

LAB 211 Assignment

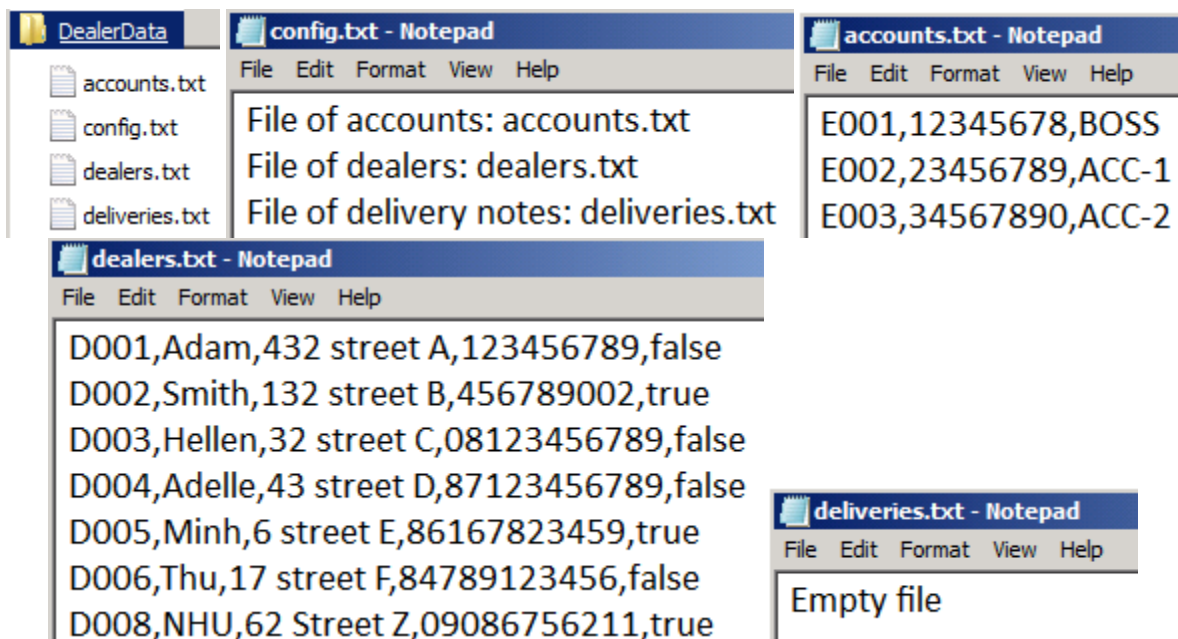
Type: Long Assignment
Code: J1.L.P0016
LOC: 500
Slot(s): N/A

Title

Dealers Management Program

Background

- AZW, a firm, needs a Java console program for managing its product dealers. This program must support a basic security. Roles in the firm include:
 - (1) Boss: Managing users
 - (2) ACC-1: Managing dealers
 - (3) ACC-2: Managing deliveries notes.
- Data files are supported as following:



File related to the program can be setup flexibly through the file *config.txt*.

All users must be login to system to carry out appropriate management activities based on his/her role. At the moment, a program for managing dealers is required to develop.

Program model is proposed:

User → Login → Managing dealers (role: ACC-1)
→ Managing deliveries (role: ACC-2) – developed afterward.

Program Specifications

Build a management program. With the following basic functions

0. Show medical examination result

1. Add new patients

2. Record medical examination

3. Real-time update processing

Others- Quit

Each menu choice should invoke an appropriate function to perform the selected menu item. Your program must display the menu after each task and wait for the user to select another option until the user chooses to quit the program.

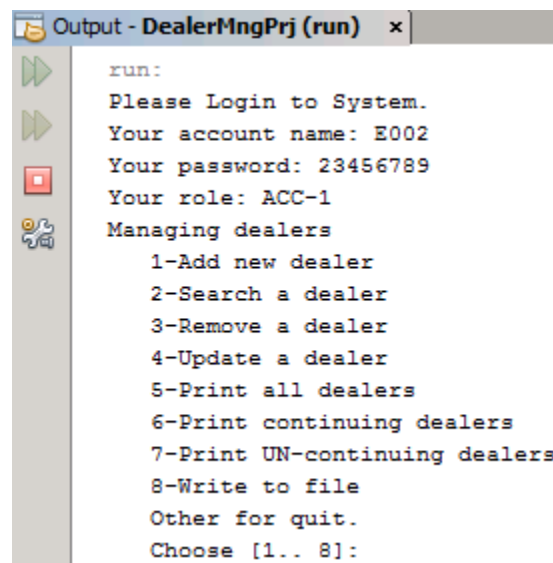
Each department has the following information: departmentID, name, createDate, lastUpdateDate

Each doctor has the properties such that doctorID, name, sex, address, departmentID, createDate, lastUpdateDate

Each patient has the following information: patientID, name, age, address. Patient information is stored in patient.dat file

The **examination.dat** file has stored examination information include: examinationID, doctorID, patientID, result, date.

