.             lblSort.setBounds(29, 461, 69, 20);
755.             memListP.add(lblSort);
756.
757.             btnById = new JButton("By ID");
758.             btnById.addActionListener(new ActionListener() {
759.                 public void actionPerformed(ActionEvent e) {
760.                     centre.sortMemById();
761.                     refreshMemTable(centre.diaplayMembership());
762.                 }
763.             });
764.             btnById.setBounds(106, 497, 86, 29);
765.             memListP.add(btnById);
766.
767.             btnByHolder = new JButton("By Holder");
768.             btnByHolder.addActionListener(new ActionListener() {
769.                 public void actionPerformed(ActionEvent e) {
770.                     centre.sortMemByHolder();
771.                     refreshMemTable(centre.diaplayMembership());
772.                 }
773.             });
774.             btnByHolder.setBounds(241, 497, 103, 29);
775.             memListP.add(btnByHolder);
776.
777.             btnByAdress = new JButton("By Adress");
778.             btnByAdress.addActionListener(new ActionListener() {
779.                 public void actionPerformed(ActionEvent e) {
780.                     centre.sortMemByAdress();
781.                     refreshMemTable(centre.diaplayMembership());
```

```
782.                 }
783.             });
784.             btnByAdress.setBounds(370, 497, 115, 29);
785.             memListP.add(btnByAdress);
786.
787.             btnByPhone = new JButton("By Phone");
788.             btnByPhone.addActionListener(new ActionListener() {
789.                 public void actionPerformed(ActionEvent e) {
790.                     centre.sortMemByPhone();
791.                     refreshMemTable(centre.diaplayMembership());
792.                 }
793.             });
794.             btnByPhone.setBounds(513, 497, 115, 29);
795.             memListP.add(btnByPhone);
796.
797.             JLabel lblPrograms = new JLabel("Programs");
798.             lblPrograms.setFont(new Font("Tahoma", Font.PLAIN, 36));
799.             lblPrograms.setEnabled(false);
800.             lblPrograms.setBounds(38, 34, 283, 44);
801.             proListP.add(lblPrograms);
802.
803.             proTable = new JScrollPane();
804.             proTable.setBounds(38, 145, 620, 305);
805.             proListP.add(proTable);
806.
807.             Object[][] prodata = centre.diaplayPrograms();
808.             table = new JTable(prodata, PROTITLE);
809.             table .setDefaultEditor(Object.class, null);
810.             proTable.setViewportView(table);
811.
812.             button_1 = new JButton("Back");
813.             button_1.addActionListener(new ActionListener() {
814.                 public void actionPerformed(ActionEvent e) {
815.                     back();
816.                 }
817.             });
818.             button_1.setBounds(572, 50, 86, 29);
819.             proListP.add(button_1);
820.
821.             button_10 = new JButton("Refresh Table");
822.             button_10.addActionListener(new ActionListener() {
823.                 public void actionPerformed(ActionEvent e) {
824.                     refreshProTable(centre.diaplayPrograms());
825.                 }
826.             });
827.             button_10.setBounds(475, 100, 183, 29);
828.             proListP.add(button_10);
829.
830.             btnTime = new JButton("By Time");
831.             btnTime.addActionListener(new ActionListener() {
832.                 public void actionPerformed(ActionEvent e) {
833.                     centre.sortProByTime();
834.                     refreshProTable(centre.diaplayPrograms());
835.                 }
836.             });
837.             btnTime.setBounds(95, 499, 115, 29);
838.             proListP.add(btnTime);
839.
840.             btnContact = new JButton("By Contact");
841.             btnContact.addActionListener(new ActionListener() {
842.                 public void actionPerformed(ActionEvent e) {
```

```
843.                     centre.sortProByName();
844.                     refreshProTable(centre.diaplayPrograms());
845.                 }
846.             });
847.             btnContact.setBounds(244, 499, 115, 29);
848.             proListP.add(btnContact);
849.
850.             lblSort_1 = new JLabel("Sort:");
851.             lblSort_1.setBounds(38, 478, 69, 20);
852.             proListP.add(lblSort_1);
853.
854.             btnByGroupSize = new JButton("By Group Size");
855.             btnByGroupSize.addActionListener(new ActionListener() {
856.                 public void actionPerformed(ActionEvent e) {
857.                     centre.sortProByGroup();
858.                     refreshProTable(centre.diaplayPrograms());
859.                 }
860.             });
861.             btnByGroupSize.setBounds(374, 499, 137, 29);
862.             proListP.add(btnByGroupSize);
863.
864.             btnByPhone_1 = new JButton("By Phone");
865.             btnByPhone_1.addActionListener(new ActionListener() {
866.                 public void actionPerformed(ActionEvent e) {
867.                     centre.sortProByPhone();
868.                     refreshProTable(centre.diaplayPrograms());
869.                 }
870.             });
871.             btnByPhone_1.setBounds(529, 499, 115, 29);
872.             proListP.add(btnByPhone_1);
873.
874.             btnSearch_6 = new JButton("Search");
875.             btnSearch_6.addActionListener(new ActionListener() {
876.                 public void actionPerformed(ActionEvent e) {
877.                     backSearchPro();
878.                 }
879.             });
880.             btnSearch_6.setBounds(471, 50, 86, 29);
881.             proListP.add(btnSearch_6);
882.
883.             JLabel lblRentings = new JLabel("Rentings");
884.             lblRentings.setFont(new Font("Tahoma", Font.PLAIN, 36));
885.             lblRentings.setEnabled(false);
886.             lblRentings.setBounds(35, 30, 192, 44);
887.             renListP.add(lblRentings);
888.
889.             Object[][] rendata = centre.diaplayRenting();
890.
891.             button_2 = new JButton("Add");
892.             button_2.addActionListener(new ActionListener() {
893.                 public void actionPerformed(ActionEvent e) {
894.                     backRenAdd();
895.                 }
896.             });
897.             button_2.setBounds(292, 46, 71, 29);
898.             renListP.add(button_2);
899.
900.
901.             button_5 = new JButton("Back");
902.             button_5.addActionListener(new ActionListener() {
903.                 public void actionPerformed(ActionEvent e) {
```

```
904.                        back();
905.                    }
906.                });
907.                button_5.setBounds(577, 46, 86, 29);
908.                renListP.add(button_5);
909.
910.                button_11 = new JButton("Refresh Table");
911.                button_11.addActionListener(new ActionListener() {
912.                    public void actionPerformed(ActionEvent e) {
913.                        refreshRenTable(centre.diaplayRenting());
914.                    }
915.                });
916.                button_11.setBounds(480, 91, 183, 29);
917.                renListP.add(button_11);
918.
919.                lblRemainingStrollers = new JLabel("Remaining Strollers:");
920.                lblRemainingStrollers.setBounds(35, 101, 153, 20);
921.                renListP.add(lblRemainingStrollers);
922.
923.                lblNewLabel = new JLabel(String.valueOf(centre.getAvStrolls()));
924.                lblNewLabel.setBounds(185, 101, 32, 20);
925.                renListP.add(lblNewLabel);
926.
927.                lblRemainingWheelchairs = new JLabel("Remaining Wheelchairs:");
928.                lblRemainingWheelchairs.setBounds(232, 101, 183, 20);
929.                renListP.add(lblRemainingWheelchairs);
930.
931.                lblN = new JLabel(String.valueOf(centre.getAvWheel()));
932.                lblN.setBounds(410, 101, 41, 20);
933.                renListP.add(lblN);
934.
935.                btnByName = new JButton("By Name");
936.                btnByName.addActionListener(new ActionListener() {
937.                    public void actionPerformed(ActionEvent e) {
938.                        centre.sortRenByName();
939.                        refreshRenTable(centre.diaplayRenting());
940.                    }
941.                });
942.                btnByName.setBounds(250, 487, 115, 29);
943.                renListP.add(btnByName);
944.
945.                btnByType = new JButton("By Type");
946.                btnByType.addActionListener(new ActionListener() {
947.                    public void actionPerformed(ActionEvent e) {
948.                        centre.sortRenByItem();
949.                        refreshRenTable(centre.diaplayRenting());
950.                    }
951.                });
952.                btnByType.setBounds(93, 487, 115, 29);
953.                renListP.add(btnByType);
954.
955.                lblSort_2 = new JLabel("Sort:");
956.                lblSort_2.setBounds(35, 461, 69, 20);
957.                renListP.add(lblSort_2);
958.
959.                btnByPhone_2 = new JButton("By Contact Number");
960.                btnByPhone_2.addActionListener(new ActionListener() {
961.                    public void actionPerformed(ActionEvent e) {
962.                        centre.sortRenByPhone();
963.                        refreshRenTable(centre.diaplayRenting());
964.                    }
```

```
965.                });
966.                btnByPhone_2.setBounds(410, 487, 115, 29);
967.                renListP.add(btnByPhone_2);
968.
969.                btnDelete_2 = new JButton("Delete");
970.                btnDelete_2.addActionListener(new ActionListener() {
971.                    public void actionPerformed(ActionEvent e) {
972.                        int i = table.getSelectedRow();
973.                        if(i >= 0)
974.                        {
975.                            //you need to find the member what is corresponded to that r
    ow
976.                            Renting r = centre.getRenIndex(i);
977.                            centre.deleteRenting(r, r.getType());
978.                            JOptionPane.showMessageDialog(null, "Success");
979.                        }
980.                        else
981.                        {
982.                            JOptionPane.showMessageDialog(null, "Select an item to delet
    e");
983.                        }
984.                        refreshRenTable(centre.diaplayRenting());
985.                    }
986.                });
987.                btnDelete_2.setBounds(378, 46, 85, 29);
988.                renListP.add(btnDelete_2);
989.
990.                scrollPane = new JScrollPane();
991.                scrollPane.addMouseWheelListener(new MouseWheelListener() {
992.                    public void mouseWheelMoved(MouseWheelEvent arg0) {
993.                    }
994.                });
995.                scrollPane.setBounds(45, 149, 618, 306);
996.                renListP.add(scrollPane);
997.
998.                table = new JTable(centre.diaplayRenting(), RENTITLE);
999.                scrollPane.setViewportView(table);
1000.
1001.               btnSearch_7 = new JButton("Search");
1002.               btnSearch_7.addActionListener(new ActionListener() {
1003.                   public void actionPerformed(ActionEvent e) {
1004.                       backSearchRen();
1005.                   }
1006.               });
1007.               btnSearch_7.setBounds(480, 46, 86, 29);
1008.               renListP.add(btnSearch_7);
1009.               table.setDefaultEditor(Object.class, null);
1010.
1011.               JLabel lblLostAndFound = new JLabel("Lost and Found");
1012.               lblLostAndFound.setFont(new Font("Tahoma", Font.PLAIN, 36));
1013.               lblLostAndFound.setEnabled(false);
1014.               lblLostAndFound.setBounds(33, 37, 283, 44);
1015.               lofoListP.add(lblLostAndFound);
1016.
1017.               lofoTable = new JScrollPane();
1018.               lofoTable.setBounds(33, 138, 630, 312);
1019.               lofoListP.add(lofoTable);
1020.
1021.               Object[][] lofodata = centre.diaplayLofo();
1022.               table = new JTable(lofodata, LOFOTITLE);
1023.               lofoTable.setViewportView(table);
```

```
1024.
1025.               button_6 = new JButton("Add");
1026.               button_6.addActionListener(new ActionListener() {
1027.                   public void actionPerformed(ActionEvent e) {
1028.                       backLofoAdd();
1029.                   }
1030.               });
1031.               button_6.setBounds(297, 53, 71, 29);
1032.               lofoListP.add(button_6);
1033.
1034.               button_8 = new JButton("Back");
1035.               button_8.addActionListener(new ActionListener() {
1036.                   public void actionPerformed(ActionEvent e) {
1037.                       back();
1038.                   }
1039.               });
1040.               button_8.setBounds(577, 53, 86, 29);
1041.               lofoListP.add(button_8);
1042.
1043.               button_12 = new JButton("Refresh Table");
1044.               button_12.addActionListener(new ActionListener() {
1045.                   public void actionPerformed(ActionEvent e) {
1046.                       refreshLofoTable(centre.diaplayLofo());
1047.                   }
1048.               });
1049.               button_12.setBounds(480, 92, 183, 29);
1050.               lofoListP.add(button_12);
1051.
1052.               lblSort_3 = new JLabel("Sort:");
1053.               lblSort_3.setBounds(33, 475, 69, 20);
1054.               lofoListP.add(lblSort_3);
1055.
1056.               btnByType_1 = new JButton("By Type");
1057.               btnByType_1.addActionListener(new ActionListener() {
1058.                   public void actionPerformed(ActionEvent e) {
1059.                       centre.sortLofoByType();
1060.                       refreshLofoTable(centre.diaplayLofo());
1061.                   }
1062.               });
1063.               btnByType_1.setBounds(15, 499, 104, 29);
1064.               lofoListP.add(btnByType_1);
1065.
1066.               btnByContact_1 = new JButton("By Contact");
1067.               btnByContact_1.addActionListener(new ActionListener() {
1068.                   public void actionPerformed(ActionEvent e) {
1069.                       centre.sortLofoByName();
1070.                       refreshLofoTable(centre.diaplayLofo());
1071.                   }
1072.               });
1073.               btnByContact_1.setBounds(134, 499, 115, 29);
1074.               lofoListP.add(btnByContact_1);
1075.
1076.               btnByDate = new JButton("By Date");
1077.               btnByDate.addActionListener(new ActionListener() {
1078.                   public void actionPerformed(ActionEvent e) {
1079.                       centre.sortLofoByDate();
1080.                       refreshLofoTable(centre.diaplayLofo());
1081.                   }
1082.               });
1083.               btnByDate.setBounds(264, 499, 104, 29);
1084.               lofoListP.add(btnByDate);
```

```
1085.
1086.                btnDelete_1 = new JButton("Delete");
1087.                btnDelete_1.addActionListener(new ActionListener() {
1088.                    public void actionPerformed(ActionEvent e) {
1089.                            int i = table.getSelectedRow();
1090.                            if(i >= 0)
1091.                            {
1092.                                LostFound l = centre.getLofoIndex(i);
1093.                                centre.deleteLofo(l);
1094.                                JOptionPane.showMessageDialog(null, "Success");
1095.                            }
1096.                            else
1097.                            {
1098.                                JOptionPane.showMessageDialog(null, "Select an item to d
    elete");
1099.                            }
1100.                            refreshLofoTable(centre.diaplayLofo());
1101.                    }
1102.                });
1103.                btnDelete_1.setBounds(383, 53, 86, 29);
1104.                lofoListP.add(btnDelete_1);
1105.
1106.                btnByDescription = new JButton("By Description");
1107.                btnByDescription.addActionListener(new ActionListener() {
1108.                    public void actionPerformed(ActionEvent e) {
1109.                        centre.sortLofoByDescrip();
1110.                        refreshLofoTable(centre.diaplayLofo());
1111.                    }
1112.                });
1113.                btnByDescription.setBounds(383, 499, 153, 29);
1114.                lofoListP.add(btnByDescription);
1115.
1116.                btnByLocation = new JButton("By Location");
1117.                btnByLocation.addActionListener(new ActionListener() {
1118.                    public void actionPerformed(ActionEvent e) {
1119.                        centre.sortLofoByLoc();
1120.                        refreshLofoTable(centre.diaplayLofo());
1121.                    }
1122.                });
1123.                btnByLocation.setBounds(548, 499, 115, 29);
1124.                lofoListP.add(btnByLocation);
1125.
1126.                btnSearch_8 = new JButton("Search");
1127.                btnSearch_8.addActionListener(new ActionListener() {
1128.                    public void actionPerformed(ActionEvent e) {
1129.                        backSearchLofo();
1130.                    }
1131.                });
1132.                btnSearch_8.setBounds(483, 53, 79, 29);
1133.                lofoListP.add(btnSearch_8);
1134.
1135.                JLabel lblDailySummary = new JLabel("Daily Summary");
1136.                lblDailySummary.setFont(new Font("Tahoma", Font.PLAIN, 36));
1137.                lblDailySummary.setEnabled(false);
1138.                lblDailySummary.setBounds(33, 34, 283, 44);
1139.                daiListP.add(lblDailySummary);
1140.
1141.                button_9 = new JButton("Back");
1142.                button_9.addActionListener(new ActionListener() {
1143.                    public void actionPerformed(ActionEvent e) {
1144.                        back();
```

```
1145.                 }
1146.             });
1147.             button_9.setBounds(566, 50, 86, 29);
1148.             daiListP.add(button_9);
1149.
1150.             button_13 = new JButton("Refresh Table");
1151.             button_13.addActionListener(new ActionListener() {
1152.                 public void actionPerformed(ActionEvent e) {
1153.                     refreshDaiTable();
1154.                 }
1155.             });
1156.             button_13.setBounds(469, 95, 183, 29);
1157.             daiListP.add(button_13);
1158.
1159.             data = centre.diaplayMembership();
1160.
1161.             lblNewMembers = new JLabel("Members in file:");
1162.             lblNewMembers.setBounds(32, 121, 157, 20);
1163.             daiListP.add(lblNewMembers);
1164.
1165.             scrollPane_3 = new JScrollPane();
1166.             scrollPane_3.setBounds(33, 343, 610, 68);
1167.             daiListP.add(scrollPane_3);
1168.
1169.             table_6 = new JTable(centre.diaplayRenting(), RENTITLE);
1170.             table_6.setEnabled(false);
1171.             scrollPane_3.setViewportView(table_6);
1172.
1173.             lblRentingHistory = new JLabel("Renting History:");
1174.             lblRentingHistory.setBounds(33, 319, 145, 20);
1175.             daiListP.add(lblRentingHistory);
1176.
1177.             scrollPane_4 = new JScrollPane();
1178.             scrollPane_4.setBounds(33, 457, 610, 73);
1179.             daiListP.add(scrollPane_4);
1180.
1181.             table_7 = new JTable(centre.diaplayLofo(), LOFOTITLE);
1182.             table_7.setEnabled(false);
1183.             scrollPane_4.setViewportView(table_7);
1184.
1185.             lblLostAndFound_1 = new JLabel("Lost and Found History:");
1186.             lblLostAndFound_1.setBounds(33, 421, 191, 20);
1187.             daiListP.add(lblLostAndFound_1);
1188.
1189.             scrollPane_1 = new JScrollPane();
1190.             scrollPane_1.setBounds(33, 140, 610, 163);
1191.             daiListP.add(scrollPane_1);
1192.             table_4 = new JTable(data, MEMTITLE);
1193.             table_4.setEnabled(false);
1194.             scrollPane_1.setViewportView(table_4);
1195.
1196.             panel = new JPanel();
1197.             scrollPane_1.setColumnHeaderView(panel);
1198.             panel.setLayout(null);
1199.
1200.             addMem = new JPanel();
1201.             frame.getContentPane().add(addMem, "name_1093167351919599");
1202.             addMem.setLayout(null);
1203.
1204.             lblFirstName = new JLabel("First name");
1205.             lblFirstName.setBounds(53, 108, 137, 20);
```

