LAB 211 Assignment

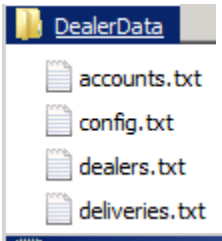
Type:	**Long Assignment**
Code:	**J1.L.P0016**
LOC:	**500**
Slot(s):	**N/A**

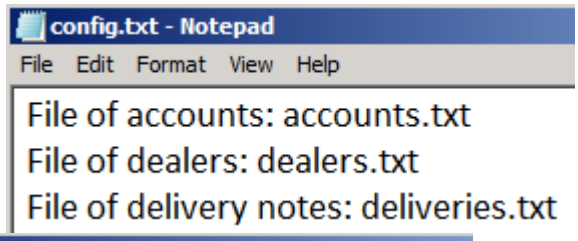
Title

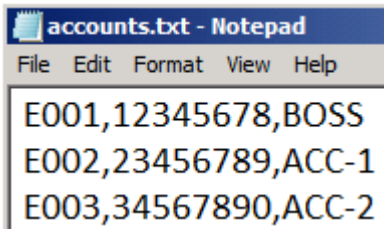
Dealers Management Program

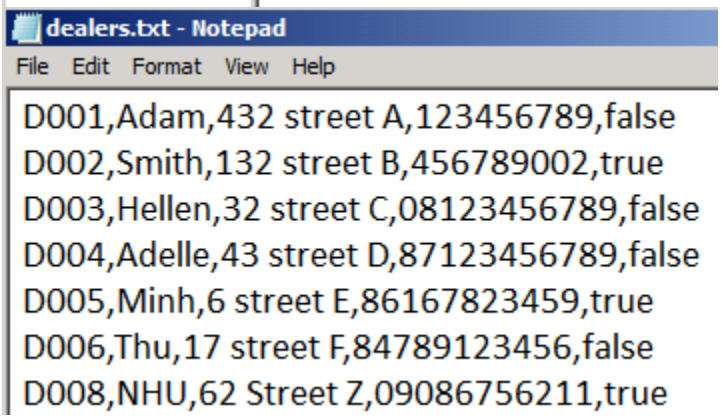
Background

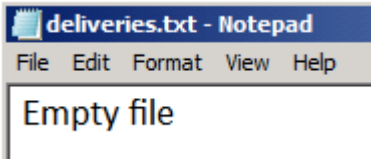
- AZW, a firm, needs a Java console program for managing it's product dealers. This program must support a basic security. Roles in the firm include:
- (1) Boss: Managing users
- (2) ACC-1: Managing dealers
- (3) ACC-2: Managing deliveries notes.
- Data files are supported as following:











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

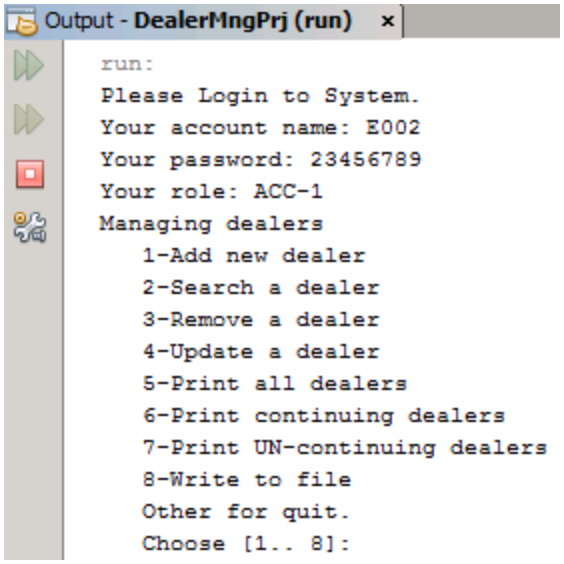
All users must be login to system to carry out appropriate management activities based on his/her role. At the moment, a program for managing dealers is required to develop.