Features:



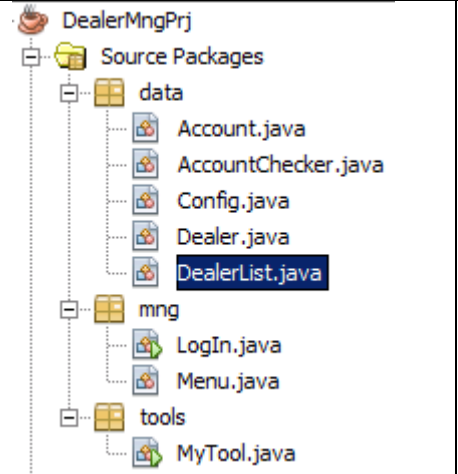
```
run:
Please Login to System.
Your account name: E002
Your password: 23456789
Your role: ACC-1
Managing dealers
    1-Add new dealer
    2-Search a dealer
    3-Remove a dealer
    4-Update a dealer
    5-Print all dealers
    6-Print continuing dealers
    7-Print UN-continuing dealers
    8-Write to file
Other for quit.
Choose [1.. 8]:
```

A Security in Code view

- ✓ An object belonging to the LogIn class should be a parameter to create a DealerList object. So, the class DealerList can only be used depending on the LogIn object.

Component/ class	Sum	LOCs	Question Mng
Structure	20	20	
Account	10	10	
MyTool	130	100	30
Config	20	20	
AccountChecker	30	30	
Menu	10	10	
DealerList	200	150	50
LogIn	80	50	30
Total	500		

Design Hint

	<ul style="list-style-type: none">Class for an accountClass supporting checking validity of an accountClass for reading config file to get related filenamesClass for a dealerClass for a list of dealers <ul style="list-style-type: none">Class for Log in interface. The main() method is put in this classClass for menu <ul style="list-style-type: none">Class for programming common used basic methods
---	---

The Hint in Implementation – Step 1

Because there is a relation between classes in different packages (the mng.LogIn class must be used in the data.DealerList class). At this step, we should write basic following code to prevent the Java compiler causing compiled errors. You should implement them in order.

1- The Account Class

```
1  /* Class for an account */
2  package data;
3
4  public class Account {
5      private String accName; // ID
6      private String pwd; // password
7      private String role;
8      //contrustor - IMPEMENT IT
9      public Account(String accName, String pwd, String role) {...5 lines }
14     // Getters-- IMPLEMENT THEM
15     public String getAccName() {...3 lines }
18
19     public String getPwd() {...3 lines }
22
23     public String getRole() {...3 lines }
26
27 } // class Account
```

```

1  ▢ /* Class for Log In interface */
2      package mng;
3  ▢  import data.Account;
4      import data.AccountChecker;
5      import data.DealerList;
6      import tools.MyTool;
7
8      public class LogIn {
9          private Account acc=null;
10             // constructor
11  ▢      public LogIn( Account acc){
12             this.acc= acc;
13             }
14  ▢  }

```

```

package tools;

public class MyTool {
}

```

```

1  ▢ /* Class for a \product dealer */
2      package data;
3  ▢  import tools.MyTool;
4
5      public class Dealer implements Comparable<Dealer> {
6          public static final char SEPARATOR = ',';
7          public static final String ID_FORMAT = "D\\d{3}";
8          public static final String PHONE_FORMAT = "\\d{9}|\\d{11}";
9          private String ID; // template D000
10         private String name; // dealers's name
11         private String addr; // dealer's address
12         private String phone; // 9 or 11 digits
13         private boolean continuing; // whether this dealer still cooperates or not
14
15         // constructor using 6 parameters - IMPLEMENT IT
16         public Dealer(String ID, String name, String addr, String phone,
17  ▢             boolean continuing) {...7 lines }
24     }