```
1206.                addMem.add(lblFirstName);
1207.
1208.                lblLastName = new JLabel("Last name");
1209.                lblLastName.setBounds(53, 183, 137, 20);
1210.                addMem.add(lblLastName);
1211.
1212.                lblAdress = new JLabel("Adress:");
1213.                lblAdress.setBounds(53, 330, 158, 20);
1214.                addMem.add(lblAdress);
1215.
1216.                lblMembershipLevel = new JLabel("Membership Level:");
1217.                lblMembershipLevel.setBounds(53, 399, 158, 20);
1218.                addMem.add(lblMembershipLevel);
1219.
1220.                label = new JLabel("");
1221.                label.setBounds(55, 389, 69, 20);
1222.                addMem.add(label);
1223.
1224.                lblPhoneNumber = new JLabel("Phone number: ");
1225.                lblPhoneNumber.setBounds(53, 260, 198, 20);
1226.                addMem.add(lblPhoneNumber);
1227.
1228.                lblAddMembership = new JLabel("Add Membership");
1229.                lblAddMembership.setFont(new Font("Tahoma", Font.PLAIN, 36));
1230.                lblAddMembership.setBounds(53, 31, 330, 55);
1231.                addMem.add(lblAddMembership);
1232.
1233.                fMemNameT = new JTextField();
1234.                fMemNameT.setBounds(348, 105, 270, 26);
1235.                addMem.add(fMemNameT);
1236.                fMemNameT.setColumns(10);
1237.
1238.                lMemNameT = new JTextField();
1239.                lMemNameT.setColumns(10);
1240.                lMemNameT.setBounds(348, 180, 270, 26);
1241.                addMem.add(lMemNameT);
1242.
1243.                memPhoneT = new JTextField();
1244.                memPhoneT.setColumns(10);
1245.                memPhoneT.setBounds(348, 257, 270, 26);
1246.                addMem.add(memPhoneT);
1247.
1248.                memAdrT = new JTextField();
1249.                memAdrT.setColumns(10);
1250.                memAdrT.setBounds(348, 327, 270, 26);
1251.                addMem.add(memAdrT);
1252.
1253.                JComboBox comboLevel = new JComboBox(LEVEL);
1254.                comboLevel.setBounds(348, 396, 270, 26);
1255.                addMem.add(comboLevel);
1256.
1257.                JButton btnAdd_1 = new JButton("Add");
1258.                btnAdd_1.addActionListener(new ActionListener() {
1259.
1260.                    public void actionPerformed(ActionEvent e) {
1261.                        try
1262.                        {
1263.                            int phonenum = Integer.parseInt(memPhoneT.getText());
1264.                            String first = fMemNameT.getText();
1265.                            String last = lMemNameT.getText();
1266.                            String memadress = memAdrT.getText();
```

```
1267.                           String memlevel = (String) comboLevel.getSelectedItem();
1268.                           centre.addMembership(first, last, phonenum, memadress, memle
    vel);
1269.
1270.                           JOptionPane.showMessageDialog(null, "Success, go back to vie
    w added membership");
1271.
1272.                           resetMemAdd();
1273.                           refreshMemTable(centre.diaplayMembership());
1274.                       }
1275.                       catch(NumberFormatException nfe)
1276.                       {
1277.                           JOptionPane.showMessageDialog(null, "please enter correct in
    formation and less than 6 digit phone number");
1278.                       }
1279.
1280.
1281.                   }
1282.               });
1283.               btnAdd_1.setBounds(140, 477, 115, 29);
1284.               addMem.add(btnAdd_1);
1285.
1286.               btnBack = new JButton("Back");
1287.               btnBack.addActionListener(new ActionListener() {
1288.                   public void actionPerformed(ActionEvent e) {
1289.                       backMem();
1290.                   }
1291.               });
1292.               btnBack.setBounds(420, 477, 115, 29);
1293.               addMem.add(btnBack);
1294.
1295.               searchMem = new JPanel();
1296.               frame.getContentPane().add(searchMem, "name_1096078152579458");
1297.               searchMem.setLayout(null);
1298.
1299.               label_1 = new JLabel("First name");
1300.               label_1.setBounds(60, 178, 137, 20);
1301.               searchMem.add(label_1);
1302.
1303.               label_2 = new JLabel("Last name");
1304.               label_2.setBounds(60, 234, 137, 20);
1305.               searchMem.add(label_2);
1306.
1307.               label_3 = new JLabel("Adress:");
1308.               label_3.setBounds(60, 340, 158, 20);
1309.               searchMem.add(label_3);
1310.
1311.               label_5 = new JLabel("");
1312.               label_5.setBounds(60, 390, 69, 20);
1313.               searchMem.add(label_5);
1314.
1315.               label_6 = new JLabel("Phone number: ");
1316.               label_6.setBounds(60, 289, 198, 20);
1317.               searchMem.add(label_6);
1318.
1319.               lblSearchMembership = new JLabel("Search Membership");
1320.               lblSearchMembership.setFont(new Font("Tahoma", Font.PLAIN, 36));
1321.               lblSearchMembership.setBounds(58, 32, 330, 55);
1322.               searchMem.add(lblSearchMembership);
1323.
1324.               textField = new JTextField();
```

```
1325.               textField.setColumns(10);
1326.               textField.setBounds(212, 175, 270, 26);
1327.               searchMem.add(textField);
1328.
1329.               textField_1 = new JTextField();
1330.               textField_1.setColumns(10);
1331.               textField_1.setBounds(212, 231, 270, 26);
1332.               searchMem.add(textField_1);
1333.
1334.               textField_2 = new JTextField();
1335.               textField_2.setColumns(10);
1336.               textField_2.setBounds(212, 286, 270, 26);
1337.               searchMem.add(textField_2);
1338.
1339.               textField_3 = new JTextField();
1340.               textField_3.setColumns(10);
1341.               textField_3.setBounds(212, 337, 270, 26);
1342.               searchMem.add(textField_3);
1343.
1344.               button_15 = new JButton("Back");
1345.               button_15.addActionListener(new ActionListener() {
1346.                   public void actionPerformed(ActionEvent e) {
1347.                       backMem();
1348.                   }
1349.               });
1350.               button_15.setBounds(283, 477, 115, 29);
1351.               searchMem.add(button_15);
1352.
1353.               lblId = new JLabel("ID:");
1354.               lblId.setBounds(60, 126, 69, 20);
1355.               searchMem.add(lblId);
1356.
1357.               idT = new JTextField();
1358.               idT.setBounds(212, 133, 270, 26);
1359.               searchMem.add(idT);
1360.               idT.setColumns(10);
1361.
1362.               btnSearch_1 = new JButton("Search");
1363.               btnSearch_1.addActionListener(new ActionListener() {
1364.                   public void actionPerformed(ActionEvent e) {
1365.                       //search based on id number. each id only correspond to one memb
    ership
1366.                       try
1367.                       {
1368.                           int id = Integer.parseInt(idT.getText());
1369.                           data = centre.diaplayMembership(centre.searchMemById(id));
1370.                           if(data.length != 0)
1371.                           {
1372.                               backMem();
1373.                               refreshMemTable(data);
1374.                               JOptionPane.showMessageDialog(null, "Success! Membership
    s with this id "
1375.                                       + " are listed. Refresh to return to view all me
    mbership");
1376.                           }
1377.                           else
1378.                           {
1379.                               JOptionPane.showMessageDialog(null, "Membership with thi
    s ID not found");
1380.                           }
1381.                       }
```

```
1382.                       catch(NumberFormatException nfe)
1383.                       {
1384.                           JOptionPane.showMessageDialog(null, "invalid input");
1385.                       }
1386.                   }
1387.               });
1388.               btnSearch_1.setBounds(515, 126, 115, 29);
1389.               searchMem.add(btnSearch_1);
1390.
1391.               btnSearch_2 = new JButton("Search");
1392.               btnSearch_2.addActionListener(new ActionListener() {
1393.                   public void actionPerformed(ActionEvent e) {
1394.                           String first = textField.getText();
1395.                           data = centre.diaplayMembership(centre.searchMemByFName(cent
    re.getMemList(), first));
1396.                           if(data.length != 0)
1397.                           {
1398.                               backMem();
1399.                               refreshMemTable(data);
1400.                               JOptionPane.showMessageDialog(null, "Success! Membership
    s with this first name "
1401.                                       + " are listed. Refresh to return to view all me
    mbership");
1402.                           }
1403.                           else
1404.                           {
1405.                               JOptionPane.showMessageDialog(null, "Membership with thi
    s first name not found");
1406.                           }
1407.                       }
1408.               });
1409.               btnSearch_2.setBounds(515, 174, 115, 29);
1410.               searchMem.add(btnSearch_2);
1411.
1412.               btnSearch_3 = new JButton("Search");
1413.               btnSearch_3.addActionListener(new ActionListener() {
1414.                   public void actionPerformed(ActionEvent e) {
1415.                           String last = textField_1.getText();
1416.                           data = centre.diaplayMembership(centre.searchMemByLName(cent
    re.getMemList(), last));
1417.                           if(data.length != 0)
1418.                           {
1419.                               backMem();
1420.                               refreshMemTable(data);
1421.                               JOptionPane.showMessageDialog(null, "Success! Membership
    s with this last name "
1422.                                       + " are listed. Refresh to return to view all me
    mbership");
1423.                           }
1424.                           else
1425.                           {
1426.                               JOptionPane.showMessageDialog(null, "Membership with thi
    s last name not found");
1427.                           }
1428.                   }
1429.               });
1430.               btnSearch_3.setBounds(515, 230, 115, 29);
1431.               searchMem.add(btnSearch_3);
1432.
1433.               btnSearch_4 = new JButton("Search");
1434.               btnSearch_4.addActionListener(new ActionListener() {
```

```
1435.                 public void actionPerformed(ActionEvent e) {
1436.                     try
1437.                     {
1438.                         int phone = Integer.parseInt(textField_2.getText());
1439.                         data = centre.diaplayMembership(centre.searchMemPhone(centre
    .getMemList(), phone));
1440.                         if(data.length != 0)
1441.                         {
1442.                             backMem();
1443.                             refreshMemTable(data);
1444.                             JOptionPane.showMessageDialog(null, "Success! Membership
    s with this phone number "
1445.                                     + " are listed. Refresh to return to view all me
    mbership");
1446.                         }
1447.                         else
1448.                         {
1449.                             JOptionPane.showMessageDialog(null, "Membership with thi
    s phone number not found");
1450.                         }
1451.                     }
1452.                     catch(NumberFormatException nfe)
1453.                     {
1454.                         JOptionPane.showMessageDialog(null, "invalid input");
1455.                     }
1456.                 }
1457.             });
1458.             btnSearch_4.setBounds(515, 285, 115, 29);
1459.             searchMem.add(btnSearch_4);
1460.
1461.             btnSearch_5 = new JButton("Search");
1462.             btnSearch_5.addActionListener(new ActionListener() {
1463.                 public void actionPerformed(ActionEvent e) {
1464.                     String adress = textField_3.getText();
1465.                     data = centre.diaplayMembership(centre.searchMemByAdress(centre.
    getMemList(), adress));
1466.                     if(data.length != 0)
1467.                     {
1468.                         backMem();
1469.                         refreshMemTable(data);
1470.                         JOptionPane.showMessageDialog(null, "Success! Memberships wi
    th this adress "
1471.                                 + " are listed. Refresh to return to view all member
    ship");
1472.                     }
1473.                     else
1474.                     {
1475.                         JOptionPane.showMessageDialog(null, "Membership with this ad
    ress not found");
1476.                     }
1477.                 }
1478.             });
1479.             btnSearch_5.setBounds(515, 336, 115, 29);
1480.             searchMem.add(btnSearch_5);
1481.
1482.             addRen = new JPanel();
1483.             frame.getContentPane().add(addRen, "name_1096600662277798");
1484.             addRen.setLayout(null);
1485.
1486.             label_4 = new JLabel("First name");
1487.             label_4.setBounds(55, 143, 137, 20);
```

```
1488.                addRen.add(label_4);
1489.
1490.                label_7 = new JLabel("Last name");
1491.                label_7.setBounds(55, 218, 137, 20);
1492.                addRen.add(label_7);
1493.
1494.                lblType = new JLabel("Type:");
1495.                lblType.setBounds(55, 365, 158, 20);
1496.                addRen.add(lblType);
1497.
1498.                label_12 = new JLabel("");
1499.                label_12.setBounds(55, 387, 69, 20);
1500.                addRen.add(label_12);
1501.
1502.                label_13 = new JLabel("Phone number: ");
1503.                label_13.setBounds(55, 295, 198, 20);
1504.                addRen.add(label_13);
1505.
1506.                lblAddRenting = new JLabel("Add Renting");
1507.                lblAddRenting.setFont(new Font("Tahoma", Font.PLAIN, 36));
1508.                lblAddRenting.setBounds(53, 29, 330, 55);
1509.                addRen.add(lblAddRenting);
1510.
1511.                textField_10 = new JTextField();
1512.                textField_10.setColumns(10);
1513.                textField_10.setBounds(350, 140, 270, 26);
1514.                addRen.add(textField_10);
1515.
1516.                textField_11 = new JTextField();
1517.                textField_11.setColumns(10);
1518.                textField_11.setBounds(350, 215, 270, 26);
1519.                addRen.add(textField_11);
1520.
1521.                textField_12 = new JTextField();
1522.                textField_12.setColumns(10);
1523.                textField_12.setBounds(350, 292, 270, 26);
1524.                addRen.add(textField_12);
1525.
1526.                button_17 = new JButton("Add");
1527.                button_17.addActionListener(new ActionListener() {
1528.                    public void actionPerformed(ActionEvent e) {
1529.                        try
1530.                        {
1531.                            int phonenum = Integer.parseInt(textField_12.getText());
1532.                            String first = textField_10.getText();
1533.                            String last = textField_11.getText();
1534.                            String type;
1535.                            if(rdbtnWheelChair.isSelected())
1536.                            {
1537.                                type = "Wheelchair";
1538.                                if(centre.getAvWheel() > 0)
1539.                                {
1540.                                    centre.addRenter(first, last, phonenum, type);
1541.                                    JOptionPane.showMessageDialog(null, "Success, go bac
    k and refresh to check changes");
1542.                                    resetRenAdd();
1543.                                    refreshRenTable(centre.diaplayRenting());
1544.                                }
1545.                                else
1546.                                {
```

```
1547.                                   JOptionPane.showMessageDialog(null, "Invalid! Not en
    ough Wheelchair.");
1548.                               }
1549.                           }
1550.                           else if(rdbtnStroller.isSelected())
1551.                           {
1552.                               type = "Stroller";
1553.                               if(centre.getAvStrolls() > 0)
1554.                               {
1555.                                   centre.addRenter(first, last, phonenum, type);
1556.                                   JOptionPane.showMessageDialog(null, "Success, go bac
    k and refresh to check changes");
1557.                                   resetRenAdd();
1558.                                   refreshRenTable(centre.diaplayRenting());
1559.                               }
1560.                               else
1561.                               {
1562.                                   JOptionPane.showMessageDialog(null, "Invalid! Not en
    ough Strollers.");
1563.                               }
1564.                           }
1565.                           else
1566.                           {
1567.                               JOptionPane.showMessageDialog(null, "Please select wheel
    chair or stoller");
1568.                           }
1569.
1570.                       }
1571.                       catch(NumberFormatException nfe)
1572.                       {
1573.                           JOptionPane.showMessageDialog(null, "please enter correct in
    formation and less than 6 digit phone number");
1574.                       }
1575.                   }
1576.               });
1577.               button_17.setBounds(140, 475, 115, 29);
1578.               addRen.add(button_17);
1579.
1580.               button_18 = new JButton("Back");
1581.               button_18.addActionListener(new ActionListener() {
1582.                   public void actionPerformed(ActionEvent e) {
1583.                       backRen();
1584.                   }
1585.               });
1586.               button_18.setBounds(420, 475, 115, 29);
1587.               addRen.add(button_18);
1588.
1589.               rdbtnWheelChair = new JRadioButton("Wheel Chair");
1590.               buttonGroup.add(rdbtnWheelChair);
1591.               rdbtnWheelChair.setBounds(155, 361, 155, 29);
1592.               addRen.add(rdbtnWheelChair);
1593.
1594.               rdbtnStroller = new JRadioButton("Stroller");
1595.               buttonGroup.add(rdbtnStroller);
1596.               rdbtnStroller.setBounds(395, 361, 155, 29);
1597.               addRen.add(rdbtnStroller);
1598.
1599.               addLofo = new JPanel();
1600.               frame.getContentPane().add(addLofo, "name_1096655461299714");
1601.               addLofo.setLayout(null);
1602.
```

```
1603.               lblAddLostfound = new JLabel("Add Lost&Found");
1604.               lblAddLostfound.setFont(new Font("Tahoma", Font.PLAIN, 36));
1605.               lblAddLostfound.setBounds(58, 26, 330, 55);
1606.               addLofo.add(lblAddLostfound);
1607.
1608.               button_21 = new JButton("Add");
1609.               button_21.addActionListener(new ActionListener() {
1610.                   public void actionPerformed(ActionEvent e) {
1611.                       try
1612.                       {
1613.                           String descrip = textField_19.getText();
1614.                           String first = textField_14.getText();
1615.                           String last = textField_18.getText();
1616.                           String location = textField_16.getText();
1617.                           int phone = Integer.parseInt(textField_17.getText());
1618.                           boolean l,f;
1619.                           if(rdbtnLost.isSelected())
1620.                           {
1621.                               l = true;
1622.                               f = false;
1623.                               centre.addLofo(descrip, first, last, location, l, f, pho
    ne);
1624.                               JOptionPane.showMessageDialog(null, "Success, go back to
     view lost and founds");
1625.
1626.                               resetLofoAdd();
1627.                               refreshLofoTable(centre.diaplayLofo());
1628.                           }
1629.                           else if(rdbtnFound.isSelected())
1630.                           {
1631.                               l = false;
1632.                               f = true;
1633.                               centre.addLofo(descrip, first, last, location, l, f, pho
    ne);
1634.                               JOptionPane.showMessageDialog(null, "Success, go back to
     view lost and founds");
1635.
1636.                               resetLofoAdd();
1637.                               refreshLofoTable(centre.diaplayLofo());
1638.                           }
1639.                           else
1640.                           {
1641.                               JOptionPane.showMessageDialog(null, "Success, go back to
     view lost and founds");
1642.                           }
1643.                       }
1644.                       catch(NumberFormatException nfe)
1645.                       {
1646.                           JOptionPane.showMessageDialog(null, "please enter correct in
    formation and less than 6 digit phone number");
1647.                       }
1648.
1649.                   }
1650.               });
1651.               button_21.setBounds(145, 472, 115, 29);
1652.               addLofo.add(button_21);
1653.
1654.               button_22 = new JButton("Back");
1655.               button_22.addActionListener(new ActionListener() {
1656.                   public void actionPerformed(ActionEvent e) {
1657.                       backlofo();
```

