LAB 211 Assignment

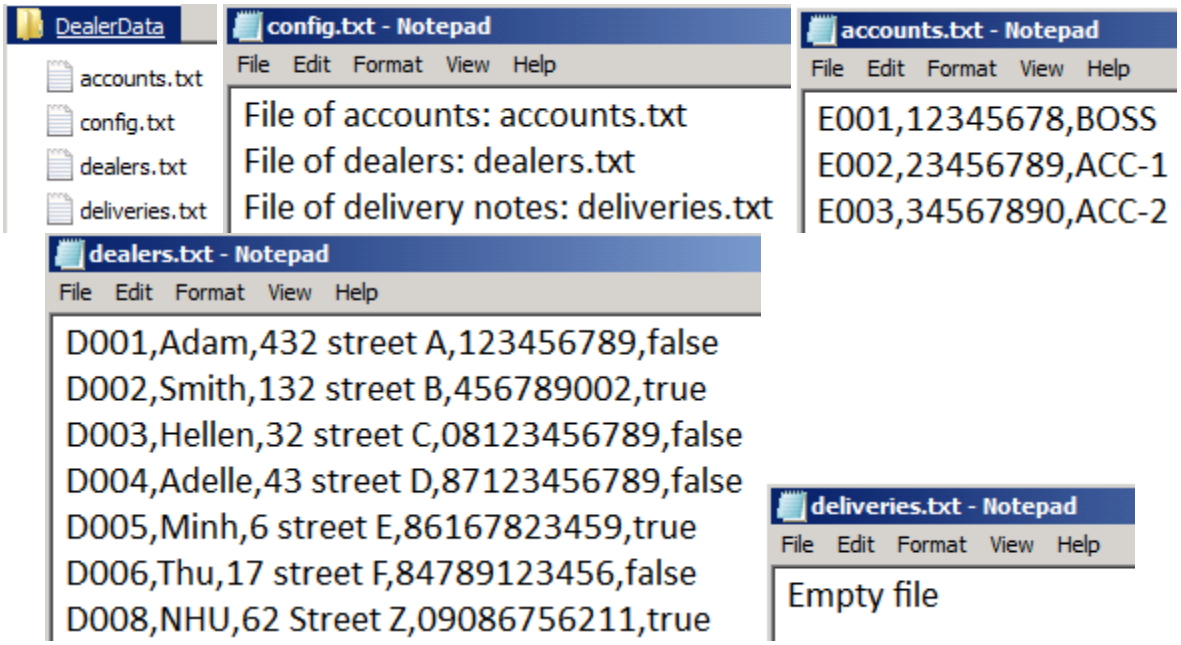
Type:	**Long Assignment**
Code:	**J1.L.P0016**
LOC:	**500**
Slot(s):	**N/A**

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.

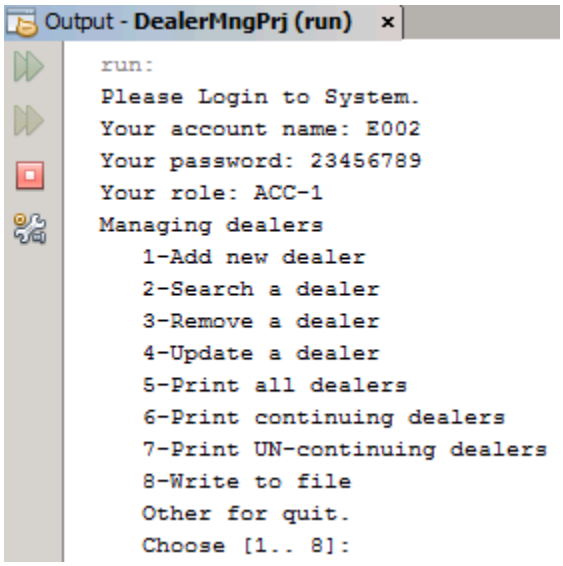
Each department has the following information: departmentID, name, createDate, lastUpdateDate

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:



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

DealerMngPrj	
Source Packages	
data	
Account.java	Class for an account
AccountChecker.java	Class supporting checking validity of an account
Config.java	Class for reading config file to get related filenames
Dealer.java	Class for a dealer
DealerList.java	Class for a list of dealers
mng	
LogIn.java	Class for Log in interface. The main() method is put in this class
Menu.java	Class for menu
tools	
MyTool.java	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 {
        private String accName; // ID
        private String pwd; // password
        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
```

```java
/* Class for Log In interface */
package mng;
import data.Account;
import data.AccountChecker;
import data.DealerList;
import tools.MyTool;

public class LogIn {
    private Account acc=null;
        // constructor
    public LogIn( Account acc){
        this.acc= acc;
    }
}
```