```

```

1  /* Class for a list of dealers */
2  package data;
3  import java.util.List;
4  import java.util.ArrayList;
5  import java.sql.Date;
6  import tools.MyTool;
7  import mng.LogIn;
8
9  public class DealerList extends ArrayList<Dealer>{
10     LogIn loginObj= null;
11     private static final String PHONEPATTERN = "\\d{9}|\\d{11}";
12     private String dataFile = "";
13     boolean changed = false; // whether data in the list changed or not
14
15     // Contructor using loginObj as a parameter - IMPLEMENT IT
16     public DealerList(LogIn loginObj) {...4 lines }
17
18
19
20
21 }

```

The Hint in Implementation – Step 2, Implementations in Details

2- The MyTool class

```

1  /* Class for validating input and inputting data using a condition
2  Date format: y: year, M: month in year, d: day in month
3  Separators: - / : but they are not mixed
4  Example: yyyy/MM/dd dd:MM:yyyy MM/dd/yyyy:
5  Regular expression for pattern
6  Phone no 9 or 11 digits: "\\d{9}|\\d{11}"
7  Phone no 9 to 11 digits: "\\d{9,11}"
8  ID format X0000 : "X\\d{4}"
9  ID format X0000 or M000: "X\\d{4}|M\\d{3}"
10 */

```

```

11 package tools;
12 import java.sql.Date; // containing year, month,m day only
13 import java.text.SimpleDateFormat; // for converting string <--> Date
14 import java.util.Scanner; // for input data
15 import java.io.File; // For checking a file
16 import java.io.FileReader; // classes for reading data from a text file
17 import java.io.BufferedReader;
18 import java.io.FileWriter; // classes for writing data to a text file
19 import java.io.PrintWriter;
20 import java.util.ArrayList; // Class for a list
21 import java.util.List; // Interface for a list
22 import java.text.ParseException; // exception when parsing data from a string
23 import java.io.IOException; // Exception when accessing file
24
25 public class MyTool {
26     public static final Scanner SC = new Scanner(System.in);
27
28     // Checking whether str matches a pattern or not
29     // Use the method String.matches (regex) - IMPLEMENT IT
30     public static boolean validStr (String str, String regex){...3 lines }
31
32     /* Checking a password with minLen in which it contains at least a character,
33        a number and 1 specific character
34        .* : there may be one or more any character
35        \\d : digit \\W : [^a-zA-Z0-9] : it is not a character and not a digit
36        */
37     public static boolean validPassword (String str, int minLen){
38         if(str.length()<minLen) return false;
39         return str.matches(".*[a-zA-Z]+.*") && // AT LEAST 1 CHARACTER
40             str.matches(".*[\\d]+.*") && // AT LEAST 1 DIGIT
41             str.matches(".*[\\W]+.*");// AT LEAST 1 SPECIAL CHAR
42     }
43
44     // Date format: yyyy/MM/dd, MM/dd/yyyy, dd/MM/yyyy, ...
45     // yyyy/dd/MM 2000/30/02 -> 2000/01/03 automatically
46     // Date string will be changed to a valid date value automatically
47     public static Date parseDate(String dateStr, String dateFormat){
48         SimpleDateFormat dF = (SimpleDateFormat)SimpleDateFormat.getInstance();
49         dF.applyPattern(dateFormat);
50         try{
51             long t = dF.parse(dateStr).getTime();
52             return new Date(t);
53         }
54         catch(ParseException e){
55             System.out.println(e);
56         }
57         return null;
58     }
59 }
60
61

```

```

62 // Use the class SimpleDateFormat
63 // Use the method applyPattern(str) to apply a specific format
64 // Use the method format(date) to convert date -> String
65 // IMPLEMENT IT
66 public static String dataToStr(Date date, String dateFormat) {...5 lines }
71
72 // Convert bool string to boolean
73 public static boolean parseBool(String boolStr){
74     char c = boolStr.trim().toUpperCase().charAt(0);
75     return (c=='1' || c=='Y' || c=='T');
76 }
77 // Tools for inputting data
78 public static String readNonBlank(String message){
79     String input = "";
80     do{
81         System.out.print(message + ": ");
82         input = SC.nextLine().trim();
83     }
84     while (input.isEmpty());
85     return input;
86 }
87 public static String readPattern(String message, String pattern)
88     String input = "";
89     boolean valid;
90     do{
91         System.out.print(message + ": ");
92         input = SC.nextLine().trim();
93         valid = validStr(input,pattern);
94     }
95     while (!valid);
96     return input;
97 }
98 public static boolean readBool(String message){
99     String input;
100     System.out.print(message + "[1/0-Y/N-T/F]: ");
101     input = SC.nextLine().trim();
102     if (input.isEmpty()) return false;
103     char c = Character.toUpperCase(input.charAt(0));
104     return (c=='1' || c=='Y' || c=='T');
105 }