```
1658.                     }
1659.                 });
1660.                 button_22.setBounds(425, 472, 115, 29);
1661.                 addLofo.add(button_22);
1662.
1663.                 label_14 = new JLabel("Location:");
1664.                 label_14.setBounds(58, 304, 137, 20);
1665.                 addLofo.add(label_14);
1666.
1667.                 label_18 = new JLabel("Phone number: ");
1668.                 label_18.setBounds(58, 359, 198, 20);
1669.                 addLofo.add(label_18);
1670.
1671.                 textField_16 = new JTextField();
1672.                 textField_16.setColumns(10);
1673.                 textField_16.setBounds(353, 301, 270, 26);
1674.                 addLofo.add(textField_16);
1675.
1676.                 textField_17 = new JTextField();
1677.                 textField_17.setColumns(10);
1678.                 textField_17.setBounds(353, 356, 270, 26);
1679.                 addLofo.add(textField_17);
1680.
1681.                 label_11 = new JLabel("First name");
1682.                 label_11.setBounds(58, 190, 137, 20);
1683.                 addLofo.add(label_11);
1684.
1685.                 label_17 = new JLabel("Last name");
1686.                 label_17.setBounds(58, 248, 137, 20);
1687.                 addLofo.add(label_17);
1688.
1689.                 textField_14 = new JTextField();
1690.                 textField_14.setColumns(10);
1691.                 textField_14.setBounds(353, 187, 270, 26);
1692.                 addLofo.add(textField_14);
1693.
1694.                 textField_18 = new JTextField();
1695.                 textField_18.setColumns(10);
1696.                 textField_18.setBounds(353, 245, 270, 26);
1697.                 addLofo.add(textField_18);
1698.
1699.                 rdbtnLost = new JRadioButton("Lost");
1700.                 buttonGroup.add(rdbtnLost);
1701.                 rdbtnLost.setBounds(155, 417, 155, 29);
1702.                 addLofo.add(rdbtnLost);
1703.
1704.                 rdbtnFound = new JRadioButton("Found");
1705.                 buttonGroup.add(rdbtnFound);
1706.                 rdbtnFound.setBounds(380, 417, 155, 29);
1707.                 addLofo.add(rdbtnFound);
1708.
1709.                 lblDescription = new JLabel("Description: ");
1710.                 lblDescription.setBounds(58, 137, 115, 20);
1711.                 addLofo.add(lblDescription);
1712.
1713.                 textField_19 = new JTextField();
1714.                 textField_19.setBounds(353, 134, 270, 26);
1715.                 addLofo.add(textField_19);
1716.                 textField_19.setColumns(10);
1717.
1718.                 searchProP = new JPanel();
```

```
1719.               frame.getContentPane().add(searchProP, "name_1255531108664222");
1720.               searchProP.setLayout(null);
1721.
1722.               lblContactName = new JLabel("Contact Name");
1723.               lblContactName.setBounds(60, 142, 137, 20);
1724.               searchProP.add(lblContactName);
1725.
1726.               lblContactPhone = new JLabel("Contact Phone:");
1727.               lblContactPhone.setBounds(60, 206, 137, 20);
1728.               searchProP.add(lblContactPhone);
1729.
1730.               lblLocation = new JLabel("Location");
1731.               lblLocation.setBounds(60, 338, 158, 20);
1732.               searchProP.add(lblLocation);
1733.
1734.               label_15 = new JLabel("");
1735.               label_15.setBounds(60, 388, 69, 20);
1736.               searchProP.add(label_15);
1737.
1738.               lblGroupSize = new JLabel("Group Size");
1739.               lblGroupSize.setBounds(60, 272, 198, 20);
1740.               searchProP.add(lblGroupSize);
1741.
1742.               lblSearchProgram = new JLabel("Search Program");
1743.               lblSearchProgram.setFont(new Font("Tahoma", Font.PLAIN, 36));
1744.               lblSearchProgram.setBounds(58, 30, 330, 55);
1745.               searchProP.add(lblSearchProgram);
1746.
1747.               textSeaProCon = new JTextField();
1748.               textSeaProCon.setColumns(10);
1749.               textSeaProCon.setBounds(212, 139, 270, 26);
1750.               searchProP.add(textSeaProCon);
1751.
1752.               textSeaProPhone = new JTextField();
1753.               textSeaProPhone.setColumns(10);
1754.               textSeaProPhone.setBounds(212, 203, 270, 26);
1755.               searchProP.add(textSeaProPhone);
1756.
1757.               textSeaProGoup = new JTextField();
1758.               textSeaProGoup.setColumns(10);
1759.               textSeaProGoup.setBounds(212, 269, 270, 26);
1760.               searchProP.add(textSeaProGoup);
1761.
1762.               textSeaProLoc = new JTextField();
1763.               textSeaProLoc.setColumns(10);
1764.               textSeaProLoc.setBounds(212, 335, 270, 26);
1765.               searchProP.add(textSeaProLoc);
1766.
1767.               bSearchProBack = new JButton("Back");
1768.               bSearchProBack.addActionListener(new ActionListener() {
1769.                   public void actionPerformed(ActionEvent e) {
1770.                       backPro();
1771.                   }
1772.               });
1773.               bSearchProBack.setBounds(283, 475, 115, 29);
1774.               searchProP.add(bSearchProBack);
1775.
1776.               bSearhProName = new JButton("Search");
1777.               bSearhProName.addActionListener(new ActionListener() {
1778.                   public void actionPerformed(ActionEvent e) {
1779.                       String name = textSeaProCon.getText();
```

```
1780.                         data = centre.diaplayPrograms(centre.searchProByName(name));
1781.                         if(data.length != 0)
1782.                         {
1783.                             backPro();
1784.                             refreshProTable(data);
1785.                             JOptionPane.showMessageDialog(null, "Success! Programs with
    this name "
1786.                                     + " are listed. Refresh to return to view all Progra
    ms");
1787.                         }
1788.                         else
1789.                         {
1790.                             JOptionPane.showMessageDialog(null, "Programs with this name
     not found");
1791.                         }
1792.                     }
1793.                 });
1794.                 bSearhProName.setBounds(515, 138, 115, 29);
1795.                 searchProP.add(bSearhProName);
1796.                 
1797.                 bSearhProPhone = new JButton("Search");
1798.                 bSearhProPhone.addActionListener(new ActionListener() {
1799.                     public void actionPerformed(ActionEvent e) {
1800.                         try
1801.                         {
1802.                             int num = Integer.parseInt(textSeaProPhone.getText());
1803.                             data = centre.diaplayPrograms(centre.searchProByPhone(num));
    
1804.                             if(data.length != 0)
1805.                             {
1806.                                 backPro();
1807.                                 refreshProTable(data);
1808.                                 JOptionPane.showMessageDialog(null, "Success! Programs w
    ith this phone number "
1809.                                         + " are listed. Refresh to return to view all Pr
    ograms");
1810.                             }
1811.                             else
1812.                             {
1813.                                 JOptionPane.showMessageDialog(null, "Programs with this
    phone number not found");
1814.                             }
1815.                         }
1816.                         catch(NumberFormatException nfe)
1817.                         {
1818.                             JOptionPane.showMessageDialog(null, "invalid input");
1819.                         }
1820.                     }
1821.                 });
1822.                 bSearhProPhone.setBounds(515, 202, 115, 29);
1823.                 searchProP.add(bSearhProPhone);
1824.                 
1825.                 bSearhProGroup = new JButton("Search");
1826.                 bSearhProGroup.addActionListener(new ActionListener() {
1827.                     public void actionPerformed(ActionEvent e) {
1828.                         try
1829.                         {
1830.                             int num = Integer.parseInt(textSeaProGoup.getText());
1831.                             data = centre.diaplayPrograms(centre.searchProByGroup(num));
    
1832.                             if(data.length != 0)
```

```
1833.                           {
1834.                               backPro();
1835.                               refreshProTable(data);
1836.                               JOptionPane.showMessageDialog(null, "Success! Programs w
    ith this group size "
1837.                                       + " are listed. Refresh to return to view all Pr
    ograms");
1838.                           }
1839.                           else
1840.                           {
1841.                               JOptionPane.showMessageDialog(null, "Programs with this
    group size not found");
1842.                           }
1843.                       }
1844.                       catch(NumberFormatException nfe)
1845.                       {
1846.                           JOptionPane.showMessageDialog(null, "invalid input");
1847.                       }
1848.                   }
1849.               });
1850.               bSearhProGroup.setBounds(515, 268, 115, 29);
1851.               searchProP.add(bSearhProGroup);
1852.
1853.               bSearhProLoc = new JButton("Search");
1854.               bSearhProLoc.addActionListener(new ActionListener() {
1855.                   public void actionPerformed(ActionEvent e) {
1856.                       String loc = textSeaProLoc.getText();
1857.                       data = centre.diaplayPrograms(centre.searchProByLoc(loc));
1858.                       if(data.length != 0)
1859.                       {
1860.                           backPro();
1861.                           refreshProTable(data);
1862.                           JOptionPane.showMessageDialog(null, "Success! Programs with
    this location "
1863.                                   + " are listed. Refresh to return to view all Progra
    ms");
1864.                       }
1865.                       else
1866.                       {
1867.                           JOptionPane.showMessageDialog(null, "Programs with this loca
    tion not found");
1868.                       }
1869.                   }
1870.               });
1871.               bSearhProLoc.setBounds(515, 334, 115, 29);
1872.               searchProP.add(bSearhProLoc);
1873.
1874.               searchRenP = new JPanel();
1875.               frame.getContentPane().add(searchRenP, "name_1256237614725708");
1876.               searchRenP.setLayout(null);
1877.
1878.               label_8 = new JLabel("Contact Name");
1879.               label_8.setBounds(58, 150, 137, 20);
1880.               searchRenP.add(label_8);
1881.
1882.               label_9 = new JLabel("Contact Phone:");
1883.               label_9.setBounds(58, 288, 137, 20);
1884.               searchRenP.add(label_9);
1885.
1886.               textSearchRenName = new JTextField();
1887.               textSearchRenName.setColumns(10);
```

```
1888.               textSearchRenName.setBounds(210, 147, 270, 26);
1889.               searchRenP.add(textSearchRenName);
1890.
1891.               textSearchRenPhone = new JTextField();
1892.               textSearchRenPhone.setColumns(10);
1893.               textSearchRenPhone.setBounds(210, 285, 270, 26);
1894.               searchRenP.add(textSearchRenPhone);
1895.
1896.               bSearchRenBack = new JButton("Back");
1897.               bSearchRenBack.addActionListener(new ActionListener() {
1898.                   public void actionPerformed(ActionEvent e) {
1899.                       backRen();
1900.                   }
1901.               });
1902.               bSearchRenBack.setBounds(281, 449, 115, 29);
1903.               searchRenP.add(bSearchRenBack);
1904.
1905.               bSearchRenName = new JButton("Search");
1906.               bSearchRenName.addActionListener(new ActionListener() {
1907.                   public void actionPerformed(ActionEvent e) {
1908.                       String name = textSearchRenName.getText();
1909.                       data = centre.diaplayRenting(centre.searchRenByName(name));
1910.                       if(data.length != 0)
1911.                       {
1912.                           backRen();
1913.                           refreshRenTable(data);
1914.                           JOptionPane.showMessageDialog(null, "Success! Renting with t
    his name "
1915.                                   + " are listed. Refresh to return to view all Rentin
    g");
1916.                       }
1917.                       else
1918.                       {
1919.                           JOptionPane.showMessageDialog(null, "Renting with this name
    not found");
1920.                       }
1921.                   }
1922.               });
1923.               bSearchRenName.setBounds(513, 146, 115, 29);
1924.               searchRenP.add(bSearchRenName);
1925.
1926.               bSearchRenPhone = new JButton("Search");
1927.               bSearchRenPhone.addActionListener(new ActionListener() {
1928.                   public void actionPerformed(ActionEvent e) {
1929.                       try
1930.                       {
1931.                           int phone = Integer.parseInt(textSearchRenPhone.getText());

1932.                           data = centre.diaplayRenting(centre.searchRenByPhone(phone))
    ;
1933.                           if(data.length != 0)
1934.                           {
1935.                               backRen();
1936.                               refreshRenTable(data);
1937.                               JOptionPane.showMessageDialog(null, "Success! Renting wi
    th this phone number "
1938.                                       + " are listed. Refresh to return to view all Re
    nting");
1939.                           }
1940.                           else
1941.                           {
```

```
1942.                               JOptionPane.showMessageDialog(null, "Renting with this p
    hone number not found");
1943.                           }
1944.                       }
1945.                       catch(NumberFormatException nfe)
1946.                       {
1947.                           JOptionPane.showMessageDialog(null, "invalid input");
1948.                       }
1949.                   }
1950.               });
1951.               bSearchRenPhone.setBounds(513, 284, 115, 29);
1952.               searchRenP.add(bSearchRenPhone);
1953.
1954.               lblSearchRenting = new JLabel("Search Renting");
1955.               lblSearchRenting.setFont(new Font("Tahoma", Font.PLAIN, 36));
1956.               lblSearchRenting.setBounds(43, 38, 330, 55);
1957.               searchRenP.add(lblSearchRenting);
1958.
1959.               searchLofoP = new JPanel();
1960.               frame.getContentPane().add(searchLofoP, "name_1256464314157790");
1961.               searchLofoP.setLayout(null);
1962.
1963.               lblKeyword = new JLabel("Keyword:");
1964.               lblKeyword.setBounds(51, 224, 137, 20);
1965.               searchLofoP.add(lblKeyword);
1966.
1967.               lblLocation_1 = new JLabel("Location:");
1968.               lblLocation_1.setBounds(51, 289, 137, 20);
1969.               searchLofoP.add(lblLocation_1);
1970.
1971.               label_21 = new JLabel("Phone number: ");
1972.               label_21.setBounds(51, 344, 198, 20);
1973.               searchLofoP.add(label_21);
1974.
1975.               lblSearchLostAnd = new JLabel("Search Lost and Found");
1976.               lblSearchLostAnd.setFont(new Font("Tahoma", Font.PLAIN, 36));
1977.               lblSearchLostAnd.setBounds(49, 30, 424, 55);
1978.               searchLofoP.add(lblSearchLostAnd);
1979.
1980.               textSearchLofoKey = new JTextField();
1981.               textSearchLofoKey.setColumns(10);
1982.               textSearchLofoKey.setBounds(203, 221, 270, 26);
1983.               searchLofoP.add(textSearchLofoKey);
1984.
1985.               textSearchLofoLoc = new JTextField();
1986.               textSearchLofoLoc.setColumns(10);
1987.               textSearchLofoLoc.setBounds(203, 286, 270, 26);
1988.               searchLofoP.add(textSearchLofoLoc);
1989.
1990.               textSearchLofoPhone = new JTextField();
1991.               textSearchLofoPhone.setColumns(10);
1992.               textSearchLofoPhone.setBounds(203, 341, 270, 26);
1993.               searchLofoP.add(textSearchLofoPhone);
1994.
1995.               button = new JButton("Back");
1996.               button.addActionListener(new ActionListener() {
1997.                   public void actionPerformed(ActionEvent e) {
1998.                       backlofo();
1999.                   }
2000.               });
2001.               button.setBounds(274, 475, 115, 29);
```

```
2002.                searchLofoP.add(button);
2003.
2004.                lblName = new JLabel("Name:");
2005.                lblName.setBounds(51, 164, 69, 20);
2006.                searchLofoP.add(lblName);
2007.
2008.                textSearchLofoName = new JTextField();
2009.                textSearchLofoName.setColumns(10);
2010.                textSearchLofoName.setBounds(203, 161, 270, 26);
2011.                searchLofoP.add(textSearchLofoName);
2012.
2013.                button_4 = new JButton("Search");
2014.                button_4.addActionListener(new ActionListener() {
2015.                    public void actionPerformed(ActionEvent e) {
2016.                        String name = textSearchLofoName.getText();
2017.                        data = centre.diaplayLofo(centre.searchLofoByName(name));
2018.                        if(data.length != 0)
2019.                        {
2020.                            backlofo();
2021.                            refreshLofoTable(data);
2022.                            JOptionPane.showMessageDialog(null, "Success! lost and found
     with this name "
2023.                                    + " are listed. Refresh to return to view all lost a
    nd found");
2024.                        }
2025.                        else
2026.                        {
2027.                            JOptionPane.showMessageDialog(null, "lost and found with thi
    s name not found");
2028.                        }
2029.                    }
2030.                });
2031.                button_4.setBounds(506, 160, 115, 29);
2032.                searchLofoP.add(button_4);
2033.
2034.                button_7 = new JButton("Search");
2035.                button_7.addActionListener(new ActionListener() {
2036.                    public void actionPerformed(ActionEvent e) {
2037.                        String key = textSearchLofoKey.getText();
2038.                        data = centre.diaplayLofo(centre.searchLofoByKey(key));
2039.                        if(data.length != 0)
2040.                        {
2041.                            backlofo();
2042.                            refreshLofoTable(data);
2043.                            JOptionPane.showMessageDialog(null, "Success! lost and found
     with this Key word in it "
2044.                                    + " are listed. Refresh to return to view all lost a
    nd found");
2045.                        }
2046.                        else
2047.                        {
2048.                            JOptionPane.showMessageDialog(null, "lost and found with thi
    s key word not found");
2049.                        }
2050.                    }
2051.                });
2052.                button_7.setBounds(506, 220, 115, 29);
2053.                searchLofoP.add(button_7);
2054.
2055.                button_14 = new JButton("Search");
2056.                button_14.addActionListener(new ActionListener() {
```

```
2057.                   public void actionPerformed(ActionEvent e) {
2058.                       String loc = textSearchLofoLoc.getText();
2059.                       data = centre.diaplayLofo(centre.searchLofoByLoc(loc));
2060.                       if(data.length != 0)
2061.                       {
2062.                           backlofo();
2063.                           refreshLofoTable(data);
2064.                           JOptionPane.showMessageDialog(null, "Success! lost and found
     with this Location "
2065.                                   + " are listed. Refresh to return to view all lost a
    nd found");
2066.                       }
2067.                       else
2068.                       {
2069.                           JOptionPane.showMessageDialog(null, "lost and found with thi
    s location not found");
2070.                       }
2071.                   }
2072.               });
2073.               button_14.setBounds(506, 280, 115, 29);
2074.               searchLofoP.add(button_14);
2075.
2076.               button_16 = new JButton("Search");
2077.               button_16.addActionListener(new ActionListener() {
2078.                   public void actionPerformed(ActionEvent e) {
2079.                       try
2080.                       {
2081.                           int phone = Integer.parseInt(textSearchLofoPhone.getText());

2082.                           data = centre.diaplayLofo(centre.searchLofoByPhone(phone));

