EECS 1510: Object Oriented Programming
Project 7 – Final Project – 220 Points

Due 11:00-1:00 p.m. in Palmer 2600, Monday May 1, 2017

Note 1: This project may be done alone or in pairs. For a pair, your partner MUST be different fromthat of Project 6; otherwise there is an automatic 60 point deduction.

Note 2: This final project has both a fixed part and an optional part. The optional part is for extra credit, and is somewhat open ended. Up to 120 extra points may be earned through extra credit.

Note 3: This document contains a solution to Project 6. You may use this code freely in you solution to Project 7.

Consider the inventory management application of Project 6. A solution to this project is given at the end of this document. You can use this code or your own code in Project 6 as a basis for this assignment. Either way is fine.

For the fixed part of this project (the 220 points) you must extend the program as follows:

- Add a delete command.
- Keep the inventory in alphabetical order. When a new item is to be added, put the entry into the appropriate location in the array.
- Construct a Graphical User Interface (GUI) for the user. Do not use JOptionPane. You must use JavaFX.
- Make the program object oriented. This means having a separate class for the internal representation of the inventory, with methods like add, list, or delete. It also means having a separate class for the GUI aspects of the program.

The program must follow the 10 Program Standards given with Project 4 (all previous projects are still posted in the folder "Previous Projects"). These standards are reposted in a file called

Program Standards for Java Course.doc

For the optional part of this project you may extend the program in various ways. Some areas for extension are as follows:

• **Usability and Aesthetics:** Make the GUI especially pleasing to see and use. One example would be to make the list command give a nice listing.

• **Human Engineering:** Provide good human engineering for the user. This means being very tolerant of user errors and making it easy to use. For example, you might give the user an option to name the inventory file, or you might check if the user tries to add another entry with the same name. Also, consider a simple Help command.

• **Reliability:** Make the program especially reliable. Try to make it so that the program will not crash even under incorrect inputs. For example, handle a missing file well or prevent an array out of bounds error.

• **Maintainability**: Make the program especially well structured and readable.

• **Functionality:** Enhance to functionality of the program in various ways, even small ways.

For functionality, one enhancement would be to check that a find or enter command actually has a non-null string for the name. A little more work would be to check the validity of an item's quantity. For instance, always verifying that an item's quantity is greater than or equal to zero. An obvious enhancement could be a delete command.

For another example, one could allow a partial match to find or delete an entry. For example, "F po" would match any entry with "pop" in the name, for example "Pop" or "Popcorn" or "Potato". You might use the function "find" in C++ for this feature.

Final Submission: Submitting a portfolio or dossier of sorts (a stapled or bound set of pages) containing

- A separate cover page (or pages) itemizing the grounds for extra credit. *It is your responsibility to itemize all grounds for extra credit.*
- A printout of your code,
- A printout from several consecutive runs, illustrating the various features of your program. For example, you must show that the file I/O works.
- A printout from the list command

In the sample runs, each of the commands "e", "f", "l", "d", and "q" should be illustrated.

A solution similar to Project 6.

```
import java.io.*;
```

```java
import java.util.*;

class Entry {
    public String name, quantity, note;
}

public class InventoryFor1510 {
  public static Entry[]  entryList;
  public static int     num_entries;
  public static Scanner  stdin = new Scanner(System.in);

public static void main(String args[]) throws Exception{
    int i;
    char C;
    String code, Command;

    entryList = new Entry[200];
    num_entries = 0;
    Command = null;
    C = ' ';
    readInventory("inventory.txt");

    System.out.println("Codes are entered as 1 to 8 characters.\nUse" +
            " \"e\" for enter," +
            " \"f\" for find," +
            " \"l\" for listing all the entries," +
            " \"q\" to quit.");

    while(C != 'q'){
        System.out.print("Command: ");
        Command = stdin.next();
        C = Command.charAt(0);
        switch (C) {
        case 'e':
            addItem();  break;
        case 'f':
            code = stdin.next();
            stdin.nextLine();
            i = index(code);
            if (i >= 0) displayEntry(entryList [i]);
            else        System.out.println("**No entry with code " + code); break;
        case 'l':
            listAllItems(); break;
        case 's':
            sortList(); break;
        case 'q':
            CopyInventoryToFile("inventory.txt");
            System.out.println("Quitting the application. All the entries are "
                    + "stored in the file inventory.txt"); break;
        default:
            System.out.println("Invalid command. Please enter the command again!!!");                 }
    }
}

public static void readInventory(String FileName) throws Exception {
    File F;
    F  = new File(FileName);
    Scanner S = new Scanner(F);

    while (S.hasNextLine()) {
        entryList [num_entries]= new Entry();
        entryList [num_entries].name   = S.next();
        entryList [num_entries].quantity = S.next();
        entryList [num_entries].note   = S.nextLine();
        num_entries++;
    }
    S.close();
}

public static void addItem() {
    String name = stdin.next();
    String quantity;
    stdin.nextLine();
    entryList [num_entries]     = new Entry();
    entryList [num_entries].name = name;

    System.out.print("Enter Quantity: ");
    quantity = stdin.nextLine();
    entryList [num_entries].quantity = quantity;

    System.out.print("Enter Notes: ");
    entryList [num_entries].note = stdin.nextLine();
    num_entries++;
}


public static int index(String Key) {
// Function to get the index of a key from an array
// if not found, returns -1
    for (int i=0; i < num_entries; i++) {
        if (entryList [i].name.equalsIgnoreCase(Key))
                return i; // Found the Key, return index.
    }
    return -1;
}

public static void displayEntry(Entry item) {
    System.out.println("--"+ item.name+"\t"+
              item.quantity+"\t"+
              item.note);
}

public static void listAllItems() {
    int i = 0;
    while (i < num_entries) {
        displayEntry(entryList [i]);
        i++;
    }
}

public static void CopyInventoryToFile(String FileName) throws Exception{
    FileOutputStream out = new FileOutputStream(FileName);
    PrintStream P       = new PrintStream( out );

    for (int i=0; i < num_entries; i++) {
        P.println(entryList [i].name + "\t" + entryList [i].number +
                  "\t" + entryList [i].note);
    }
}
}
```