```

```

106      /* Method for reading lines from text file
107      Create an array list, named as list
108      Open file
109      While ( still read succesfully a line in the file){
110          trim the line;
111          if line is not empty, add line to the list
112      }
113      Close file
114      return list;
115      IMPLEMENT IT
116      */
117      public static List<String> readLinesFromFile (String filename) {...23 lines }
140
141      /* Method for writing a list to a text file line-by-line
142      Open the file for writing
143      For each object in the list, write th eobject to file
144      Close the file
145      IMPLEMENT IT
146      */
147      public static void writeFile(String filename, List list) {...11 lines }
158
159      // Test- It is optional
160      public static void main (String[] args){
161          // Phone: 9 or 11 digits - OK
162          System.out.println("Tests with phone numbers:");
163          System.out.println(validStr("012345678", "\\d{9}|\\d{11}"));
164          System.out.println(validStr("01234567891", "\\d{9}|\\d{11}"));
165          System.out.println(validStr("12345678", "\\d{9}|\\d{11}"));
166
167          // Test password - OK
168          System.out.println(validPassword("qwerty", 8)); // false
169          System.out.println(validPassword("qwertyABC", 8)); // false
170          System.out.println(validPassword("12345678", 8)); // false
171          System.out.println(validPassword("qbc123456", 8)); // false
172          System.out.println(validPassword("qbc@123456", 8)); // true
173          // ID format D000 -> OK
174          System.out.println("Tests with IDs:");
175          System.out.println(validStr("A0001", "D\\d{3}"));
176          System.out.println(validStr("10001", "D\\d{3}"));
177          System.out.println(validStr("D0001", "D\\d{3}"));
178          System.out.println(validStr("D101", "D\\d{3}"));
179

```



```

180 //Test date format -> OK
181 Date d = parseDate("2022:12:07", "yyyy:MM:dd");
182 System.out.println(d);
183 System.out.println(dataToStr(d, "dd/MM/yyyy")); // test OK
184 d = parseDate("12/07/2022", "MM/dd/yyyy");
185 System.out.println(d);
186 d = parseDate("2022/07/12", "yyyy/dd/MM");
187 System.out.println(d);
188 d = parseDate("2000/29/02", "yyyy/dd/MM");
189 System.out.println(d);
190 d = parseDate("2000/30/02", "yyyy/dd/MM");
191 System.out.println(d);
192 d = parseDate("2000/40/16", "yyyy/dd/MM");
193 System.out.println(d);
194 // Test input data -> ok
195 String input = readNonBlank("Input a non-blank string");
196 System.out.println(input); // OK
197 input = readPattern("Phone 9/11 digits", "\\d{9}|\\d{11}");
198 System.out.println(input); // OK
199 input = readPattern("ID- format X00000", "X\\d{5}");
200 System.out.println(input); // OK
201 boolean b = readBool("Input boolean");
202 System.out.println(b); // OK
203
204 }
205 } //class MyTool

```

3- The Config Class

```

1  /* Class for reading config.txt file*/
2  package data;
3  import java.util.List;
4  import tools.MyTool;
5  public class Config {
6      private static final String CONFIG_FILE = "DealerData/config.txt";
7      private String accountFile;
8      private String dealerFile;
9      private String deliveryFile;
10
11      public Config() {
12          readData();
13      }

```

config.txt - Notepad

File Edit Format View Help

File of accounts: accounts.txt
File of dealers: dealers.txt
File of delivery notes: deliveries.txt

```

14 private void readData() {
15     List<String> lines = MyTool.readLinesFromFile(CONFIG_FILE);
16     for (String line: lines) {
17         line = line.toUpperCase();
18         String[] parts = line.split(":");
19         if (line.indexOf("ACCOUN")>=0)
20             accountFile = "DealerData/" + parts[1].trim();
21         else if (line.indexOf("DEALE")>=0)
22             dealerFile = "DealerData/" + parts[1].trim();
23         else if (line.indexOf("DELIVER")>=0)
24             deliveryFile = "DealerData/" + parts[1].trim();
25     }
26 }

27 // Getters- Implement IT
28 public String getAccountFile() {...3 lines }
31
32 public String getDealerFile() {...3 lines }
35
36 public String getDeliveryFile() {...3 lines }
39
40 } //class Config