2083.                           if(data.length != 0)
2084.                           {
2085.                               backlofo();
2086.                               refreshLofoTable(data);
2087.                               JOptionPane.showMessageDialog(null, "Success! Lost and f
    ound with this phone number "
2088.                                       + " are listed. Refresh to return to view all Lo
    st and found");
2089.                           }
2090.                           else
2091.                           {
2092.                               JOptionPane.showMessageDialog(null, "Lost and found with
     this phone number not found");
2093.                           }
2094.                       }
2095.                       catch(NumberFormatException nfe)
2096.                       {
2097.                           JOptionPane.showMessageDialog(null, "invalid input");
2098.                       }
2099.                   }
2100.               });
2101.               button_16.setBounds(506, 340, 115, 29);
2102.               searchLofoP.add(button_16);
2103.           }
2104.
2105.           /**
2106.            * refresh the lost and found tables to show new changes
2107.            */
2108.           public void refreshDaiLofoTable(Object[][] data)
2109.           {
```

```
2110.               table = new JTable(data, LOFOTITLE);
2111.               daiListP.add(scrollPane_4);
2112.               scrollPane_4.setViewportView(table);
2113.               table.setDefaultEditor(Object.class, null);
2114.           }
2115.
2116.           /**
2117.            * refresh the membership tables to show new changes in Daily summary
2118.            */
2119.           public void refreshDaiMemTable(Object[][] data)
2120.           {
2121.               table = new JTable(data, MEMTITLE);
2122.               daiListP.add(scrollPane_1);
2123.               scrollPane_1.setViewportView(table);
2124.               table.setDefaultEditor(Object.class, null);
2125.           }
2126.
2127.           /**
2128.            * refresh the rentings tables to show new changes in daily summary
2129.            */
2130.           public void refreshDaiRenTable(Object[][] data)
2131.           {
2132.               table = new JTable(data, RENTITLE);
2133.               daiListP.add(scrollPane_3);
2134.               scrollPane_3.setViewportView(table);
2135.               table.setDefaultEditor(Object.class, null);
2136.           }
2137.           /**
2138.            * refresh the daily summary tables to show new changes
2139.            */
2140.           public void refreshDaiTable()
2141.           {
2142.               //refresh all the other tables plus new members and del membetrs
2143.               refreshDaiMemTable(centre.diaplayMembership());
2144.               refreshDaiRenTable(centre.diaplayRenting());
2145.               refreshDaiLofoTable(centre.diaplayLofo());
2146.
2147.           }
2148.
2149.           /**
2150.            * refresh the lost and found tables to show new changes
2151.            */
2152.           public void refreshLofoTable(Object[][] data)
2153.           {
2154.               table = new JTable(data, LOFOTITLE);
2155.               lofoListP.add(lofoTable);
2156.               lofoTable.setViewportView(table);
2157.               table.setDefaultEditor(Object.class, null);
2158.           }
2159.
2160.           /**
2161.            * refresh the membership tables to show new changes
2162.            */
2163.           public void refreshMemTable(Object[][] data)
2164.           {
2165.               table = new JTable(data, MEMTITLE);
2166.               memListP.add(memTable);
2167.               memTable.setViewportView(table);
2168.               table.setDefaultEditor(Object.class, null);
2169.           }
2170.           /**
```

```
2171.           * refresh the programs tables to show new changes
2172.           */
2173.          public void refreshProTable(Object[][] data)
2174.          {
2175.              table = new JTable(data, PROTITLE);
2176.              proListP.add(proTable);
2177.              proTable.setViewportView(table);
2178.              table.setDefaultEditor(Object.class, null);
2179.          }
2180.
2181.          /**
2182.           * refresh the rentings tables to show new changes
2183.           */
2184.          public void refreshRenTable(Object[][] data)
2185.          {
2186.              table = new JTable(data, RENTITLE);
2187.              renListP.add(scrollPane);
2188.              scrollPane.setViewportView(table);
2189.              lblN.setText(String.valueOf(centre.getAvWheel()));
2190.              lblNewLabel.setText(String.valueOf(centre.getAvStrolls()));
2191.              table.setDefaultEditor(Object.class, null);
2192.          }
2193.
2194.          /**
2195.           * Once a lost and found is added, reset the membership text fields for futu
    re use
2196.           */
2197.          public void resetLofoAdd()
2198.          {
2199.              textField_19.setText("");
2200.              textField_14.setText("");
2201.              textField_18.setText("");
2202.              textField_16.setText("");
2203.              textField_17.setText("");
2204.
2205.          }
2206.
2207.          /**
2208.           * Once a membership is added, reset the membership text fields for future u
    se
2209.           */
2210.          public void resetMemAdd()
2211.          {
2212.              memPhoneT.setText("");
2213.              fMemNameT.setText("");
2214.              lMemNameT.setText("");
2215.              memAdrT.setText("");
2216.
2217.          }
2218.          /**
2219.           * Once a membership is added, reset the renting text fields for future use
2220.           */
2221.          public void resetRenAdd()
2222.          {
2223.              textField_10.setText("");
2224.              textField_11.setText("");
2225.              textField_12.setText("");
2226.          }
2227.          /**
2228.           * an exit where it will inform the client that file will not be saved
2229.           */
```

```
2230.          public void safeExit()
2231.          {
2232.              int a = JOptionPane.showConfirmDialog(null, "Would you like to close? \n
     Your changes will not be saved. "
2233.                      + "To save them, exit with 'Exit and save' button", "Message", J
    OptionPane.YES_NO_OPTION);
2234.              if(a==JOptionPane.YES_OPTION)
2235.              {
2236.                  System.exit(0);
2237.              }
2238.          }
2239.      }
```

```
1.  import java.io.*;
2.  import java.util.*;
3.  /**
4.   * Stores information about memberships in the science centre. It can
5.   * add, delete a member, check out the programs that is happening on the day,
6.   * record rented items, lost and found items, and will be able to print
7.   * out a daily summary of the changes made per visit.
8.   *
9.   *
10.  * @author KaiYuan Chi
11.  * @version 2018-03-07
12.  */
13. public class Centre
14. {
15.
16.     //Constants declaration
17.
18.     private static final int INITIAL_COUNTER = 0;
19.     private static final int LOFO_FIELDS = 7;
20.     private static final int MEMBERSHIP_FIELDS = 6;
21.     private static final int NUMBER_OF_LINES_PER_LOFO = 10;
22.     private static final int NUMBER_OF_LINES_PER_MEM = 9;
23.     private static final int NUMBER_OF_LINES_PER_PRO = 7;
24.     private static final int PROGRAM_FIELDS = 5;
25.     private static final int RENTING_FIELDS = 3;
26.     private static final String STROLLER = "Stroller";
27.     private static final String WHEELCHAIR = "Wheelchair";
28.
29.     //Class field declarations
30.
31.     private static Calendar cal = Calendar.getInstance();
32.     private static int avStrolls;
33.     private static int avWheels;
34.
35.     //instance filed declaration
36.
37.     private ArrayList<Membership> delMems;
38.     private ArrayList<LostFound> lofos;
39.     private ArrayList<Membership> memberships;
40.     private ArrayList<Membership> newMems;
41.     private ArrayList<Program> programs;
42.     private ArrayList<Renting> rentings;
43.
44.     /*
45.      * Static Methods
46.      */
47.
48.     /**
49.      * find the earliest lost and found by using the
50.      * findMinDate recursive method as a helper method
51.      *
52.      * @param nums an integer array of numbers, representing the days
53.      * @param index used to aid the recursion
54.      * @return the minimum number in that array
55.      */
56.     public static int findMinDate(int[] nums, int index)
57.     {
58.         //recursion method
59.         int min = index -1;
60.         if(index < nums.length - 1)
61.         {
```

```
62.             min = findMinDate(nums, index + 1);
63.         }
64.         if(nums[index] < nums[min])
65.         {
66.             min = index;
67.         }
68.         return min;
69.     }
70.     /**
71.      * search membership by adress
72.      *
73.      * @param ad the adress of the target member
74.      * @return the array of membership with that ID, null if not found
75.      */
76.     public static ArrayList<Membership> searchMemByAdress(ArrayList<Membership> members
    hips, String ad)
77.     {
78.         ArrayList<Membership> tars = new ArrayList<>();
79.         boolean include;
80.         for(int i = 0; i < memberships.size(); i++)
81.         {
82.             include = false;
83.             if((memberships.get(i).getAdress().equalsIgnoreCase(ad)))
84.             {
85.                 include = true;
86.             }
87.             if(include)
88.             {
89.                 tars.add(memberships.get(i));
90.             }
91.         }
92.         return tars;
93.     }
94.     /**
95.      * Search membership by holder's last name
96.      *
97.      * @param last target member's last name
98.      * @return an array list of all memberships that contain these names
99.      */
100.           public static ArrayList<Membership> searchMemByLName(ArrayList<Membership> m
    emberships, String last)
101.           {
102.               ArrayList<Membership> tars = new ArrayList<>();
103.               boolean include;
104.               for(int i = 0; i < memberships.size(); i++)
105.               {
106.                   include = false;
107.                   for(int k = 0; k < memberships.get(i).getArAdults().size(); k ++)
108.                   {
109.                       if((memberships.get(i).getArAdults()).get(k).getLastName().equal
    sIgnoreCase(last))
110.                       {
111.                           include = true;
112.                       }
113.                   }
114.                   if(include)
115.                   {
116.                       tars.add(memberships.get(i));
117.                   }
118.               }
119.               return tars;
```

```
120.            }
121.
122.            /*
123.             * non static Methods
124.             */
125.
126.
127.            /**
128.             * Constructs a centre object with a given array of members, programs, renti
    ngs, lost and founds,
129.             * number of wheelchair and strollers in stock.
130.             *
131.             * @param memFile a file with membership information
132.             * @param proFile a file with programs information
133.             * @param lofoFile a file with lost and found information
134.             * @param w number of available wheelchairs
135.             * @param s number of available strollers.
136.             */
137.            public Centre(String memFile, String proFile, String lofoFile, int w, int s)

138.            {
139.                memberships = new ArrayList<Membership>();
140.                programs = new ArrayList<Program>();
141.                lofos = new ArrayList<LostFound>();
142.                rentings = new ArrayList<Renting>();
143.                newMems = new ArrayList<Membership>();
144.                delMems = new ArrayList<Membership>();
145.
146.                //read from file
147.                try {
148.                    this.readMemFile("membershipData.txt");
149.                    this.readProFile(proFile);
150.                    this.readLofoFile(lofoFile);
151.                }
152.                catch(IOException ioe) {
153.                    System.out.println("IOException");
154.                }
155.                avWheels = w;
156.                avStrolls = s;
157.            }
158.
159.
160.
161.            /*
162.             * memberships cannot be sorted by names since
163.             * each membership may have two holders.
164.             */
165.
166.            /**
167.             * Add lost and found from file
168.             *
169.             * @param descrip a general description or note of the item
170.             * @param first the first name of the contact
171.             * @param last the last name of the contact
172.             * @param loc the location of where it is found or lost
173.             * @param l state whether the object is lost
174.             * @param f state whether the object is found.
175.             * @param phonenum the phone number of the contact
176.             */
177.            public void addFileLofo(String descrip, String first,
```

```
178.                    String last, String loc, boolean l, boolean f, int phonenum, int d,
    int m, int y)
179.            {
180.                lofos.add(new LostFound(d, m, y, descrip, first, last, loc, l, f,phonenu
    m));
181.            }
182.            /*
183.             * About display
184.             */
185.
186.            /**
187.             * Add a membership from file
188.             *
189.             * @param phonenum member's phone number
190.             * @param idnum the id of the member
191.             * @param d the expiring day
192.             * @param m the expiring month
193.             * @param y the expiring year
194.             * @param memadress member's adress
195.             * @param memlevelt member'level of membership
196.             * @param first the first name of a member under the membership
197.             * @param last the last name of a member under the membership
198.             */
199.            public void addFileMembership(String first, String last,int idnum, int phone
    num,
200.                    String memadress, String memlevel, int d, int m, int y)
201.            {
202.                memberships.add(new Membership(first, last, idnum, phonenum, memadress,
    memlevel, d, m, y));
203.                newMems.add(new Membership(first, last, idnum, phonenum, memadress, meml
    evel, d, m, y));
204.            }
205.
206.            /**
207.             * Add lost and found
208.             *
209.             * @param descrip a general description or note of the item
210.             * @param first the first name of the contact
211.             * @param last the last name of the contact
212.             * @param loc the location of where it is found or lost
213.             * @param l state whether the object is lost
214.             * @param f state whether the object is found.
215.             * @param phonenum the phone number of the contact
216.             */
217.            public void addLofo(String descrip, String first,
218.                    String last, String loc, boolean l, boolean f, int phonenum)
219.            {
220.                cal = Calendar.getInstance();
221.                int calYear = cal.get(Calendar.YEAR);
222.                int calMonth = cal.get(Calendar.MONTH) + 1;
223.                int calDay = cal.get(Calendar.DAY_OF_MONTH);
224.                lofos.add(new LostFound(calDay, calMonth, calYear, descrip, first, last,
     loc, l, f,phonenum));
225.            }
226.
227.            /**
228.             * Add a membership
229.             *
230.             * @param phonenum member's phone number
231.             * @param memadress member's adress
232.             * @param memlevelt member'level of membership
```

```
233.             * @param first the first name of a member under the membership
234.             * @param last the last name of a member under the membership
235.             */
236.            public void addMembership(String first, String last,int phonenum,
237.                    String memadress, String memlevel)
238.            {
239.                cal = Calendar.getInstance();
240.                int calYear = cal.get(Calendar.YEAR);
241.                int calMonth = cal.get(Calendar.MONTH) + 1;
242.                int calDay = cal.get(Calendar.DAY_OF_MONTH);
243.                memberships.add(new Membership(first, last, phonenum, memadress, memleve
    l, calDay, calMonth, calYear+1));
244.                newMems.add(new Membership(first, last, phonenum, memadress, memlevel, c
    alDay, calMonth, calYear+1));
245.            }
246.
247.            /**
248.             * Add a program from the file
249.             *
250.             * @param start the program's start time
251.             * @param end the program's end time
252.             * @param loc the program's location
253.             * @param gro the size of the group participating the program
254.             * @param first the program's contact's first name
255.             * @param last the program's contact's last name
256.             * @param phonenum the program's contact's phone number
257.             */
258.            public void addProgram(int start, int end, String loc, int gro, String first
    ,
259.                    String last, int phonenum)
260.            {
261.                programs.add(new Program(start, end, loc, gro, first, last, phonenum));

262.            }
263.
264.            /**
265.             * Add a stroller or wheelchair, only allowed when there is one remaining
266.             *
267.             * @param first renter's first name
268.             * @param last renter's last name
269.             * @param phonenum renter's phone number
270.             * @param type the type states wether it is a stroller or a wheelchair
271.             */
272.            public void addRenter(String first, String last, int phonenum, String type)

273.            {
274.                if(type.equalsIgnoreCase("Wheelchair"))
275.                {
276.                    rentings.add(new Renting(first, last , phonenum, "Wheelchair"));
277.                    avWheels = avWheels -1;
278.                }
279.                else
280.                {
281.                    rentings.add(new Renting(first, last , phonenum, "Stroller"));
282.                    avStrolls = avStrolls -1;
283.                }
284.            }
285.
286.            /**
287.              * Delete a certain item
288.              *
```

```
289.              * @param
290.              */
291.             public void deleteLofo(LostFound lofo)
292.             {
293.                 lofos.remove(lofo);
294.             }
295.
296.            /**
297.             * delete a membership by inputing the particular membership.
298.             *
299.             * @param mem the membership which is needed to be removed.
300.             */
301.            public void deleteMembership(Membership mem)
302.            {
303.                memberships.remove(mem);
304.            }
305.
306.            /**
307.             * delete a renting item
308.             *
309.             * @param rent an item
310.             */
311.            public void deleteRenting(Renting ren, String type)
312.            {
313.                if(type == WHEELCHAIR)
314.                {
315.                    rentings.remove(ren);
316.                    avWheels = avWheels +1;
317.                }
318.                else
319.                {
320.                    rentings.remove(ren);
321.                    avStrolls = avStrolls +1;
322.                }
323.            }
324.
325.            /**
326.             * display the summary of all new members added.
327.             *
328.             * @return a 2D array of all the new members information
329.             */
330.            public Object[][] diaplayDelMem()
331.            {
332.                //String[] dmemTitle = {"ID:", "Holders", "Adress", "contact", "Membersh
    ip Level", "Expiring date"}
333.                Object[][] dmem = new Object[delMems.size()][MEMBERSHIP_FIELDS];
334.                Membership member;
335.
336.                for(int i = 0; i < delMems.size(); i ++)
337.                {
338.                    member = delMems.get(i);
339.                    dmem[i][0] = member.getAdults();
340.                    dmem[i][1] = member.getId();
341.                    dmem[i][2] = member.getAdress();
342.                    dmem[i][3] = member.getPhone();
343.                    dmem[i][4] = member.getLevel();
344.                    dmem[i][5] = member.getExpiringDate();
345.                }
346.
347.                return dmem;
348.            }
```

```
349.
350.            /**
351.             * Display lost and found by date
352.             *
353.             * @return a 2D array of all the lost and found information
354.             */
355.            public Object[][] diaplayLofo()
356.            {
357.                this.refreshLofo();
358.                //String[] lofoTitle = {"Date:", "Item Description", Contact", "Contact
    Number:", "Lost?", "Found?", "Location"}
359.                Object[][] lofo = new Object[lofos.size()][LOFO_FIELDS];
360.                LostFound item;
361.
362.                for(int i = 0; i < lofos.size(); i ++)
363.                {
364.                    item = lofos.get(i);
365.                    lofo[i][0] = item.getDate();
366.                    lofo[i][1] = item.getDescrip();
367.                    lofo[i][2] = item.getWholeName();
368.                    lofo[i][3] = item.getPhone();
369.                    lofo[i][4] = item.strLost();
370.                    lofo[i][5] = item.strFound();
371.                    lofo[i][6] = item.getLoc();
372.                }
373.
374.                return lofo;
375.            }
376.
377.            /**
378.             * Display lost and found from the parameter given
379.             *
380.             * @param loFos an array list of lost and found to be displayed
381.             * @return a 2D array of all the lost and found information
382.             */
383.            public Object[][] diaplayLofo(ArrayList<LostFound> loFos)
384.            {
385.                //String[] lofoTitle = {"Date:", "Item Description", Contact", "Contact
    Number:", "Lost?", "Found?", "Location"}
386.                Object[][] lofo = new Object[loFos.size()][LOFO_FIELDS];
387.                LostFound item;
388.
389.                for(int i = 0; i < loFos.size(); i ++)
390.                {
391.                    item = loFos.get(i);
392.                    lofo[i][0] = item.getDate();
393.                    lofo[i][1] = item.getDescrip();
394.                    lofo[i][2] = item.getWholeName();
395.                    lofo[i][3] = item.getPhone();
396.                    lofo[i][4] = item.strLost();
397.                    lofo[i][5] = item.strFound();
398.                    lofo[i][6] = item.getLoc();
399.                }
400.
401.                return lofo;
402.            }
403.
404.            /**
405.             * Display all the memberships
406.             *
407.             * @return a 2D array of all the membership informations
```