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



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 AccountChecker.java Config.java Dealer.java DealerList.java mng LogIn.java Menu.java tools MyTool.java	Class for an account Class supporting checking validity of an account Class for reading config file to get related filenames Class for a dealer Class for a list of dealers Class for Log in interface. The main() method is put in this class Class for menu Class for programming common used basic methods

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- Class Account

```
/* Class for an account */
package data;

public class Account {
    private String accName; // ID
    private String pwd; // password
    private String role;
    //contrustor - IMPEMENT IT
    public Account(String accName, String pwd, String role) {...5 lines }
    // Getters-- IMPLEMENT THEM
    public String getAccName() {...3 lines }

    public String getPwd() {...3 lines }

    public String getRole() {...3 lines }

}// class Account
```

```
/* 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;
    }
}
```

```
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 }

}
```

Implementation – Step 2, Implementations in Details

2- Class MyTool

```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}"
*/
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 }

```

```java
    /* 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;
    }

    // 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;
     }
```

```java
    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');
    }
/* 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}"));
```

```
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 iput 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- Class Config

```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();
    }
   private void readData(){
      List<String> lines = MyTool.readLinesFromFile(CONFIG_FILE);
      for (String line: lines){
           line = line.toUpperCase();
           String[] parts = line.split(":");
           if (line.indexOf("ACCOUN")>=0)
              accountFile = "DealerData/" + parts[1].trim();
           else if (line.indexOf("DEALE")>=0)
              dealerFile = "DealerData/" + parts[1].trim();
           else if (line.indexOf("DELIVER")>=0)
              deliveryFile = "DealerData/" + parts[1].trim();
      }
   }
    // Getters- Implement IT
    public String getAccountFile() {...3 lines }

    public String getDealerFile() {...3 lines }

    public String getDeliveryFile() {...3 lines }

}//class Config
```

config.txt - Notepad

File Edit Format View Help

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

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

4- Class AccountChecker

```java
/* Class for checking validity of an account */
package data;

import tools.MyTool;
import java.util.List;

public class AccountChecker {
    private String accFile;
    private static String SEPARATOR=",";
    public AccountChecker() {
        setupAccFile();
    }
    private void setupAccFile(){
        Config cR = new Config();
        accFile = cR.getAccountFile();
    }
```

accounts.txt - Notepad

File Edit Format View Help

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;
    }
```

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

```
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- Class Dealer

```
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
              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
            public String toString() {
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
        +   public int compareTo(Dealer o) {...3 lines }
78      }//class Dealer
```

D001,Adam,432 street A,123456789,false

6- Class Menu

```
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- Class DealerList

```java
/* Class for a list of dealers */
package data;
import java.util.List;
import java.util.ArrayList;
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 }


    /* Load dealers form file
    Use MyTool to read lines from the data file, List lines
    For each line in lines, create a dealer using this line as parameter
    Add this created dealer to the list
    IMPLEMENT IT
    */
    private void loadDealerFromFile(){...7 lines }

    // initializing basic data in files
    public void initWithFile(){
        Config cR = new Config();
        dataFile = cR.getDealerFile(); // get file containing dealers
        loadDealerFromFile(); // load dealers from file
    }

```

```
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
```

```
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);
                    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
167                  // update phone - IMPLEMENT IT
168
182
183             }
184         }
185         // Print all dealers - IMPLEMENT IT
186         public void printAllDealers(){
187             if (this.isEmpty()) System.out.println("Empty List!");
188             else System.out.println(this);
189         }
190         // Print all continuing dealers
191         public void printContinuingDealers(){
192             this.getContinuingList().printAllDealers();
193         }
194         // Print all un-continuing dealers - IMPLEMENT IT
195         public void printUnContinuingDealers(){...3 lines }
198
```

```
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- Class LogIn

```
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      /* Input data of an account - IMPLEMENT IT
15         Create new Account
16         return this account
17      */
18      public static Account inputAccount(){...13 lines }
31
32      // getter
33
34      public Account getAcc() {
35          return acc;
36      }
37
```

```
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.