```java
package tools;

public class MyTool {
}
```

```java
/* Class for a \product dealer  */
package data;
import tools.MyTool;

public class Dealer implements Comparable<Dealer> {
    public static final char SEPARATOR = ',';
    public static final String ID_FORMAT = "D\\d{3}";
    public static final String PHONE_FORMAT = "\\d{9}|\\d{11}";
    private String ID; // template D000
    private String name; // dealers's name
    private String addr; // dealer's address
    private String phone; // 9 or 11 digits
    private boolean continuing; // whether this dealer still cooperates or not

    // constructor using 6 parameters - IMPLEMENT IT
    public Dealer(String ID, String name, String addr, String phone,
                  boolean continuing) {...7 lines }
}
```

```java
/* Class for a list of dealers */
package data;
import java.util.List;
import java.util.ArrayList;
import java.sql.Date;
import tools.MyTool;
import mng.LogIn;

public class DealerList extends ArrayList<Dealer>{
    LogIn loginObj= null;
    private static final String PHONEPATTERN = "\\d{9}|\\d{11}";
    private String dataFile ="";
    boolean changed = false; // whether data in the list changed or not

    // Contructor using logInObj as a parameter - IMPLEMENT IT
    public DealerList(LogIn loginObj) {...4 lines }

}
```

The Hint in Implementation – Step 2, Implementations in Details

2- <u>The MyTool class</u>

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

```java
package tools;
import java.sql.Date; // containing year, month,m day only
import java.text.SimpleDateFormat; // for converting string <--> Date
import java.util.Scanner; // for input data
import java.io.File;      // For checking a file
import java.io.FileReader;    // classes for reading data from a text file
import java.io.BufferedReader;
import java.io.FileWriter;    // classes for writing data to a text file
import java.io.PrintWriter;
import java.util.ArrayList;   // Class for a list
import java.util.List;        // Interface for a list
import java.text.ParseException;  // exception when parsing data from a string
import java.io.IOException;  // Exception when accessing file

public class MyTool {
    public static final Scanner SC = new Scanner(System.in);

    // Checking whether str matches a pattern or not
    // Use the method String.matches (regEx) - IMPLEMENT IT
    public static boolean validStr (String str, String regEx){...3 lines }

    /* Checking a password with minLen in which it contains at least a character,
          a number and 1 specific character
    .* : there may be one or more any character
       \\d : digit   \\W : [^a-zA-Z0-9] : it is not a character and not a digit
    */
    public static boolean validPassword (String str, int minLen){
        if(str.length()<minLen) return false;
        return str.matches(".*[a-zA-Z]+.*") && // AT LEAST 1 CHARACTER
               str.matches(".*[\\d]+.*") && // AT LEAST 1 DIGIT
               str.matches(".*[\\W]+.*");// AT LEAST 1 SPECIAL CHAR
    }

    // Date format: yyyy/MM/dd, MM/dd/yyyy, dd/MM/yyyy, ...
    // yyyy/dd/MM 2000/30/02 -> 2000/01/03 automatically
    // Date string will be changed to a valid date value automatically
    public static Date parseDate(String dateStr, String dateFormat){
        SimpleDateFormat dF = (SimpleDateFormat)SimpleDateFormat.getInstance();
        dF.applyPattern(dateFormat);
        try{
            long t = dF.parse(dateStr).getTime();
            return new Date(t);
        }
        catch(ParseException e){
            System.out.println(e);
        }
        return null;
    }

```

```java
    // Use the class SimpleDateFormat
    // Use the method applyPatter(str) to apply a specific format
    // Use the method format(date) to convert date -> String
    // IMPLEMENT IT
    public static String dataToStr(Date date, String dateFormat){...5 lines }

    // Conmvert bool string to boolean
    public static boolean parseBool(String boolStr){
        char c = boolStr.trim().toUpperCase().charAt(0);
        return (c=='1' || c=='Y' || c=='T');
    }
    // Tools for inputting data
    public static String readNonBlank(String message){
        String input ="";
        do{
            System.out.print(message + ": ");
            input = SC.nextLine().trim();
        }
        while (input.isEmpty());
        return input;
    }
    public static String readPattern(String message, String pattern)
        String input ="";
        boolean valid;
        do{
            System.out.print(message + ": ");
            input = SC.nextLine().trim();
            valid = validStr(input,pattern);
        }
        while (!valid);
        return input;
    }
    public static boolean readBool(String message){
        String input;
        System.out.print(message + "[1/0-Y/N-T/F]: ");
        input = SC.nextLine().trim();
        if (input.isEmpty()) return false;
        char c = Character.toUpperCase(input.charAt(0));
        return (c=='1' || c=='Y' || c=='T');
    }
```

```java
    /* Method for reading lines from text file
    Create an array list, named as list
    Open file
    While ( still read succesfully a line in the file){
        trim the line;
        if line is not empty, add line to the list
    }
    Close file
    return list;
    IMPLEMENT IT
    */
    public static List<String> readLinesFromFile (String filename){...23 lines }