```
408.             */
409.            public Object[][] diaplayMembership()
410.            {
411.                //String[] memTitle = {"ID:", "Holders", "Adress", "contact", "Membershi
    p Level", "Expiring date"}
412.                Object[][] mem = new Object[memberships.size()][MEMBERSHIP_FIELDS];
413.                Membership member;
414.
415.                for(int i = 0; i < memberships.size(); i ++)
416.                {
417.                    member = memberships.get(i);
418.                    mem[i][0] = member.getId();
419.                    mem[i][1] = member.getAdults();
420.                    mem[i][2] = member.getAdress();
421.                    mem[i][3] = member.getPhone();
422.                    mem[i][4] = member.getLevel();
423.                    mem[i][5] = member.getExpiringDate();
424.                }
425.
426.                return mem;
427.            }
428.
429.            /**
430.             * Display all the memberships by inputing an array of members.
431.             *
432.             * @param mems an array of memberships that will be displayed in a 2d fashio
    n
433.             * @return a 2D array of all the membership informations
434.             */
435.            public Object[][] diaplayMembership(ArrayList<Membership> mems )
436.            {
437.                //String[] memTitle = {"ID:", "Holders", "Adress", "contact", "Membershi
    p Level", "Expiring date"}
438.                Object[][] mem = new Object[mems.size()][MEMBERSHIP_FIELDS];
439.                Membership member;
440.
441.                for(int i = 0; i < mems.size(); i ++)
442.                {
443.                    member = mems.get(i);
444.                    mem[i][0] = member.getId();
445.                    mem[i][1] = member.getAdults();
446.                    mem[i][2] = member.getAdress();
447.                    mem[i][3] = member.getPhone();
448.                    mem[i][4] = member.getLevel();
449.                    mem[i][5] = member.getExpiringDate();
450.                }
451.
452.                return mem;
453.            }
454.
455.            /**
456.             * display the summary of all new members added.
457.             *
458.             * @return a 2D array of all the new members information
459.             */
460.            public Object[][] diaplayNewMem()
461.            {
462.                //String[] dmemTitle = {"ID:", "Holders", "Adress", "contact", "Membersh
    ip Level", "Expiring date"}
463.                Object[][] nmem = new Object[newMems.size()][MEMBERSHIP_FIELDS];
464.                Membership member;
```

```
465.
466.                 for(int i = 0; i < newMems.size(); i ++)
467.                 {
468.                     member = newMems.get(i);
469.                     nmem[i][0] = member.getAdults();
470.                     nmem[i][1] = member.getId();
471.                     nmem[i][2] = member.getAdress();
472.                     nmem[i][3] = member.getPhone();
473.                     nmem[i][4] = member.getLevel();
474.                     nmem[i][5] = member.getExpiringDate();
475.                 }
476.
477.                 return nmem;
478.             }
479.
480.             /**
481.              * Display all the programs.
482.              *
483.              * @return a 2D array of all the daily programs on that day
484.              */
485.             public Object[][] diaplayPrograms()
486.             {
487.                 //String[] proitle = {"Time:", "Location:", "Group size:", "Contact", "C
    ontact Phone"}
488.                 //should also sort the program
489.                 Object[][] pro = new Object[programs.size()][PROGRAM_FIELDS];
490.                 Program program;
491.
492.                 for(int i = 0; i < programs.size(); i ++)
493.                 {
494.                     program = programs.get(i);
495.                     pro[i][0] = program.displayTimePeriod();
496.                     pro[i][1] = program.getLocation();
497.                     pro[i][2] = program.getGroupSize();
498.                     pro[i][3] = program.getWholeName();
499.                     pro[i][4] = program.getPhone();
500.                 }
501.
502.                 return pro;
503.             }
504.
505.             /**
506.              * Display all the programs that is given by the parameter, used specificall
    y for searching
507.              *
508.              * @param pros an array of programs
509.              * @return a 2D array of all the daily programs on that day
510.              */
511.             public Object[][] diaplayPrograms(ArrayList<Program> pros)
512.             {
513.                 //String[] proitle = {"Time:", "Location:", "Group size:", "Contact", "C
    ontact Phone"}
514.                 //should also sort the program
515.                 Object[][] pro = new Object[pros.size()][PROGRAM_FIELDS];
516.                 Program program;
517.
518.                 for(int i = 0; i < pros.size(); i ++)
519.                 {
520.                     program = pros.get(i);
521.                     pro[i][0] = program.displayTimePeriod();
522.                     pro[i][1] = program.getLocation();
```

```
523.                     pro[i][2] = program.getGroupSize();
524.                     pro[i][3] = program.getWholeName();
525.                     pro[i][4] = program.getPhone();
526.                 }
527.
528.                 return pro;
529.             }
530.
531.             /**
532.              * display all the rentings.
533.              *
534.              * @return a 2D array of all the renting informations
535.              */
536.             public Object[][] diaplayRenting()
537.             {
538.                 //String[] renTitle = {"Renting Item:", "Renter:", "Contact number:"}
539.                 Object[][] ren = new Object[rentings.size()][RENTING_FIELDS];
540.                 Renting renn;
541.
542.                 for(int i = 0; i < rentings.size(); i ++)
543.                 {
544.                     renn = rentings.get(i);
545.                     ren[i][0] = renn.getType();
546.                     ren[i][1] = renn.getWholeName();
547.                     ren[i][2] = renn.getPhone();
548.                 }
549.
550.                 return ren;
551.             }
552.
553.             /**
554.              * display all the rentings from the given arrayList.
555.              *
556.              * @param rens a given arrayList of rentings
557.              * @return a 2D array of all the renting informations
558.              */
559.             public Object[][] diaplayRenting(ArrayList<Renting> rens)
560.             {
561.                 //String[] renTitle = {"Renting Item:", "Renter:", "Contact number:"}
562.                 Object[][] ren = new Object[rens.size()][RENTING_FIELDS];
563.
564.                 for(int i = 0; i < rens.size(); i ++)
565.                 {
566.                     Renting renn = rens.get(i);
567.                     ren[i][0] = renn.getType();
568.                     ren[i][1] = renn.getWholeName();
569.                     ren[i][2] = renn.getPhone();
570.                 }
571.
572.                 return ren;
573.             }
574.
575.             /**
576.              * Return the number of available stroller
577.              *
578.              * @return the number of available stroller
579.              */
580.             public int getAvStrolls()
581.             {
582.                 return avStrolls;
583.             }
```

```
584.
585.            /**
586.             * Return the number of available wheelchair
587.             *
588.             * @return the number of available wheelchair
589.             */
590.            public int getAvWheel()
591.            {
592.                return avWheels;
593.            }
594.            /**
595.             * return a lost and found item from its index
596.             *
597.             * @return a lost and found item from its index
598.             */
599.             public LostFound getLofoIndex(int index)
600.             {
601.                 return lofos.get(index);
602.             }
603.
604.            /**
605.             * Return the member at that index
606.             *
607.             * @param index an integer representing the index of a person
608.             * @return a member at that index
609.             */
610.            public Membership getMemIndex(int index)
611.            {
612.                return memberships.get(index);
613.            }
614.
615.            /**
616.             * Return the Memberships arrayList
617.             *
618.             * @return the Memberships arrayList
619.             */
620.            public ArrayList<Membership> getMemList()
621.            {
622.                return memberships;
623.            }
624.
625.            /**
626.             * Return an array of programs
627.             *
628.             * @return an array of programs
629.             */
630.            public ArrayList<Program> getProList()
631.            {
632.                return programs;
633.            }
634.
635.            /**
636.             * Return a renting from the index
637.             *
638.             * @param index the index which the renting item is at
639.             * @return a renting from the index
640.             */
641.            public Renting getRenIndex(int index)
642.            {
643.                return rentings.get(index);
644.            }
```

```
645.
646.            /**
647.             * Check if file exist
648.             *
649.             * @param fileName the name of the file
650.             * @return true if the file is found, false if the file is not found.
651.             */
652.            public boolean isFileValid(String fileName)
653.            {
654.                boolean fileFound = true;
655.                fileFound = true;
656.                try
657.                {
658.                    BufferedReader inputFile = new BufferedReader(new FileReader(fileNam
    e));
659.
660.                }
661.                catch(FileNotFoundException e)
662.                {
663.                    fileFound = false;
664.                }
665.                return fileFound;
666.            }
667.
668.
669.            /**
670.             * Read Lost and Found File
671.             *
672.             * @param fileName the name of the file
673.             */
674.            public void readLofoFile(String fileName) throws IOException //exception cap
    tured by other classes
675.            {
676.                BufferedReader inputFile = new BufferedReader(new FileReader(fileName));

677.                //declare temporary local variables
678.                String lineOfText = inputFile.readLine();
679.                int counter = 1;
680.                int Index = 0;
681.                String descrip= null;
682.                String first = null;
683.                String last = null;
684.                String loc = null;
685.                boolean l = false;
686.                boolean f = false;
687.                int phone = 0;
688.                int day = 0;
689.                int month = 0;
690.                int year = 0;
691.
692.                while(lineOfText != null)
693.                {
694.                    //read lines, all data are correct
695.                    if(counter%NUMBER_OF_LINES_PER_LOFO == 1)
696.                    {
697.                        descrip = lineOfText;
698.                    }
699.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 2)
700.                    {
701.                        first = lineOfText;
702.                    }
```

```
703.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 3)
704.                    {
705.                        last = lineOfText;
706.                    }
707.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 4)
708.                    {
709.                        loc = lineOfText;
710.                    }
711.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 5)
712.                    {
713.                        if(lineOfText.equalsIgnoreCase("true"))
714.                        {
715.                            l = true;
716.                        }
717.                        else
718.                        {
719.                            l = false;
720.                        }
721.                    }
722.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 6)
723.                    {
724.                        if(lineOfText.equalsIgnoreCase("true"))
725.                        {
726.                            f = true;
727.                        }
728.                        else
729.                        {
730.                            f = false;
731.                        }
732.                    }
733.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 7)
734.                    {
735.                        phone = Integer.parseInt(lineOfText);
736.
737.                    }
738.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 8)
739.                    {
740.                        day = Integer.parseInt(lineOfText);
741.                    }
742.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 9)
743.                    {
744.                        month = Integer.parseInt(lineOfText);
745.                    }
746.                    else if(counter%NUMBER_OF_LINES_PER_LOFO == 0)
747.                    {
748.                        year = Integer.parseInt(lineOfText);
749.                        this.addFileLofo(descrip, first, last, loc, l, f, phone, day, mo
    nth, year);
750.                    }
751.                    counter = counter + 1;
752.                    lineOfText = inputFile.readLine();
753.                }//end while
754.
755.                inputFile.close();
756.            }
757.
758.            /**
759.             * Read Membership File
760.             *
761.             * @param fileName the name of the file
762.             */
```

```
763.            public void readMemFile(final String fileName) throws IOException //exceptio
    n captured by other classes
764.            {
765.                BufferedReader inputFile = new BufferedReader(new FileReader(fileName));

766.                //declare temporary local variables
767.                String lineOfText = inputFile.readLine();
768.                int counter = 1;
769.                int memIndex = 0;
770.                int id = 0;
771.                String adress = null;
772.                int phone = 0;
773.                String level = null;
774.                String first = null;
775.                String last = null;
776.                int day = 0;
777.                int month = 0;
778.                int year = 0;
779.
780.                while(lineOfText != null)
781.                {
782.                    //read lines, all data are correct
783.                    if(counter% NUMBER_OF_LINES_PER_MEM == 1)
784.                    {
785.                        id = Integer.parseInt(lineOfText);
786.                    }
787.                    else if(counter% NUMBER_OF_LINES_PER_MEM == 2)
788.                    {
789.                        adress = lineOfText;
790.                    }
791.                    else if(counter% NUMBER_OF_LINES_PER_MEM == 3)
792.                    {
793.                        phone = Integer.parseInt(lineOfText);
794.                    }
795.                    else if(counter% NUMBER_OF_LINES_PER_MEM == 4)
796.                    {
797.                        level = lineOfText;
798.                    }
799.                    else if(counter%NUMBER_OF_LINES_PER_MEM == 5)
800.                    {
801.                        first = lineOfText;
802.                    }
803.                    else if(counter%NUMBER_OF_LINES_PER_MEM == 6)
804.                    {
805.                        last = lineOfText;
806.                    }
807.                    else if(counter%NUMBER_OF_LINES_PER_MEM == 7)
808.                    {
809.                        day = Integer.parseInt(lineOfText);
810.                    }
811.                    else if(counter%NUMBER_OF_LINES_PER_MEM == 8)
812.                    {
813.                        month = Integer.parseInt(lineOfText);
814.                    }
815.                    else if(counter%NUMBER_OF_LINES_PER_MEM == 0)
816.                    {
817.                        year = Integer.parseInt(lineOfText);
818.                        memberships.add(new Membership(first, last, id, phone, adress, l
    evel, day, month, year));
819.                    }
820.                    counter = counter + 1;
```

```
821.                     lineOfText = inputFile.readLine();
822.                 }//end while
823.
824.                 inputFile.close();
825.             }
826.
827.
828.             /**
829.              * Read program File
830.              *
831.              * @param fileName the name of the file
832.              */
833.             public void readProFile(String fileName) throws IOException //exception capt
    ured by other classes
834.             {
835.                 BufferedReader inputFile = new BufferedReader(new FileReader(fileName));

836.                 //declare temporary local variables
837.                 String lineOfText = inputFile.readLine();
838.                 int counter = 1;
839.                 int Index = 0;
840.                 int start = 0;
841.                 int end = 0;
842.                 String loc = null;
843.                 int gro = 0;
844.                 String first = null;
845.                 String last = null;
846.                 int phone = 0;
847.
848.                 while(lineOfText != null)
849.                 {
850.                     //read lines, all data are correct
851.                     if(counter%NUMBER_OF_LINES_PER_PRO == 1)
852.                     {
853.                         start = Integer.parseInt(lineOfText);
854.                     }
855.                     else if(counter%NUMBER_OF_LINES_PER_PRO == 2)
856.                     {
857.                         end = Integer.parseInt(lineOfText);
858.                     }
859.                     else if(counter%NUMBER_OF_LINES_PER_PRO == 3)
860.                     {
861.                         loc = lineOfText;
862.                     }
863.                     else if(counter%NUMBER_OF_LINES_PER_PRO == 4)
864.                     {
865.                         gro = Integer.parseInt(lineOfText);
866.                     }
867.                     else if(counter%NUMBER_OF_LINES_PER_PRO == 5)
868.                     {
869.                         first = lineOfText;
870.                     }
871.                     else if(counter%NUMBER_OF_LINES_PER_PRO == 6)
872.                     {
873.                         last = lineOfText;
874.                     }
875.                     else if(counter%NUMBER_OF_LINES_PER_PRO == 0)
876.                     {
877.                         phone = Integer.parseInt(lineOfText);
878.                         this.addProgram(start, end, loc, gro, first, last, phone);
879.                     }
```

```
880.                     counter = counter + 1;
881.                     lineOfText = inputFile.readLine();
882.                 }//end while
883.
884.                 inputFile.close();
885.             }
886.
887.             /**
888.              * Refresh lost and found list, remove every item that is both lost
889.              * and found, remove any item that is over a month.
890.              */
891.             public void refreshLofo()
892.             {
893.                 for(int i = 0; i < lofos.size(); i ++)
894.                 {
895.                     if(lofos.get(i).isPaired() || lofos.get(i).isOver(lofos.get(i).getYe
    ar(), lofos.get(i).getMonth(), lofos.get(i).getDay()))
896.                     {
897.                         lofos.remove(lofos.get(i));
898.                     }
899.                 }
900.             }
901.
902.             /**
903.              * search lost and found by date
904.              *
905.              * @param y the year of the item lost/found
906.              * @param m the month of the item lost/found
907.              * @param d the day that the item was lost/found
908.              * @return an array list of lost and found that was lost on this date
909.              */
910.             public ArrayList<LostFound> searchLofoByDate(int y, int m, int d)
911.             {
912.                 ArrayList<LostFound> tars = new ArrayList<>();
913.                 boolean include;
914.                 for(int i = 0; i < lofos.size(); i ++)
915.                 {
916.                     include = false;
917.                     if(lofos.get(i).getYear() == y && lofos.get(i).getMonth() == m && lo
    fos.get(i).getDay() == d)
918.                     {
919.                         include = true;
920.                     }
921.                     if(include)
922.                     {
923.                         tars.add(lofos.get(i));
924.                     }
925.                 }
926.                 return tars;
927.             }
928.
929.             /**
930.              * search by description key word
931.              *
932.              * @param str the keyword that is wished to look for
933.              * @return the arraylist of all items with that keyword in description
934.              */
935.             public ArrayList<LostFound> searchLofoByKey(String str)
936.             {
937.                 ArrayList<LostFound> tars = new ArrayList<>();
938.                 boolean include;
```

```
939.                for(int i = 0; i < lofos.size(); i ++)
940.                {
941.                    include = false;
942.                    String des = lofos.get(i).getDescrip();
943.                    String[] s = des.split(" ");
944.                    for(int k = 0; k < s.length; k++)
945.                    {
946.                        if(s[k].equalsIgnoreCase(str))
947.                        {
948.                            include = true;
949.                        }
950.                    }
951.
952.                    if(include)
953.                    {
954.                        tars.add(lofos.get(i));
955.                    }
956.                }
957.                return tars;
958.            }
959.
960.            /**
961.             * search Lost and Found by location
962.             *
963.             * @param loc the location of the target member
964.             * @return the array of locations, null if not found
965.             */
966.            public ArrayList<LostFound> searchLofoByLoc(String loc)
967.            {
968.                ArrayList<LostFound> tars = new ArrayList<>();
969.                boolean include;
970.                for(int i = 0; i < lofos.size(); i++)
971.                {
972.                    include = false;
973.                    if((lofos.get(i).getLoc().equalsIgnoreCase(loc)))
974.                    {
975.                        include = true;
976.                    }
977.                    if(include)
978.                    {
979.                        tars.add(lofos.get(i));
980.                    }
981.                }
982.                return tars;
983.            }
984.
985.            /**
986.             * Search lost and found by name
987.             *
988.             * @param name the name of the target member
989.             * @return the array of Rentings, null if not found
990.             */
991.            public ArrayList<LostFound> searchLofoByName(String name)
992.            {
993.                ArrayList<LostFound> tars = new ArrayList<>();
994.                boolean include;
995.                for(int i = 0; i < lofos.size(); i++)
996.                {
997.                    include = false;
998.                    if((lofos.get(i).getWholeName().equalsIgnoreCase(name)))
999.                    {
```