```

File of accounts: accounts.txt
File of dealers: dealers.txt
File of delivery notes: deliveries.txt

4- The AccountChecker Class

```

1  /* Class for checking validity of an account */
2  package data;
3
4  import tools.MyTool;
5  import java.util.List;
6
7  public class AccountChecker {
8      private String accFile;
9      private static String SEPARATOR=",";
10     public AccountChecker() {
11         setupAccFile();
12     }
13     private void setupAccFile(){
14         Config cR = new Config();
15         accFile = cR.getAccountFile();
16     }

```

accounts.txt - Notepad

File Edit Format View Help

E001,12345678,BOSS
E002,23456789,ACC-1
E003,34567890,ACC-2

```

17 // Check validity of an account
18 public boolean check(Account acc){
19     // Read data in file
20     List<String> lines = MyTool.readLinesFromFile(accFile);
21     // Traverse each line for checking
22     for (String line: lines){
23         String[] parts= line.split(this.SEPARATOR);
24         if (parts.length<3) return false;
25         if( parts[0].equalsIgnoreCase(acc.getAccName()) &&
26             parts[1].equals(acc.getPwd()) &&
27             parts[2].equalsIgnoreCase(acc.getRole()))
28             return true;
29     }
30     return false;
31 }
32 // Test OK - It is optional
33 public static void main(String[] args){
34     AccountChecker aChk = new AccountChecker();
35     Account acc = new Account("E001", "12345678", "BOSS");
36     boolean valid = aChk.check(acc);
37     System.out.println("Needs OK, OK?: " + valid);
38     acc = new Account("E002", "23456789", "ACC-1");
39     valid = aChk.check(acc);
40     System.out.println("Needs OK: OK? " + valid);
41     acc = new Account("E003", "123456789", "ACC-2");
42     valid = aChk.check(acc);
43     System.out.println("Needs NO OK, OK?: " + valid);
44 }
45 }
46
47 } // class AccountChecker

```

5- The Dealer Class

```

1  ☐ /* Class for a product dealer */
2  package data;
3  ☐ import tools.MyTool; D001,Adam,432 street A,123456789,false
4
5  public class Dealer implements Comparable<Dealer> {
6      public static final char SEPARATOR = ',';
7      public static final String ID_FORMAT = "D\\d{3}";
8      public static final String PHONE_FORMAT = "\\d{9}|\\d{11}";
9      private String ID; // template D000
10     private String name; // dealers's name
11     private String addr; // dealer's address
12     private String phone; // 9 or 11 digits
13     private boolean continuing; // whether this dealer still cooperates or not
14
15     // constructor using 5 parameters - IMPLEMENT IT
16     public Dealer(String ID, String name, String addr, String phone,
17 ☐ boolean continuing) {...7 lines }
24
25     // constructor using a line using the separator ','
26 ☐ public Dealer(String line) { D001,Adam,432 street A,123456789,false
27         String[] parts = line.split("" + this.SEPARATOR);
28         ID = parts[0].trim(); // dealer ID
29         name = parts[1].trim(); // dealers's name
30         addr = parts[2].trim(); // dealer's address
31         phone = parts[3].trim(); // 9 or 11 digits
32         continuing = MyTool.parseBool(parts[4]);
33     }
34     // getters, setters- IMPLEMENT THEM
35 ☐ public String getID() {...3 lines }
38 ☐ public void setID(String ID) {...3 lines }
41 ☐ public String getName() {...3 lines }
44 ☐ public void setName(String name) {...4 lines }
48 ☐ public String getAddr() {...3 lines }
51 ☐ public void setAddr(String addr) {...3 lines }
54 ☐ public String getPhone() {...3 lines }
57 ☐ public void setPhone(String phone) {...3 lines }
60 ☐ public boolean isContinuing() {...3 lines }
63 ☐ public void setContinuing(boolean continuing) {...3 lines }
66
67     @Override
68 ☒ ☐ public String toString() { D001,Adam,432 street A,123456789,false
69         return ID + SEPARATOR + name + SEPARATOR +
70             addr + SEPARATOR + phone + SEPARATOR +
71             continuing + "\n";
72     }