     /* Method for writing a list to a text file line-by-line
     Open the file for writing
     For each object in the list, write th eobject to file
     Close the file
     IMPLEMENT IT
     */
     public static void writeFile(String filename, List list){...11 lines }

     // Test- It is optional
     public static void main (String[] args){
         // Phone: 9 or 11 digits - OK
         System.out.println("Tests with phone numbers:");
         System.out.println(validStr("012345678", "\\d{9}|\\d{11}"));
         System.out.println(validStr("01234567891", "\\d{9}|\\d{11}"));
         System.out.println(validStr("12345678", "\\d{9}|\\d{11}"));

         // Test password - OK
         System.out.println(validPassword("qwerty", 8)); // false
         System.out.println(validPassword("qwertyABC", 8)); // false
         System.out.println(validPassword("12345678", 8));  // false
         System.out.println(validPassword("qbc123456", 8));  // false
         System.out.println(validPassword("qbc@123456", 8)); // true
         // ID format D000 -> OK
         System.out.println("Tests with IDs:");
         System.out.println(validStr("A0001", "D\\d{3}"));
         System.out.println(validStr("10001", "D\\d{3}"));
         System.out.println(validStr("D0001", "D\\d{3}"));
         System.out.println(validStr("D101", "D\\d{3}"));
```

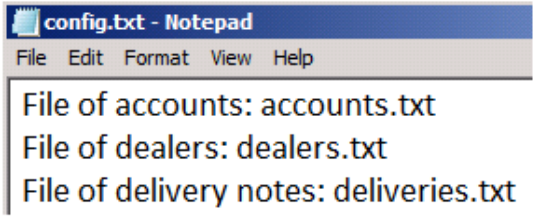
```java
            //Test date format -> OK
            Date d = parseDate("2022:12:07", "yyyy:MM:dd");
            System.out.println(d);
            System.out.println(dataToStr(d, "dd/MM/yyyy")); // test OK
            d = parseDate("12/07/2022", "MM/dd/yyyy");
            System.out.println(d);
            d = parseDate("2022/07/12", "yyyy/dd/MM");
            System.out.println(d);
            d = parseDate("2000/29/02", "yyyy/dd/MM");
            System.out.println(d);
            d = parseDate("2000/30/02", "yyyy/dd/MM");
            System.out.println(d);
            d = parseDate("2000/40/16", "yyyy/dd/MM");
            System.out.println(d);
            // Test iput data -> ok
            String input = readNonBlank("Input a non-blank string");
            System.out.println(input);// OK
            input = readPattern("Phone 9/11 digits","\\d{9}|\\d{11}");
            System.out.println(input);// OK
            input = readPattern("ID- format X00000","X\\d{5}");
            System.out.println(input);// OK
            boolean b = readBool("Input boolean");
            System.out.println(b);// OK

        }
    }//class MyTool
```

3- The Config Class

```java
/* Class for reading config.txt file*/
package data;
import java.util.List;
import tools.MyTool;
public class Config {
    private static final String CONFIG_FILE = "DealerData/config.txt";
    private String accountFile;
    private String dealerFile;
    private String deliveryFile;

    public Config() {
        readData();
    }
```



```java
14      private void readData(){
15          List<String> lines = MyTool.readLinesFromFile(CONFIG_FILE);
            for (String line: lines){
17              line = line.toUpperCase();
18              String[] parts = line.split(":");
                if (line.indexOf("ACCOUN")>=0)
20                  accountFile = "DealerData/" + parts[1].trim();
                else if (line.indexOf("DEALE")>=0)
22                  dealerFile = "DealerData/" + parts[1].trim();
                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

```java
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;
        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
```

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

```java
    // Check valiadity od an account
    public boolean check(Account acc){
        // Read data in file
        List<String> lines = MyTool.readLinesFromFile(accFile);
        // Traverse each line for checking
        for (String line: lines){
            String[] parts= line.split(this.SEPARATOR);
            if (parts.length<3) return false;
            if( parts[0].equalsIgnoreCase(acc.getAccName()) &&
                parts[1].equals(acc.getPwd()) &&
                parts[2].equalsIgnoreCase(acc.getRole()))
                return true;
        }
        return false;
    }
    // Test OK - It is optional
    public static void main(String[] args){
        AccountChecker aChk = new AccountChecker();
        Account acc = new Account("E001", "12345678", "BOSS");
        boolean valid = aChk.check(acc);
        System.out.println("Needs OK, OK?: " + valid);
        acc = new Account("E002", "23456789", "ACC-1");
        valid = aChk.check(acc);
        System.out.println("Needs OK: OK? " + valid);
        acc = new Account("E003", "123456789", "ACC-2");
        valid = aChk.check(acc);
        System.out.println("Needs NO OK, OK?: " + valid);

    }

}// class AccountChecker
```