```
1000.                       include = true;
1001.                   }
1002.                   if(include)
1003.                   {
1004.                       tars.add(lofos.get(i));
1005.                   }
1006.               }
1007.               return tars;
1008.           }
1009.
1010.
1011.           /**
1012.            * search Lost and found by contact
1013.            *
1014.            * @param num the phone number of the target member
1015.            * @return an array list of lost and found that contain these phone numbers
1016.            */
1017.           public ArrayList<LostFound> searchLofoByPhone(int num)
1018.           {
1019.               ArrayList<LostFound> tars = new ArrayList<>();
1020.               boolean include;
1021.               for(int i = 0; i < lofos.size(); i++)
1022.               {
1023.                   include = false;
1024.                   if((lofos.get(i).getPhone() == num))
1025.                   {
1026.                       include = true;
1027.                   }
1028.                   if(include)
1029.                   {
1030.                       tars.add(lofos.get(i));
1031.                   }
1032.               }
1033.               return tars;
1034.           }
1035.
1036.           /**
1037.            * main search, which combinding all searches
1038.            *
1039.            * @return a 2D array of ojects that satisfy these requirements.
1040.            */
1041.           public ArrayList<Membership> searchmain(int id, String first, String last, i
    nt phonenum, String adress)
1042.           {
1043.               ArrayList<Membership> tar = new ArrayList<>();
1044.               ArrayList<Membership> temp = new ArrayList<>();
1045.                   if(id != 0)
1046.                   {
1047.                       tar = new ArrayList<>();
1048.                       for(int i = 0; i < this.searchMemById(id).size(); i ++)
1049.                       {
1050.                           tar.add(this.searchMemById(id).get(i));
1051.                       }
1052.                   }
1053.                   if(first != null)
1054.                   {
1055.
1056.
1057.                       tar = searchMemByFName(tar, first);
1058.                   }
1059.                   if(last != null)
```

```
1060.                   {
1061.                       tar = searchMemByLName(tar, last);
1062.                   }
1063.                   if(phonenum != 0)
1064.                   {
1065.                       tar = searchMemPhone(tar, phonenum);
1066.                   }
1067.                   if(adress != null)
1068.                   {
1069.                       tar = searchMemByAdress(tar, adress);
1070.                   }
1071.               return tar;
1072.           }
1073.
1074.           /**
1075.            * Search membership by holder's first name
1076.            *
1077.            * @param first target member's first name
1078.            * @return an array list of all memberships that contain these names
1079.            */
1080.           public ArrayList<Membership> searchMemByFName(ArrayList<Membership> membersh
    ips, String first)
1081.           {
1082.               ArrayList<Membership> tars = new ArrayList<>();
1083.               boolean include;
1084.               for(int i = 0; i < memberships.size(); i++)
1085.               {
1086.                   include = false;
1087.                   for(int k = 0; k < memberships.get(i).getArAdults().size(); k ++)
1088.                   {
1089.                       if((memberships.get(i).getArAdults()).get(k).getFirstName().equa
    lsIgnoreCase(first))
1090.                       {
1091.                           include = true;
1092.                       }
1093.                   }
1094.                   if(include)
1095.                   {
1096.                       tars.add(memberships.get(i));
1097.                   }
1098.               }
1099.               return tars;
1100.           }
1101.
1102.           /**
1103.            * Search membership by ID
1104.            *
1105.            * @param num the ID number of the target member
1106.            * @return the membership with that ID, null if not found
1107.            */
1108.           public ArrayList<Membership> searchMemById(int id)
1109.           {
1110.               //Perform Binary Search, since every Id is different.
1111.               ArrayList<Membership> mems = new ArrayList<>();
1112.
1113.               this.sortMemById();
1114.               int bottom = INITIAL_COUNTER;
1115.               int top = memberships.size() -1;
1116.               int middle;
1117.               boolean found = false;
1118.               int location = -1;
```

```
1119.
1120.               while(bottom <= top && !found)
1121.               {
1122.                   middle = (bottom + top)/2;
1123.                   if(memberships.get(middle).getId() == id)
1124.                   {
1125.                       found = true;
1126.                       location = middle;
1127.                   }
1128.                   else if(memberships.get(middle).getId() < id)
1129.                   {
1130.                       bottom = middle+1;
1131.                   }
1132.                   else
1133.                   {
1134.                       top = middle -1;
1135.                   }
1136.               }
1137.               if(location != -1)
1138.               {
1139.                   mems.add(memberships.get(location));
1140.               }
1141.
1142.               return mems;
1143.           }
1144.
1145.           /*
1146.            * About lostFound
1147.            */
1148.
1149.           /**
1150.            * search membership by contact
1151.            *
1152.            * @param num the phone of the target member
1153.            * @return an array list of all memberships that contain these p
1154.            */
1155.           public ArrayList<Membership> searchMemPhone(ArrayList<Membership> membership
    s, int num)
1156.           {
1157.               ArrayList<Membership> tars = new ArrayList<>();
1158.               boolean include;
1159.               for(int i = 0; i < memberships.size(); i++)
1160.               {
1161.                   include = false;
1162.                   if((memberships.get(i).getPhone() == num))
1163.                   {
1164.                       include = true;
1165.                   }
1166.                   if(include)
1167.                   {
1168.                       tars.add(memberships.get(i));
1169.                   }
1170.               }
1171.               return tars;
1172.           }
1173.
1174.           /**
1175.            * Search program by group
1176.            *
1177.            * @param num the group size of the target member
1178.            * @return an array list of all programs that have the group size
```

```
1179.          */
1180.         public ArrayList<Program> searchProByGroup(int num)
1181.         {
1182.             ArrayList<Program> tars = new ArrayList<>();
1183.             boolean include;
1184.             for(int i = 0; i < programs.size(); i++)
1185.             {
1186.                 include = false;
1187.                 if((programs.get(i).getGroupSize() == num))
1188.                 {
1189.                     include = true;
1190.                 }
1191.                 if(include)
1192.                 {
1193.                     tars.add(programs.get(i));
1194.                 }
1195.             }
1196.             return tars;
1197.         }
1198.
1199.         /**
1200.          * search program by location
1201.          *
1202.          * @param loc the location of the target member
1203.          * @return the array of all programs at that location
1204.          */
1205.         public ArrayList<Program> searchProByLoc(String loc)
1206.         {
1207.             ArrayList<Program> tars = new ArrayList<>();
1208.             boolean include;
1209.             for(int i = 0; i < programs.size(); i++)
1210.             {
1211.                 include = false;
1212.                 if((programs.get(i).getLocation().equalsIgnoreCase(loc)))
1213.                 {
1214.                     include = true;
1215.                 }
1216.                 if(include)
1217.                 {
1218.                     tars.add(programs.get(i));
1219.                 }
1220.             }
1221.             return tars;
1222.         }
1223.
1224.          /**
1225.          * search programs by Whole name (First, last)
1226.          *
1227.          * @param name the neame of the target member
1228.          * @return the array of programs, null if not found
1229.          */
1230.         public ArrayList<Program> searchProByName(String name)
1231.         {
1232.             ArrayList<Program> tars = new ArrayList<>();
1233.             boolean include;
1234.             for(int i = 0; i < programs.size(); i++)
1235.             {
1236.                 include = false;
1237.                 if((programs.get(i).getWholeName().equalsIgnoreCase(name)))
1238.                 {
1239.                     include = true;
```

```
1240.                   }
1241.                   if(include)
1242.                   {
1243.                       tars.add(programs.get(i));
1244.                   }
1245.               }
1246.               return tars;
1247.           }
1248.           /**
1249.            * search program by contact
1250.            *
1251.            * @param num the phone number of the target member
1252.            * @return an array list of all memberships that contain these phone numbers
1253.            */
1254.           public ArrayList<Program> searchProByPhone(int num)
1255.           {
1256.               ArrayList<Program> tars = new ArrayList<>();
1257.               boolean include;
1258.               for(int i = 0; i < programs.size(); i++)
1259.               {
1260.                   include = false;
1261.                   if((programs.get(i).getPhone() == num))
1262.                   {
1263.                       include = true;
1264.                   }
1265.                   if(include)
1266.                   {
1267.                       tars.add(programs.get(i));
1268.                   }
1269.               }
1270.               return tars;
1271.           }
1272.
1273.           /**
1274.            * Search a program by start time
1275.            *
1276.            * @param time the time of the target member
1277.            * @return the array of programs at that location
1278.            */
1279.           public ArrayList<Program> searchProByTime(int time)
1280.           {
1281.               ArrayList<Program> tars = new ArrayList<>();
1282.               boolean include;
1283.               for(int i = 0; i < programs.size(); i++)
1284.               {
1285.                   include = false;
1286.                   if((programs.get(i).getStartTime() == time))
1287.                   {
1288.                       include = true;
1289.                   }
1290.                   if(include)
1291.                   {
1292.                       tars.add(programs.get(i));
1293.                   }
1294.               }
1295.               return tars;
1296.           }
1297.
1298.           /**
1299.            * search Renting by Whole name (First, last)
```

```
1300.           *
1301.           * @param name the name of the target member
1302.           * @return the array of Rentings, null if not found
1303.           */
1304.          public ArrayList<Renting> searchRenByName(String name)
1305.          {
1306.              ArrayList<Renting> tars = new ArrayList<>();
1307.              boolean include;
1308.              for(int i = 0; i < rentings.size(); i++)
1309.              {
1310.                  include = false;
1311.                  if((rentings.get(i).getWholeName().equalsIgnoreCase(name)))
1312.                  {
1313.                      include = true;
1314.                  }
1315.                  if(include)
1316.                  {
1317.                      tars.add(rentings.get(i));
1318.                  }
1319.              }
1320.              return tars;
1321.          }
1322.
1323.          /**
1324.           * search rent by contact number
1325.           *
1326.           * @param num the phone number of the target member
1327.           * @return an array list of all rentings that contain these phone numbers
1328.           */
1329.          public ArrayList<Renting> searchRenByPhone(int num)
1330.          {
1331.              ArrayList<Renting> tars = new ArrayList<>();
1332.              boolean include;
1333.              for(int i = 0; i < rentings.size(); i++)
1334.              {
1335.                  include = false;
1336.                  if((rentings.get(i).getPhone() == num))
1337.                  {
1338.                      include = true;
1339.                  }
1340.                  if(include)
1341.                  {
1342.                      tars.add(rentings.get(i));
1343.                  }
1344.              }
1345.              return tars;
1346.          }
1347.
1348.          /**
1349.           * sort lost and found by date with the latest first and earliest last
1350.           */
1351.          public void sortLofoByDate()
1352.          {
1353.              //since all losts and founds are in less than a month
1354.              int[] days = new int[lofos.size()];
1355.              for(int i = 0; i < lofos.size(); i ++)
1356.              {
1357.                  days[i] = lofos.get(i).getDays();
1358.              }
1359.              for(int i = 0; i < lofos.size(); i++)
1360.              {
```

```
1361.                   LostFound temp = lofos.get(i);
1362.                   lofos.set(i, lofos.get(findMinDate(days, i)));
1363.                   lofos.set(findMinDate(days,i), temp);
1364.               }
1365.           }
1366.
1367.           /**
1368.            * sort lost and found by description
1369.            */
1370.           public void sortLofoByDescrip()
1371.           {
1372.               //selection sort
1373.               for(int i = 0; i < lofos.size()-1; i ++)
1374.               {
1375.                   int min = i;
1376.                   for(int j = i+1; j < lofos.size(); j ++)
1377.                   {
1378.                       if((lofos.get(j).getDescrip()).compareToIgnoreCase(lofos.get(min
    ).getDescrip()) < 0)
1379.                       {
1380.                           min = j;
1381.                       }
1382.                   }
1383.                   //switch
1384.                   LostFound temp = lofos.get(i);
1385.                   lofos.set(i, lofos.get(min));
1386.                   lofos.set(min, temp);
1387.               }
1388.           }
1389.
1390.           /**
1391.            * sort lost and found by locaton
1392.            */
1393.           public void sortLofoByLoc()
1394.           {
1395.               //selection sort
1396.               for(int i = 0; i < lofos.size()-1; i ++)
1397.               {
1398.                   int min = i;
1399.                   for(int j = i+1; j < lofos.size(); j ++)
1400.                   {
1401.                       if((lofos.get(j).getLoc()).compareToIgnoreCase(lofos.get(min).ge
    tLoc()) < 0)
1402.                       {
1403.                           min = j;
1404.                       }
1405.                   }
1406.                   //switch
1407.                   LostFound temp = lofos.get(i);
1408.                   lofos.set(i, lofos.get(min));
1409.                   lofos.set(min, temp);
1410.               }
1411.           }
1412.
1413.           /**
1414.            * Sort lost and found by name
1415.            */
1416.           public void sortLofoByName()
1417.           {
1418.               //selection sort
1419.               for(int i = 0; i < lofos.size()-1; i ++)
```

```
1420.               {
1421.                   int min = i;
1422.                   for(int j = i+1; j < lofos.size(); j ++)
1423.                   {
1424.                       if((lofos.get(j).getWholeName()).compareToIgnoreCase(lofos.get(m
    in).getWholeName()) < 0)
1425.                       {
1426.                           min = j;
1427.                       }
1428.                   }
1429.                   //switch
1430.                   LostFound temp = lofos.get(i);
1431.                   lofos.set(i, lofos.get(min));
1432.                   lofos.set(min, temp);
1433.               }
1434.           }
1435.
1436.           /**
1437.            * sort lost and found by contact number
1438.            */
1439.           public void sortLofoByPhone()
1440.           {
1441.               //selection sort
1442.               for(int i = 0; i < lofos.size()-1; i ++)
1443.               {
1444.                   int min = i;
1445.                   for(int j = i+1; j < lofos.size(); j ++)
1446.                   {
1447.                       if(lofos.get(j).getPhone() < lofos.get(min).getPhone())
1448.                       {
1449.                           min = j;
1450.                       }
1451.                   }
1452.                   //switch
1453.                   LostFound temp = lofos.get(i);
1454.                   lofos.set(i,lofos.get(min));
1455.                   lofos.set(min, temp);
1456.               }
1457.           }
1458.
1459.
1460.           /**
1461.            * sort lost and found by type
1462.            */
1463.           public void sortLofoByType()
1464.           {
1465.               this.refreshLofo();
1466.               ArrayList<LostFound> losts = new ArrayList<LostFound>();
1467.               ArrayList<LostFound> founds = new ArrayList<LostFound>();
1468.               for(int i = 0; i < lofos.size(); i++)
1469.               {
1470.                   if(lofos.get(i).isLost())
1471.                   {
1472.                       losts.add(lofos.get(i));
1473.                   }
1474.                   else
1475.                   {
1476.                       founds.add(lofos.get(i));
1477.                   }
1478.               }
1479.               for(int i = 0; i < losts.size(); i++)
```

```
1480.               {
1481.                   lofos.set(i, losts.get(i));
1482.               }
1483.               int counter = INITIAL_COUNTER;
1484.               for(int i = losts.size(); i < lofos.size(); i++)
1485.               {
1486.                   lofos.set(i, founds.get(counter));
1487.                   counter++;
1488.               }
1489.           }
1490.
1491.           /**
1492.            * Sort by the membership's adress
1493.            */
1494.           public void sortMemByAdress()
1495.           {
1496.               //selection sort
1497.               for(int i = 0; i < memberships.size()-1;i++)
1498.               {
1499.                   int min = i;
1500.                   for(int j = i + 1; j < memberships.size(); j++)
1501.                   {
1502.                       if((memberships.get(j).getAdress()).compareToIgnoreCase(membersh
    ips.get(min).getAdress()) < 0)
1503.                       {
1504.                           min = j;
1505.                       }
1506.                   }
1507.                   //switch
1508.                   Membership temp = memberships.get(i);
1509.                   memberships.set(i, memberships.get(min));
1510.                   memberships.set(min, temp);
1511.               }
1512.           }
1513.
1514.           /**
1515.            * sort by the first holder in the holder list of a membership
1516.            */
1517.           public void sortMemByHolder()
1518.           {
1519.               //selection sort
1520.               for(int i = 0; i < memberships.size()-1;i++)
1521.               {
1522.                   int min = i;
1523.                   for(int j = i + 1; j < memberships.size(); j++)
1524.                   {
1525.                       if((memberships.get(j).getAdults()).compareToIgnoreCase(membersh
    ips.get(min).getAdults()) < 0)
1526.                       {
1527.                           min = j;
1528.                       }
1529.                   }
1530.                   //switch
1531.                   Membership temp = memberships.get(i);
1532.                   memberships.set(i, memberships.get(min));
1533.                   memberships.set(min, temp);
1534.               }
1535.           }
1536.
1537.           /**
1538.            * Sort all the memberships by ID
```

```
1539.           */
1540.          public void sortMemById()
1541.          {
1542.              //selection sort, since there will not be repeating ids
1543.              for(int i = 0; i < memberships.size()-1; i ++)
1544.              {
1545.                  int min = i;
1546.                  for(int j = i+1; j < memberships.size(); j ++)
1547.                  {
1548.                      if(memberships.get(j).getId() < memberships.get(min).getId())
1549.                      {
1550.                          min = j;
1551.                      }
1552.                  }
1553.                  //switch
1554.                  Membership temp = memberships.get(i);
1555.                  memberships.set(i, memberships.get(min));
1556.                  memberships.set(min, temp);
1557.              }
1558.          }
1559.
1560.          /**
1561.           * Sort membership by phone number
1562.           */
1563.          public void sortMemByPhone()
1564.          {
1565.              //selection sort
1566.              for(int i = 0; i < memberships.size()-1; i ++)
1567.              {
1568.                  int min = i;
1569.                  for(int j = i+1; j < memberships.size(); j ++)
1570.                  {
1571.                      if(memberships.get(j).getPhone() < memberships.get(min).getPhone
    ())
1572.                      {
1573.                          min = j;
1574.                      }
1575.                  }
1576.                  //switch
1577.                  Membership temp = memberships.get(i);
1578.                  memberships.set(i, memberships.get(min));
1579.                  memberships.set(min, temp);
1580.              }
1581.          }
1582.
1583.          /**
1584.           * Sort by group size.
1585.           */
1586.          public void sortProByGroup()
1587.          {
1588.              //selection sort
1589.              for(int i = 0; i < programs.size()-1; i ++)
1590.              {
1591.                  int min = i;
1592.                  for(int j = i+1; j < programs.size(); j ++)
1593.                  {
1594.                      if(programs.get(j).getGroupSize() < programs.get(min).getGroupSi
    ze())
1595.                      {
1596.                          min = j;
1597.                      }
```