```

```

73         //Comparing tool: comparing based on their ID- IMPLEMENT IT
74         @Override
75         public int compareTo(Dealer o) {...3 lines }
78     } //class Dealer

```

6- The Menu Class

```

1  /* Class for a menu */
2  package mng;
3  import java.util.ArrayList;
4  import tools.MyTool;
5
6  public class Menu extends ArrayList<String>{
7      public Menu() {
8          super();
9      }
10     public Menu(String[] items) {
11         super();
12         for (String item: items) this.add(item);
13     }
14     // Get user choice -- IMPLEMENT IT
15     public int getChoice(String title) {...10 lines }
25 } // class Menu

```

7- The DealerList Class

```

1  /* Class for a list of dealers */
2  package data;
3  import java.util.List;
4  import java.util.ArrayList;
5  import tools.MyTool;
6  import mng.LogIn;
7
8  public class DealerList extends ArrayList<Dealer>{
9      LogIn loginObj= null;
10     private static final String PHONEPATTERN = "\\d{9}|\\d{11}";
11     private String dataFile = "";
12     boolean changed = false; // whether data in the list changed or not
13
14     // Contructor using loginObj as a parameter - IMPLEMENT IT
15     public DealerList(LogIn loginObj) {...4 lines }
19
20

```

```

21      /* Load dealers form file
22      Use MyTool to read lines from the data file, List lines
23      For each line in lines, create a dealer using this line as parameter
24      Add this created dealer to the list
25      IMPLEMENT IT
26      */
27      + private void loadDealerFromFile() {...7 lines }
34
35      // initializing basic data in files
36      - public void initWithFile(){
37          Config cR = new Config();
38          dataFile = cR.getDealerFile(); // get file containing dealers
39          loadDealerFromFile(); // load dealers from file
40      }
41
42      /* Get the list of continuing dealers
43      Create new result list belonging to DealerList
44      For each Dealer d in this
45          if d.isContinuing() == true then add d to result list;
46      Return result;
47      IMPLEMENT IT
48      */
49      + public DealerList getContinuingList() {...6 lines }
55
56      /* Get the list of un-continuing dealers
57          This method is similar to getContinuingList()
58          but using d.isContinuing() == false
59          IMPLEMENT IT
60      */
61      + public DealerList getUnContinuingList() {...6 lines }
67
68      /* Search dealer - Use linear search-- IMPLEMENT IT
69      Convert the parameter ID to uppercase
70      N= size of this list
71      for (i=0; i<N; i++)
72          if (i(th)dealer having the same ID ) return i;
73      return -1
74      */
75      + private int searchDealer(String ID) {...6 lines }
81

```

```

82      /* Search dealer - IMPLEMENT IT
83      Input String ID
84      Call searchDealer(ID) and assign it's return value to pos
85      if (pos<0) output "NOT FOUND!"
86      else output the pos(th) dealer in this list
87      */
88      public void searchDealer() { ...7 lines }
95
96      // Add new dealer
97      public void addDealer() {
98          String ID;
99          String name; // dealers's name
100         String addr; // dealer's address
101         String phone; // 9 or 11 digits
102         boolean continuing;
103         int pos;
104         do{ // input data
105             ID = MyTool.readPattern("ID of new dealer", Dealer.ID_FORMAT);
106             ID= ID.toUpperCase();
107             pos= searchDealer(ID);
108             if (pos>=0) System.out.println("ID is duplicated!");
109         }
110         while (pos>=0);
111         name = MyTool.readNonBlank("Name of new dealer: ").toUpperCase();
112         addr = MyTool.readNonBlank("Address of new dealer: ");
113         phone = MyTool.readPattern("Phone number: ", Dealer.PHONE_FORMAT);
114         continuing = true; // default value for new dealer
115         Dealer d = new Dealer(ID, name, addr, phone, continuing);
116         this.add(d);
117         System.out.println("New dealer has been added.");
118         changed= true;
119     }
120
121     /* Remove a dealer: Assign continuing = false -- IMPLEMENT IT
122     Input ID
123     pos = search(ID)
124     if (pos<0) output "Not ffound!"
125     else{
126         set field continuing of the pos(th) element to FALSE
127         output "Removed"
128         changed = true ; // data changed
129     }
130     */
131     public void removeDealer() { ...11 lines }
141