5- The Dealer Class

```java
/* Class for a product dealer  */
package data;
import tools.MyTool;

public class Dealer implements Comparable<Dealer> {
    public static final char SEPARATOR = ',';
    public static final String ID_FORMAT = "D\\d{3}";
    public static final String PHONE_FORMAT = "\\d{9}|\\d{11}";
    private String ID; // template D000
    private String name; // dealers's name
    private String addr; // dealer's address
    private String phone; // 9 or 11 digits
    private boolean continuing; // whether this dealer still cooperates or not

    // constructor using 5 parameters - IMPLEMENT IT
    public Dealer(String ID, String name, String addr, String phone,
                  boolean continuing) {...7 lines }

    // constructor using a line using the separator ','
    public Dealer(String line) {
        String[] parts = line.split("" + this.SEPARATOR);
        ID = parts[0].trim(); // dealer ID
        name = parts[1].trim(); // dealers's name
        addr = parts[2].trim(); // dealer's address
        phone = parts[3].trim(); // 9 or 11 digits
        continuing = MyTool.parseBool(parts[4]);
    }
    // getters, setters- IMPLEMENT THEM
    public String getID() {...3 lines }
    public void setID(String ID) {...3 lines }
    public String getName() {...3 lines }
    public void setName(String name) {...4 lines }
    public String getAddr() {...3 lines }
    public void setAddr(String addr) {...3 lines }
    public String getPhone() {...3 lines }
    public void setPhone(String phone) {...3 lines }
    public boolean isContinuing() {...3 lines }
    public void setContinuing(boolean continuing) {...3 lines }

    @Override
    public String toString() {
        return ID + SEPARATOR + name + SEPARATOR +
               addr + SEPARATOR + phone + SEPARATOR +
               continuing + "\n";
    }
```

D001,Adam,432 street A,123456789,false

D001,Adam,432 street A,123456789,false

D001,Adam,432 street A,123456789,false

```
73          //Comparing tool: comparing based on their ID- IMPLEMENT IT
74          @Override
            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();
            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)dearler 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         /* Remove a dealer: Assign continuing = false -- IMPLEMENT IT
121         Input ID
122         pos = search(ID)
123         if (pos<0) output "Not fpound!)
124         else{
125             set field continuing of the pos(th) element to FALSE
126             output "Removed"
127             changed = true ; // data changed
128         }
129         */
130         public void removeDealer(){...11 lines }
141
```

```java
    // update a dealer
    // Only name, addr and phne can be changed
    // Only changing name is expressedm you do all remainders
    public void updateDealer(){
        System.out.print("Dealer's ID needs updating: ");
        String ID = MyTool.SC.nextLine();
        int pos = searchDealer(ID);
        if (pos<0) System.out.println("Dealer " + ID + " not found!")
        else {
            Dealer d = this.get(pos);
            String newName="";// Update name
            System.out.print("new name, ENTER for omitting: ");
            newName = MyTool.SC.nextLine().trim().toUpperCase();
            if (!newName.isEmpty()){
                d.setName(newName);
                changed = true;
            }
            // update addr - IMPLEMENT IT
             // update phone - IMPLEMENT IT


        }
    }
    // Print all dealers - IMPLEMENT IT
    public void printAllDealers(){
        if (this.isEmpty()) System.out.println("Empty List!");
        else System.out.println(this);
    }
    // Print all continuing dealers
    public void printContinuingDealers(){
        this.getContinuingList().printAllDealers();
    }
    // Print all un-continuing dealers - IMPLEMENT IT
    public void printUnContinuingDealers(){...3 lines }
```

```java
    // Write dealer list to file
    public void writeDealerToFile(){
        if (changed){
            MyTool.writeFile(dataFile, this);
            changed= false;
        }
    }

    // getters, setters - IMPLEMENT THEM
    public boolean isChanged() {...3 lines }
    public void setChanged(boolean changed) {...3 lines }
}// class DealerList
```

8- The LogIn Class

```java
/* Class for Log In interface */
package mng;
import data.Account;
import data.AccountChecker;
import data.DealerList;
import tools.MyTool;

public class LogIn {
    private Account acc=null; // account will log in
        // constructor
    public LogIn( Account acc){
        this.acc= acc;
    }
    /* Input data of an account - IMPLEMENT IT
       Create new Account
       return this account
    */
    public static Account inputAccount() {...13 lines }

    // getter

    public Account getAcc() {
        return acc;
    }

```

```java
38        // Main prgram
39        public static void main(String[] args){
40            Account acc = null; // account will login to system
41            boolean cont = false; // login again?
            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); // chexk 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();
                  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.