```
1598.                   }
1599.                   //switch
1600.                   Program temp = programs.get(i);
1601.                   programs.set(i, programs.get(min));
1602.                   programs.set(min, temp);
1603.               }
1604.           }
1605.
1606.           /**
1607.            * Sort program by contact name
1608.            */
1609.           public void sortProByName()
1610.           {
1611.               //selection sort
1612.               for(int i = 0; i < programs.size()-1; i ++)
1613.               {
1614.                   int min = i;
1615.                   for(int j = i+1; j < programs.size(); j ++)
1616.                   {
1617.                       if((programs.get(j).getWholeName()).compareToIgnoreCase(programs
    .get(min).getWholeName()) < 0)
1618.                       {
1619.                           min = j;
1620.                       }
1621.                   }
1622.                   //switch
1623.                   Program temp = programs.get(i);
1624.                   programs.set(i, programs.get(min));
1625.                   programs.set(min, temp);
1626.               }
1627.           }
1628.
1629.           /**
1630.            * export a daily summary file
1631.            */
1632.
1633.           /**
1634.            * Sort by contact number
1635.            */
1636.           public void sortProByPhone()
1637.           {
1638.               //selection sort
1639.               for(int i = 0; i < programs.size()-1; i ++)
1640.               {
1641.                   int min = i;
1642.                   for(int j = i+1; j < programs.size(); j ++)
1643.                   {
1644.                       if(programs.get(j).getPhone() < programs.get(min).getPhone())
1645.                       {
1646.                           min = j;
1647.                       }
1648.                   }
1649.                   //switch
1650.                   Program temp = programs.get(i);
1651.                   programs.set(i, programs.get(min));
1652.                   programs.set(min, temp);
1653.               }
1654.           }
1655.
1656.           /**
1657.            * Sort programs by time
```

```
1658.           */
1659.          public void sortProByTime()
1660.          {
1661.              //selection sort
1662.              for(int i = 0; i < programs.size()-1; i ++)
1663.              {
1664.                  int min = i;
1665.                  for(int j = i+1; j < programs.size(); j ++)
1666.                  {
1667.                      if(programs.get(j).getStartTime() < programs.get(min).getStartTi
    me())
1668.                      {
1669.                          min = j;
1670.                      }
1671.                  }
1672.                  //switch
1673.                  Program temp = programs.get(i);
1674.                  programs.set(i, programs.get(min));
1675.                  programs.set(min, temp);
1676.              }
1677.          }
1678.
1679.          /**
1680.           * Sort rent by wheelchairs and strollers
1681.           */
1682.          public void sortRenByItem()
1683.          {
1684.              ArrayList<Renting> ss = new ArrayList<>();
1685.              ArrayList<Renting> ws = new ArrayList<>();
1686.              for(int i = 0; i < rentings.size(); i++)
1687.              {
1688.                  Renting r = rentings.get(i);
1689.                  if(r.getType().equalsIgnoreCase(WHEELCHAIR))
1690.                  {
1691.                      ws.add(r);
1692.                  }
1693.                  else
1694.                  {
1695.                      ss.add(r);
1696.                  }
1697.              }
1698.              for(int i = 0; i<ss.size(); i ++)
1699.              {
1700.                  rentings.set(i, ss.get(i));
1701.              }
1702.              for (int k = ss.size(); k < ws.size(); k++)
1703.              {
1704.                  rentings.set(k, ws.get(k-ss.size()));
1705.              }
1706.          }
1707.
1708.          /**
1709.           * sort renting by rentor's name
1710.           */
1711.          public void sortRenByName()
1712.          {
1713.              //selection sort
1714.
1715.              for(int i = 0; i < rentings.size()-1; i ++)
1716.              {
1717.                  int min = i;
```

```
1718.                   for(int j = i+1; j < rentings.size(); j ++)
1719.                   {
1720.                       if((rentings.get(j).getWholeName()).compareToIgnoreCase(rentings
    .get(min).getWholeName()) < 0)
1721.                       {
1722.                           min = j;
1723.                       }
1724.                   }
1725.                   //switch
1726.                   Renting temp = rentings.get(i);
1727.                   rentings.set(i, rentings.get(min));
1728.                   rentings.set(min, temp);
1729.               }
1730.           }
1731.
1732.           /**
1733.            * sort renting by contact number
1734.            */
1735.           public void sortRenByPhone()
1736.           {
1737.               //selection sort
1738.               for(int i = 0; i < rentings.size()-1; i ++)
1739.               {
1740.                   int min = i;
1741.                   for(int j = i+1; j < rentings.size(); j ++)
1742.                   {
1743.                       if(rentings.get(j).getPhone() < rentings.get(min).getPhone())
1744.                       {
1745.                           min = j;
1746.                       }
1747.                   }
1748.                   //switch
1749.                   Renting temp = rentings.get(i);
1750.                   rentings.set(i, rentings.get(min));
1751.                   rentings.set(min, temp);
1752.               }
1753.           }
1754.
1755.           /**
1756.            * Write daily summary file
1757.            *
1758.            * @param fileName the name of the file
1759.            */
1760.           public void writedaiFile(String name) throws IOException //exception capture
    d by other classes
1761.           {
1762.               PrintWriter outputFile = new PrintWriter(new FileWriter(name));
1763.               outputFile.println("New Members\n");
1764.               for(int i = 0; i < newMems.size(); i++)
1765.               {
1766.                   outputFile.println("ID:" + Integer.valueOf(newMems.get(i).getId()));

1767.                   outputFile.println("First name: " + newMems.get(i).getFirst());
1768.                   outputFile.println("Last name: " + newMems.get(i).getLast());
1769.               }
1770.               outputFile.println("Deleted Members\n");
1771.               for(int i = 0; i < delMems.size(); i++)
1772.               {
1773.                   outputFile.println("ID:" + Integer.valueOf(delMems.get(i).getId()));

1774.                   outputFile.println("First name: " + delMems.get(i).getFirst());
```

```
1775.                 outputFile.println("Last name: " + delMems.get(i).getLast());
1776.             }
1777.             outputFile.println("Daily Programs\n");
1778.             for(int i = 0; i < programs.size(); i++)
1779.             {
1780.                 outputFile.println("Time:" + programs.get(i).displayTimePeriod());
1781.                 outputFile.println("Location: : " + programs.get(i).getLocation());

1782.                 outputFile.println("Contact: " + programs.get(i).getWholeName());
1783.             }
1784.             outputFile.println("Renting History\n");
1785.             for(int i = 0; i < rentings.size(); i++)
1786.             {
1787.                 outputFile.println("Item:" + rentings.get(i).getType());
1788.                 outputFile.println("Name: : " + rentings.get(i).getWholeName());
1789.             }
1790.             outputFile.println("Lost and Found History\n");
1791.             for(int i = 0; i < lofos.size(); i++)
1792.             {
1793.                 outputFile.println("Description: " + lofos.get(i).getDescrip());
1794.                 outputFile.println("Contact: " + lofos.get(i).getWholeName());
1795.                 outputFile.println("Location: " + lofos.get(i).getLoc());
1796.                 outputFile.println("Lost? " + lofos.get(i).strLost());
1797.                 outputFile.println("Found? " + lofos.get(i).strFound());
1798.                 outputFile.println("Phone number: " + Integer.valueOf(lofos.get(i).g
    etPhone()));
1799.             }
1800.             outputFile.close();
1801.         }
1802.         /*
1803.          * static methods
1804.          */
1805.
1806.         /**
1807.          *  write lost and found file
1808.          *
1809.          *  @param fileName the name of the file
1810.          */
1811.         public void writeLofoFile(String name) throws IOException //exception captur
    ed by other classes
1812.         {
1813.             PrintWriter outputFile = new PrintWriter(new FileWriter(name));
1814.             for(int i = 0; i < lofos.size(); i++)
1815.             {
1816.                 outputFile.println(lofos.get(i).getDescrip());
1817.                 outputFile.println(lofos.get(i).getFirstName());
1818.                 outputFile.println(lofos.get(i).getlastName());
1819.                 outputFile.println(lofos.get(i).getLoc());
1820.                 outputFile.println(lofos.get(i).strLost());
1821.                 outputFile.println(lofos.get(i).strFound());
1822.                 outputFile.println(Integer.valueOf(lofos.get(i).getPhone()));
1823.                 outputFile.println(Integer.valueOf(lofos.get(i).getDay()));
1824.                 outputFile.println(Integer.valueOf(lofos.get(i).getMonth()));
1825.                 outputFile.println(Integer.valueOf(lofos.get(i).getYear()));
1826.             }
1827.             outputFile.close();
1828.         }
1829.
1830.         /**
1831.          *  write membership file
1832.          *
```

```
1833.           *  @param name the name of the file
1834.           */
1835.          public void writeMemFile(String name) throws IOException //exception capture
   d by other classes
1836.          {
1837.              PrintWriter outputFile = new PrintWriter(new FileWriter(name));
1838.              for(int i = 0; i < memberships.size(); i++)
1839.              {
1840.                  outputFile.println(Integer.valueOf(memberships.get(i).getId()));
1841.                  outputFile.println(memberships.get(i).getAdress());
1842.                  outputFile.println(Integer.valueOf(memberships.get(i).getPhone()));

1843.                  outputFile.println(memberships.get(i).getLevel());
1844.                  outputFile.println(memberships.get(i).getFirst());
1845.                  outputFile.println(memberships.get(i).getLast());
1846.                  outputFile.println(Integer.valueOf(memberships.get(i).getExDay()));

1847.                  outputFile.println(Integer.valueOf(memberships.get(i).getExMonth()))
   ;
1848.                  outputFile.println(Integer.valueOf(memberships.get(i).getExYear()));

1849.              }
1850.              outputFile.close();
1851.          }
1852.      }
```

```
1.  import java.io.*;
2.  import java.util.*;
3.  /**
4.   * A membership of the science centre has a name, an ID, a phone number
5.   * a family adress (in Canada), an level of membership with an expiring
6.   * date, and list of family member under this ID up to 2 adults and 4
7.   * children.
8.   *
9.   * @author KaiYuan
10.  * @version 2018-03-06
11.  */
12. public class Membership
13. {
14.     //Constants declaration
15.
16.     //Each number corresponds to a certain level of membership
17.     private static final String LUNAR = "Lunar";
18.     private static final String PLANETARY = "Planetary";
19.     private static final String STELLAR = "Stellar";
20.     private static final String GALACTIC = "Galactic";
21.     private static final String COSMIC = "Cosmic";
22.     private static final int INITIAL_ID = 0;
23.     private static final int ID_INCREMENT = 1;
24.     private static final int EXPIRE_YEAR = 1;
25.     private static final int LUNAR_CHILD = 2;
26.     private static final int PLA_STE_CHILD = 4;
27.     private static final int GAL_COS_CHILD = 6;
28.
29.     //static fields declaration
30.     private static int lastId = 0;
31.     private static Calendar cal = Calendar.getInstance();
32.     //instance fields decalration
33.
34.     private String adress;
35.     private int exDay;
36.     private int exMonth;
37.     private int exYear;
38.     private int id;
39.     //membership level is identified by numbers;
40.     private String level;
41.     private int phone;
42.     private int children;
43.     private ArrayList<Adult> adults;
44.
45.
46.     /**
47.      * Constructs a member with a first name, last name, an ID,
48.      * a phone number, an adress (in Canada), and holds an membership
49.      * with an expiring date.
50.      *
51.      * @param phonenum member's phone number
52.      * @param memadress member's adress
53.      * @param idnum the id of a existed member
54.      * @param memlevelt member'level of membership
55.      * @param day the day that the membership will expire (expires in one year)
56.      * @param month the month that the membership will expire (expires in one year)
57.      * @param year the year that the membership will expire (expires in one year)
58.      * @param first the first name of a member under the membership
59.      * @param last the last name of a member under the membership
60.      */
61.     public Membership(String first, String last, int idnum, int phonenum,
```

```
62.     String memadress, String memlevel, int day, int month, int year)
63.     {
64.         // the validity of each field is checked in the main method.
65.         id = idnum;
66.         if(id > lastId)
67.         {
68.             lastId = id;
69.         }
70.         phone = phonenum;
71.         adress = memadress;
72.         level = memlevel;
73.         exDay = day;
74.         exMonth = month;
75.         exYear = year;
76.
77.         if(level == LUNAR)
78.         {
79.             children = LUNAR_CHILD;
80.         }
81.         else if(level == PLANETARY || level == STELLAR)
82.         {
83.             children = PLA_STE_CHILD;
84.         }
85.         else
86.         {
87.             children = GAL_COS_CHILD;
88.         }
89.
90.         adults = new ArrayList<>();
91.         this.addAdult(first, last);
92.     }
93.     /*
94.      * Accessors
95.      */
96.
97.     /**
98.      * Constructs a member with a first name, last name, an ID,
99.      * a phone number, an adress (in Canada), and holds an membership
100.             * with an expiring date.
101.             *
102.             * @param phonenum member's phone number
103.             * @param memadress member's adress
104.             * @param memlevelt member'level of membership
105.             * @param day the day that the membership will expire (expires in one year)
106.             * @param month the month that the membership will expire (expires in one ye
    ar)
107.             * @param year the year that the membership will expire (expires in one year
    )
108.             * @param first the first name of a member under the membership
109.             * @param last the last name of a member under the membership
110.             */
111.            public Membership(String first, String last,int phonenum,
112.            String memadress, String memlevel, int day, int month, int year)
113.            {
114.                // the validity of each field is checked in the main method.
115.                id = lastId + ID_INCREMENT;
116.                lastId = id;
117.                phone = phonenum;
118.                adress = memadress;
119.                level = memlevel;
120.                exDay = day;
```

```
121.                exMonth = month;
122.                exYear = year;
123.
124.                if(level == LUNAR)
125.                {
126.                    children = LUNAR_CHILD;
127.                }
128.                else if(level == PLANETARY || level == STELLAR)
129.                {
130.                    children = PLA_STE_CHILD;
131.                }
132.                else
133.                {
134.                    children = GAL_COS_CHILD;
135.                }
136.
137.                adults = new ArrayList<>();
138.                this.addAdult(first, last);
139.            }
140.
141.            /**
142.             * Add an adult member to the this membership id
143.             *
144.             * @param first member's first name
145.             * @param last member's last name
146.             */
147.            public void addAdult(String first, String last)
148.            {
149.                adults.add(new Adult(first, last));
150.            }
151.
152.            /**
153.             * Delete an adult from this membership
154.             *
155.             * @param adul the adult which need to be removed
156.             */
157.            public void deleteAdult(Adult adul)
158.            {
159.                adults.remove(adul);
160.            }//end of deleteCustomerByName(String name)
161.
162.            /**
163.             * Returns member's adress
164.             *
165.             * @return member's adress
166.             */
167.            public String getAdress()
168.            {
169.                return adress;
170.            }
171.
172.            /**
173.             * Returns all the Adults' name that is under this membership
174.             *
175.             * @return all the adults' name that is under this membership
176.             */
177.            public String getAdults()
178.            {
179.                String output = "";
180.                for(int i = 0; i < adults.size(); i++)
181.                {
```

```
182.                     output = output + adults.get(i).getFirstName() + " " + adults.get(i)
    .getLastName();
183.                     if(i != (adults.size() -1))
184.                     {
185.                         output = output + ", ";
186.                     }
187.                 }
188.                 return output;
189.             }
190.
191.             /**
192.              * return all the adults in an arrayList
193.              *
194.              * @return all the adults in an arrayList
195.              */
196.             public ArrayList<Adult> getArAdults()
197.             {
198.                 ArrayList<Adult> a = new ArrayList<>();
199.                 for(int i = 0; i < adults.size(); i ++)
200.                 {
201.                     a.add(adults.get(i));
202.                 }
203.                 return a;
204.             }
205.
206.             /**
207.              * Returns the number of children
208.              *
209.              * @return the number of children
210.              */
211.             public int getChildren()
212.             {
213.                 return children;
214.             }
215.
216.             /**
217.              * Returns member's expiring day
218.              *
219.              * @return member's expiring day
220.              */
221.             public int getExDay()
222.             {
223.                 return exDay;
224.             }
225.
226.             /**
227.              * Returns member's expiring month
228.              *
229.              * @return member's expiring month
230.              */
231.             public int getExMonth()
232.             {
233.                 return exMonth;
234.             }
235.
236.             /**
237.              * Returns the expiring date of the membership
238.              *
239.              * @return the expiring date of the membership
240.              */
241.             public String getExpiringDate()
```

```
242.            {
243.                String date = exYear + "/" + exMonth + "/" + exDay;
244.                return date;
245.            }
246.
247.            /**
248.             * Returns member's expiring year
249.             *
250.             * @return member's expiring year
251.             */
252.            public int getExYear()
253.            {
254.                return exYear;
255.            }
256.
257.            /**
258.             * return first holder's first name of the membership
259.             *
260.             * @return return first holder of the membership
261.             */
262.             public String getFirst()
263.             {
264.                 return adults.get(0).getFirstName();
265.             }
266.
267.            /**
268.             * Returns member's ID
269.             *
270.             * @return member's ID
271.             */
272.            public int getId()
273.            {
274.                return id;
275.            }
276.            /**
277.             * return first holder's last name of the membership
278.             *
279.             * @return return last holder of the membership
280.             */
281.             public String getLast()
282.             {
283.                 return adults.get(0).getLastName();
284.             }
285.
286.             /**
287.             * Returns the membership level of this membership
288.             *
289.             * @return member's membership level
290.             */
291.            public String getLevel()
292.            {
293.                return level;
294.            }
295.
296.            /**
297.             * Returns the family's home phone number
298.             *
299.             * @return member's home phone number
300.             */
301.            public int getPhone()
302.            {
```

```
303.                return phone;
304.            }
305.
306.            /*
307.             * Mutators
308.             */
309.
310.            /**
311.             * Check if the card is expired
312.             *
313.             * @return if the card is expired or not
314.             */
315.            public boolean isExpired()
316.            {
317.                boolean isEx;
318.                cal = Calendar.getInstance();
319.                int calYear = cal.get(Calendar.YEAR);
320.                int calMonth = cal.get(Calendar.MONTH) + 1;
321.                int calDay = cal.get(Calendar.DAY_OF_MONTH);
322.
323.                if(exYear > 0 && (exYear + EXPIRE_YEAR) < calYear)
324.                {
325.                    isEx = true;
326.                }
327.                else if(exYear > 0 && (exYear + EXPIRE_YEAR) == calYear && exMonth < cal
    Month)
328.                {
329.                    isEx = true;
330.                }
331.                else if(exYear > 0 && (exYear + EXPIRE_YEAR) == calYear && exMonth == ca
    lMonth && exDay < calDay)
332.                {
333.                    isEx = true;
334.                }
335.                else
336.                {
337.                    isEx = false;
338.                }
339.
340.                return isEx;
341.            }
342.
343.            /**
344.             * Renew the card for a year
345.             */
346.            public void renew()
347.            {
348.                exYear = exYear + EXPIRE_YEAR;
349.            }
350.
351.            /**
352.             * change the adress of the membership
353.             *
354.             * @param ad the new adress
355.             */
356.            public void setAdress(String ad)
357.            {
358.                adress = ad;
359.            }
360.
361.            /**
```