```

```

142 // update a dealer
143 // Only name, addr and phne can be changed
144 // Only changing name is expressedm you do all remainders
145 public void updateDealer() {
146     System.out.print("Dealer's ID needs updating: ");
147     String ID = MyTool.SC.nextLine();
148     int pos = searchDealer(ID);
149     if (pos < 0) System.out.println("Dealer " + ID + " not found!");
150     else {
151         Dealer d = this.get(pos);
152         String newName = ""; // Update name
153         System.out.print("new name, ENTER for omitting: ");
154         newName = MyTool.SC.nextLine().trim().toUpperCase();
155         if (!newName.isEmpty()) {
156             d.setName(newName);
157             changed = true;
158         }
159         // update addr - IMPLEMENT IT
160         // update phone - IMPLEMENT IT
161     }
162 }
163
164 // Print all dealers - IMPLEMENT IT
165 public void printAllDealers() {
166     if (this.isEmpty()) System.out.println("Empty List!");
167     else System.out.println(this);
168 }
169
170 // Print all continuing dealers
171 public void printContinuingDealers() {
172     this.getContinuingList().printAllDealers();
173 }
174
175 // Print all un-continuing dealers - IMPLEMENT IT
176 public void printUnContinuingDealers() { ...3 lines }
177
178

```



```

199 // Write dealer list to file
200 public void writeDealerToFile() {
201     if (changed) {
202         MyTool.writeFile(dataFile, this);
203         changed= false;
204     }
205 }
206
207 // getters, setters - IMPLEMENT THEM
208 public boolean isChanged() {...3 lines }
211 public void setChanged(boolean changed) {...3 lines }
214 } // class DealerList

```

8- The LogIn Class

```

1  /* Class for Log In interface */
2  package mng;
3  import data.Account;
4  import data.AccountChecker;
5  import data.DealerList;
6  import tools.MyTool;
7
8  public class LogIn {
9      private Account acc=null; // account will log in
10         // constructor
11     public LogIn( Account acc){
12         this.acc= acc;
13     }
14
15     /* Input data of an account - IMPLEMENT IT
16        Create new Account
17        return this account
18     */
19     public static Account inputAccount() {...13 lines }
20
21     // getter
22
23     public Account getAcc() {
24         return acc;
25     }
26
27

```

```

38 // Main program
39 public static void main(String[] args){
40     Account acc = null; // account will login to system
41     boolean cont = false; // login again?
42     boolean valid= false; // valid account or not
43     do {
44         AccountChecker accChk = new AccountChecker();
45         acc= inputAccount(); // input account's data
46         valid = accChk.check(acc); // check validity
47         if (!valid)
48             cont = MyTool.readBool("Invalid account- Try again?");
49         if (!valid&& !cont) System.exit(0); // quit the program
50     }
51     while (cont);
52     LogIn loginObj = new LogIn(acc); // create a login obj for valid acc
53     // Run Dealer manager
54     if (acc.getRole().equalsIgnoreCase("ACC-1")){
55         // Setup menu
56         String[] options = { "Add new dealer", "Search a dealer",
57                               "Remove a dealer", "Update a dealer",
58                               "Print all dealers", "Print continuing dealers",
59                               "Print UN-continuing dealers", "Write to file"
60         };
61         Menu mnu = new Menu(options);
62         DealerList dList = new DealerList(loginObj); // Setup DealerList
63         dList.initWithFile();
64         int choice=0;
65         do{ // Do activities
66             choice = mnu.getChoice("Managing dealers");
67             switch(choice){
68                 case 1: dList.addDealer(); break;
69                 case 2: dList.searchDealer(); break;
70                 case 3: dList.removeDealer(); break;
71                 case 4: dList.updateDealer(); break;
72                 case 5: dList.printAllDealers(); break;
73                 case 6: dList.printContinuingDealers(); break;
74                 case 7: dList.printUnContinuingDealers(); break;
75                 case 8: dList.writeDealerToFile(); break;
76                 default:
77                     if(dList.isChanged()){
78                         boolean res= MyTool.readBool("Data changed. Write to file?");
79                         if (res==true) dList.writeDealerToFile();
80                     }
81             }
82         }
83         while (choice > 0 && choice < mnu.size());
84         System.out.println("Bye.");
85     }
86 } // main()
87 } // class LogIn

```

- ✚ The above specifications are only basic information; you must perform a requirements analysis step and build the application according to real requirements.
- ✚ The lecturer will explain the requirement only once on the first slot of the assignment.