```
362.             * change the level of the membership
363.             *
364.             * @param lev the new level of membership
365.             */
366.            public void setLevel(String lev)
367.            {
368.                //it is assumed that all the information entered is correct
369.                level = lev;
370.                if(level == LUNAR)
371.                {
372.                    children = LUNAR_CHILD;
373.                }
374.                else if(level == PLANETARY || level == STELLAR)
375.                {
376.                    children = PLA_STE_CHILD;
377.                }
378.                else
379.                {
380.                    children = GAL_COS_CHILD;
381.                }
382.            }
383.
384.            /**
385.             * change the phone of the membership
386.             *
387.             * @param num new phone number
388.             */
389.            public void setPhone(int num)
390.            {
391.                phone = num;
392.            }
393.
394.            /**
395.             * Returns a string representation of the fields of a membership.
396.             *
397.             * @return a string representation of the fields of a mebership
398.             */
399.            public String toString()
400.            {
401.                return
402.                "\nID: " + id +
403.                "\nPhone Number: " + phone +
404.                "\nAdress: " + adress +
405.                "\nExpiring date: " + exYear + " " + exMonth + " " + exDay;
406.            }// end toString()
407.        }
```

```java
/**
 * A program has a time, location, a group size, a contact person, and a
 * contact number.
 *
 * @author KaiYuan
 * @version 2018-03-06
 */
public class Program
{
    //instance fields declaration
    private int startTime;
    private int endTime;
    private String location;
    private int group;
    private String firstName;
    private String lastName;
    private int phone;

    /**
     * Constract a program with a start time, end time, location, group
     * size, contact name and phone number
     *
     * @param start the program's start time
     * @param end the program's end time
     * @param loc the program's location
     * @param gro the size of the group participating the program
     * @param first the program's contact's first name
     * @param last the program's contact's last name
     * @param phonenum the program's contact's phone number
     */
    public Program(int start, int end, String loc, int gro, String first,
    String last, int phonenum)
    {
        startTime = start;
        //if end time is before 0800 or after 2000, then it is invalid
        endTime = end;
        //two program cannot be happening at the same location
        location = loc;
        //a group size cannot be negative
        if(gro < 0)
        {
            group = 0;
        }
        else
        {
            group = gro;
        }

        firstName = first;
        lastName = last;
        phone = phonenum;
    }

    /*
     * Accesors
     */

    /**
     * Display the time period as a String
     *
     * @return the time period of a program as a String
```

```
62.      */
63.     public String displayTimePeriod()
64.     {
65.         String start = String.format("%04d", startTime);
66.         String end = String.format("%04d", endTime);
67.
68.         String nstart = start.substring(0, 2) + ":" + start.substring(2, 4);
69.         String nend = end.substring(0, 2) + ":" + end.substring(2, 4);
70.         String period = nstart + "-" + nend;
71.         return period;
72.     }
73.
74.     /**
75.      * Return the program's end time
76.      *
77.      * @return program's end time
78.      */
79.     public int getEndTime()
80.     {
81.         return endTime;
82.     }
83.
84.     /**
85.      * return the program's contact first name
86.      *
87.      * @return program's contact first name
88.      */
89.     public String getFirstName()
90.     {
91.         return firstName;
92.     }
93.
94.     /**
95.      * Return the program's group size
96.      *
97.      * @return program's group size
98.      */
99.     public int getGroupSize()
100.        {
101.            return group;
102.        }
103.
104.        /**
105.         * return the program's contact last name
106.         *
107.         * @return program's contact last name
108.         */
109.        public String getLastName()
110.        {
111.            return lastName;
112.        }
113.
114.        /**
115.         * Return the program's location
116.         *
117.         * @return program's location
118.         */
119.        public String getLocation()
120.        {
121.            return location;
122.        }
```

```
123.
124.            /**
125.             * return the program's contact phone number
126.             *
127.             * @return program's contact phone number
128.             */
129.            public int getPhone()
130.            {
131.                return phone;
132.            }
133.
134.            /**
135.             * Return the program's start time
136.             *
137.             * @return program's start time
138.             */
139.            public int getStartTime()
140.            {
141.                return startTime;
142.            }
143.
144.            /**
145.             * return the whole name of the program's contact
146.             *
147.             * @return the whole name of the program's contact
148.             */
149.            public String getWholeName()
150.            {
151.                String name = firstName + " " + lastName;
152.                return name;
153.            }
154.
155.            /*
156.             * programs are provided by another department, it will be taken from
157.             * an existing formatted file. User is unable to mutate any
158.             * information here.
159.             */
160.        }
```

```java
/**
 * A renting object is either a wheelchair or a stroller, it is has the
 * name of the renter, and the contact number of the renter.
 *
 * @author KaiYuan
 * @version 2018-03-07
 */
public class Renting
{
    //instance fileds declaration

    private String firstName;
    private String lastName;
    private String type;
    private int phone;

    /**
     * Constructs renting object is either a wheelchair or a stroller, it is has the
     * name of the renter, and the contact number of the renter.
     *
     * @param last the last name of the contact
     * @param loc the location of where it is found or lost
     * @param phonenum the phone number of the contact
     * @param typeof stating whether it is a stroller or a wheelchair
     */
    public Renting(String first, String last, int phonenum, String typeof)
    {
        firstName = first;
        lastName = last;
        phone = phonenum;
        type = typeof;
    }

    /*
     * Accesors
     */

    /**
     * Returns the first name of the contact
     *
     * @return the first name of the contact
     */
    public String getFirstName()
    {
        return firstName;
    }

    /**
     * Returns the last name of the contact
     *
     * @return the last name of the contact
     */
    public String getlastName()
    {
        return lastName;
    }

    /**
     * Returns the phone of the object when it is found/lst
     *
     * @return the phone of the object when it is found/lst
```

```
62.      */
63.     public int getPhone()
64.     {
65.         return phone;
66.     }
67.
68.     /**
69.      * return the type of Renting
70.      *
71.      * @param ren
72.      * @return the type of the renting
73.      */
74.     public String getType()
75.     {
76.         return type;
77.     }
78.
79.     /**
80.      * return the whole name of the renter
81.      *
82.      * @return the whole name of the renter
83.      */
84.     public String getWholeName()
85.     {
86.         String name = firstName + " " + lastName;
87.         return name;
88.     }
89. }
```

```
1.  import java.io.*;
2.  import java.util.*;
3.  /**
4.   * A lostFound object is an lost or found item with a date, description
5.   * contact name, contact number, identification of whether it is a lost
6.   * item or a found item, and the location which it was lost of found.
7.   *
8.   * @author KaiYuna
9.   * @version 2018-03-06
10.  */
11. public class LostFound
12. {
13.
14.
15.     //class fields decalrations
16.     private static Calendar cal = Calendar.getInstance();
17.
18.     //contant declarations
19.     private static final int DECEMBER = 12;
20.     private static final int MAX_DAY_THIRTY = 30;
21.     private static final int MAX_DAY_THIRTY_ONE = 31;
22.     private static final int MAX_DAY_TWENTY_EIGHT = 28;
23.
24.     //instance fields declaration
25.     private int day;
26.     private int month;
27.     private int year;
28.     private String description;
29.     private String firstName;
30.     private String lastName;
31.     private int phone;
32.
33.     private String location;
34.     private boolean lost;
35.     private boolean found;
36.     private int days;
37.
38.     /**
39.      * Constructs a lostFound profile with a date, description
40.      * contact name, contact number, identification of whether it is a lost
41.      * item or a found item, and the location which it was lost of found.
42.      *
43.      * @param d the date that the item is reported.
44.      * @param m the month that the item is reported.
45.      * @param y the year that the item is reported.
46.      * @param descrip a general description or note of the item
47.      * @param first the first name of the contact
48.      * @param last the last name of the contact
49.      * @param loc the location of where it is found or lost
50.      * @param l state whether the object is lost
51.      * @param f state whether the object is found.
52.      * @param phonenum the phone number of the contact
53.      */
54.     public LostFound(int d, int m, int y, String descrip, String first,
55.     String last, String loc, boolean l, boolean f, int phonenum)
56.     {
57.         //if the object originally is lost and found, then lostFound profile will not b
    e created
58.         day = d;
59.         month = m;
60.         year = y;
```

```
61.         description = descrip;
62.         firstName = first;
63.         lastName = last;
64.         location = loc;
65.         lost = l;
66.         found = f;
67.         phone = phonenum;
68.         days = 1;
69.     }
70.
71.     /*
72.      * Accesors
73.      */
74.
75.     /**
76.      * Returns the date of theitem recorded
77.      *
78.      * @return the date of theitem recorded
79.      */
80.     public String getDate()
81.     {
82.         String date = year + "/" + month + "/" + day;
83.         return date;
84.     }
85.
86.     /**
87.      * Returns the day that the item is lost of found
88.      *
89.      * @return the day that the item is lost of found
90.      */
91.     public int getDay()
92.     {
93.         return day;
94.     }
95.
96.     /**
97.      * Returns the number of days that the item is left in lost and found
98.      *
99.      * @return number of days that the item is left in lost and found
100.           */
101.          public int getDays()
102.          {
103.              int days;
104.              cal = Calendar.getInstance();
105.              int calYear = cal.get(Calendar.YEAR);
106.              int calMonth = cal.get(Calendar.MONTH) + 1;
107.              int calDay = cal.get(Calendar.DAY_OF_MONTH);
108.
109.              if(month != calMonth)
110.              {
111.                  int maxDayThatMonth;
112.                  switch(month)
113.                  {
114.                      case 1:
115.                          maxDayThatMonth = MAX_DAY_THIRTY_ONE;
116.                          break;
117.                      case 2:
118.                          maxDayThatMonth = MAX_DAY_TWENTY_EIGHT;
119.                          break;
120.                      case 3:
121.                          maxDayThatMonth = MAX_DAY_THIRTY_ONE;
```

```
122.                             break;
123.                         case 4:
124.                             maxDayThatMonth = MAX_DAY_THIRTY;
125.                             break;
126.                         case 5:
127.                             maxDayThatMonth = MAX_DAY_THIRTY_ONE;
128.                             break;
129.                         case 6:
130.                             maxDayThatMonth = MAX_DAY_THIRTY;
131.                             break;
132.                         case 7:
133.                             maxDayThatMonth = MAX_DAY_THIRTY_ONE;
134.                             break;
135.                         case 8:
136.                             maxDayThatMonth = MAX_DAY_THIRTY_ONE;
137.                             break;
138.                         case 9:
139.                             maxDayThatMonth = MAX_DAY_THIRTY;
140.                             break;
141.                         case 10:
142.                             maxDayThatMonth = MAX_DAY_THIRTY_ONE;
143.                             break;
144.                         case 11:
145.                             maxDayThatMonth = MAX_DAY_THIRTY;
146.                             break;
147.                         case 12:
148.                             maxDayThatMonth = MAX_DAY_THIRTY_ONE;
149.                             break;
150.                         default:
151.                             maxDayThatMonth = MAX_DAY_THIRTY_ONE;
152.                     }
153.                     days = calDay + (maxDayThatMonth - day);
154.                 }
155.                 else
156.                 {
157.                     days = calDay - day;
158.                 }
159. 
160.                 return days;
161.             }
162. 
163.             /**
164.              * Returns the description of the lost of found object
165.              *
166.              * @return the description of the lost of found object
167.              */
168.             public String getDescrip()
169.             {
170.                 return description;
171.             }
172. 
173.             /**
174.              * Returns the first name of the contact
175.              *
176.              * @return the first name of the contact
177.              */
178.             public String getFirstName()
179.             {
180.                 return firstName;
181.             }
182. 
```

```
183.            /**
184.             * Returns the last name of the contact
185.             *
186.             * @return the last name of the contact
187.             */
188.            public String getlastName()
189.            {
190.                return lastName;
191.            }
192.
193.            /**
194.             * Returns the location of the object when it is found/lst
195.             *
196.             * @return the location of the object when it is found/lst
197.             */
198.            public String getLoc()
199.            {
200.                return location;
201.            }
202.
203.            /**
204.             * Returns the month that the item is lost of found
205.             *
206.             * @return the month that the item is lost of found
207.             */
208.            public int getMonth()
209.            {
210.                return month;
211.            }
212.
213.            /**
214.             * Returns the phone of the object when it is found/lst
215.             *
216.             * @return the phone of the object when it is found/lst
217.             */
218.            public int getPhone()
219.            {
220.                return phone;
221.            }
222.
223.            /**
224.             * return the whole name of the contact
225.             *
226.             * @return the whole name of the contact
227.             */
228.            public String getWholeName()
229.            {
230.                String name = firstName + " " + lastName;
231.                return name;
232.            }
233.
234.            /**
235.             * Returns the year that the item is lost of found
236.             *
237.             * @return the year that the item is lost of found
238.             */
239.            public int getYear()
240.            {
241.                return year;
242.            }
243.
```

```
244.            /**
245.             * Returns whether the item is lost or found as a boolean statement
246.             *
247.             * @returnwhether the item is lost or found
248.             */
249.            public boolean isFound()
250.            {
251.                return found;
252.            }
253.
254.
255.            /**
256.             * Returns whether the item is lost or found as a boolean statement
257.             *
258.             * @returnwhether the item is lost or found
259.             */
260.            public boolean isLost()
261.            {
262.                return lost;
263.            }
264.
265.            /**
266.             * Checks if the item is over 30 days by returning a boolean
267.             * expression once the method is called
268.             *
269.             * @param year the year that the item is recorded
270.             * @param month the month that the item is recorded
271.             * @param day the day that the item is recorded
272.             * @return if the item is over 30 days.
273.             */
274.            public boolean isOver(int year, int month, int day)
275.            {
276.                boolean isO;
277.                cal = Calendar.getInstance();
278.                int calYear = cal.get(Calendar.YEAR);
279.                int calMonth = cal.get(Calendar.MONTH) + 1;
280.                int calDay = cal.get(Calendar.DAY_OF_MONTH);
281.                if(year > 0 && year + 1 < calYear)
282.                {
283.                    isO = true;
284.                }
285.                else if(year > 0 && (year + 1) == calYear && month != DECEMBER)
286.                {
287.                    isO = true;
288.                }
289.                else if(year > 0 && (year + 1) == calYear && month == DECEMBER)
290.                {
291.                    if(calMonth == 1)
292.                    {
293.                        isO = false;
294.                    }
295.                    else
296.                    {
297.                        isO = true;
298.                    }
299.                }
300.                else if(year > 0 && year== calYear && (month + 1) == calMonth && day <=
    calDay)
301.                {
302.                    isO = true;
303.                }
```

```
304.                else
305.                {
306.                    isO = false;
307.                }
308.                return isO;
309.            }
310.
311.            /**
312.             * Checks if the item is both lost and found by returning
313.             * a boolean expression
314.             *
315.             * @return if the item is both lost and found
316.             */
317.            public boolean isPaired()
318.            {
319.                if(lost == true && found == true)
320.                {
321.                    return true;
322.                }
323.                else
324.                {
325.                    return false;
326.                }
327.            }
328.
329.            /*
330.             * Mutators
331.             */
332.            /*
333.             * Checks if the object is both lost and found
334.             * check if the day is over thirty days.
335.             *
336.             */
337.
338.            /**
339.             * Return yes of no to represent whether the item is found
340.             *
341.             * @return yes of no to represent whether the item is found
342.             */
343.            public String strFound()
344.            {
345.                String str;
346.                if(found)
347.                {
348.                    str = "True";
349.                }
350.                else
351.                {
352.                    str = "False";
353.                }
354.                return str;
355.            }
356.
357.            /**
358.             * Return yes of no to represent whether the item is lost
359.             *
360.             * @return yes of no to represent whether the item is lost
361.             */
362.            public String strLost()
363.            {
364.                String str;
```

```
365.                if(lost)
366.                {
367.                    str = "True";
368.                }
369.                else
370.                {
371.                    str = "False";
372.                }
373.                return str;
374.            }
375.        }
```

```java
/**
 * An adult is a member of the science Centre
 *
 * @author KaiYuan Chi
 * @version 2018-03-06
 */
public class Adult extends Member
{
    /**
     * Constructor for objects of class Adult
     *
     * @param frist the adult's first name
     * @param last the adult's last name
     */
    public Adult(String first, String last)
    {
        super(first, last);
    }
}
```

```java
/**
 * A member object has a name, and it is under a membership.
 *
 * @author KaiYun
 * @version 2018-03-06
 */
public abstract class Member
{
    //instance fields declaration
    private String firstName;
    private String lastName;


    /**
     * Constructs A member with a first name and a last name
     */
    public Member(String first, String last)
    {
        firstName = first;
        lastName = last;
    }

    /*
     * Accessors
     */

    /**
     * Return the first name of a member
     *
     * @return the first name of a member
     */
    public String getFirstName()
    {
        return firstName;
    }

    /**
     * Return the last name of a member
     *
     * @return the last name of a member
     */
    public String getLastName()
    {
        return lastName;
    }
}
```

Read/Write Text File Samples

```
1
2342 cool road
9328420
Planetary
Lily
Yang
4
5
2018
2
999 Sesame street
2834293
stellar
Tam
Hang
3
5
2019
4
7 Sandy Drive
83427923
lunar
Sam
Louis
23
9
2018
7
Takeout road
839283
Planetary
Marcus
Canp
10
3
2019
9
100 Woburn Road
839283
Planetary
Kassi
Rong
10
3
2019
11
3849 Handover drive
999999
Planetary
Landy
Mop
10
3
2019
12
7 Nitrogen street
8392839
Planetary
```

```
59. Tony
60. Yang
61. 11
62. 3
63. 2019
64. 14
65. 11 Sodium Rd
66. 738393
67. Stellar
68. Cicy
69. Tango
70. 11
71. 3
72. 2019
```

```
1200
1430
science hotspot
23
Nate
Stones
3427183
1445
1600
auditorium
230
kathy
matt
8239482
0830
1740
IMAX theatre
180
Madarin
Lumen
839382
0900
1000
Mini Classroom
30
Mr.
Gara
839382
1500
1900
RainForest Exibition
70
Castera
Glandi
839283
1445
1545
Science Hotspot
30
Mimi
Calendi
839204
1800
2030
IMAX Theatre
195
Colan
Vestumdum
839203
```

```
A yellow Pillow Case
Tiny
Bin
Cat's eye Exibit
False
True
839283
10
3
2018
A blue glove
Cindy
Wei
Animal Kindom
True
False
839283
10
3
2018
Blue ink pen
Catherine
Lai
Tim Hortons
True
False
839283
10
3
2018
A can of shooting starts
Melody
Lyperd
Music hall
True
False
8392034
10
3
2018
```

```
New Members

Deleted Members

Daily Programs

Time:12:00-02:30
Location: : science hotspot
Contact: Nate Stones
Time:02:45-04:00
Location: : auditorium
Contact: kathy matt
Renting History

Lost and Found History

Description: A yellow Pillow Case
Contact: Tiny Bin
Location: Cat's eye Exibit
Lost? False
Found? True
Phone number: 839283
Description: A blue glove
Contact: Cindy Wei
Location: Animal Kindom
Lost? True
Found? False
Phone number: 839283
Description: Blue ink pen
Contact: Catherine Lai
Location: Tim Hortons
Lost? True
Found? False
Phone number: 839283
Description: A can of shooting starts
Contact: Melody Lyperd
Location: Music hall
Lost? True
Found? False
Phone number: 8392034
```