COMP 348: ASSIGNMENT 3

DUE DATE: 11:55 PM, Sunday, April 8

Note that assignments must be submitted on time in order to received full value. Appropriate late penalties (< 1 hour = 10%, < 24 hours = 25%) will be applied, as appropriate.

DESCRIPTION: It's time to try a little functional programming. In this case, your job will be to develop a very simple Sales Order application. REALLY simple. In fact, all it will really do is load data from a series of three disk files. This data will then form your Sales database. Each table will have a "schema" that indicates the fields inside. So your DB will look like this:

cust.txt: This is the data for the customer table. The schema is

<custID, name, address, phoneNumber>

An example of the cust.txt disk file might be:

```
1|John Smith|123 Here Street|456-4567
2|Sue Jones|43 Rose Court Street|345-7867
3|Fan Yuhong|165 Happy Lane|345-4533
```

Note that no error checking is required for any of the data files. You can assume that they have been created properly and all fields are present. Each field is separated by a "|" and contains a non-empty string.

prod.txt: This is the data for the product table. The schema is

<prodID, itemDescription, unitCost>

An example of the prod.txt disk file might be:

```
1|shoes|14.96
2|milk|1.98
3|jam|2.99
4|gum|1.25
5|eggs|2.98
6|jacket|42.99
```

sales.txt: This is the data for the main sales table. The schema is

<salesID, custID, prodID, itemCount>

An example of the sales.txt disk file might be:

```
1|1|1|3
2|2|2|3
3|2|1|1
4|3|3|4
```

The first record (salesID 1), for example, indicates that John Smith (customer 1) bought 3 pairs of shoes (product 1). Again, you can assume that all of the values in the file (e.g., custID, prodID) are valid.

So now you have to do something with your data. You will provide the following menu to allow the user to perform actions on the data:

```
*** Sales Menu ***
------------------

1. Display Customer Table
2. Display Product Table
3. Display Sales Table
4. Total Sales for Customer
5. Total Count for Product
6. Exit

Enter an option?
```

The options will work as follows

1. You will display the contents of the Customer table. The output should be similar (not necessarily identical) to

   1: ["John Smith" "123 Here Street" "456-4567"]
   2: ["Sue Jones" "43 Rose Court Street" "345-7867"]
   3: ["Fan Yuhong" "165 Happy Lane" "345-4533"]

2. Same thing for the prod table.

3. The sales table is a little different. ID values aren't very useful for viewing purposes, so the custID should be replaced by the customer name and the prodID by the product description, as follows:

   1: ["John Smith" "shoes" "3"]
   2: ["Sue Jones" "milk" "3"]
   3: ["Sue Jones" "shoes" "1"]
   4: ["Fan Yuhong" "jam" "4"]

4. For option 4, you will prompt the user for a customer name. You will then determine the total value of the purchases for this customer. So for Sue Jones you would display a result like:

   Sue Jones: $20.90

   This represents 1 pair of shoes and 3 cartons of milk.

5. Here, we do the same thing, except we are calculating the sales count for a given product. So, for shoes, we might have:

   Shoes: 4

   This represents three pairs for John Smith and one for Sue Jones.

6. Finally, if the Exit option is entered the program will terminate with a "Good Bye" message. Otherwise, the menu will be displayed again.

So that's the basic idea. There are a few final points to keep in mind:

1. You do not want to load the data each time a request is made. So before the menu is displayed the first time, your data should be loaded and stored in appropriate data structures (you can do this any way that you like).
2. This is a Clojure assignment, not a Java assignment. So Java should not be used for any main functionality. It might be necessary to use Java classes to convert text to numbers in order to do the sales calculations, but Java should not be used for much more than that.
3. The I/O in this assignment is trivial. While it is possible to use complex I/O techniques, it is not necessary to read the text files. Instead, you should just use "slurp", a Clojure function that will read a text file into a string. For the input from the user the "read-line" function can be used.
4. Do not worry about efficiency. There are ways to make this program (both the data management and the menu) more efficient, but that is not our focus here. I just want you to use basic functionality to try to get everything working.

DELIVERABLES: Your submission will have just 2 source files. The "main" file will be called `menu.clj` and, as the name implies, it will be the starting point for the program and will provide the interface to the user. The second file will be called `db.clj` and will contain all of the data management code. It will essentially function as a library that will be loaded by the `menu.clj` file. Do not include any data files, as the markers will provide their own.

Once you are ready to submit, place the .clj files into a zip file. The name of the zip file will consist of "a3" + last name + first name + student ID + ".zip", using the underscore character "_" as the separator. For example, if your name is John Smith and your ID is "123456", then your zip file would be combined into a file called a3_Smith_John_123456.zip". The final zip file will be submitted

through the course web site on Moodle. You simply upload the file using the link on the assignment web page.

FINAL NOTE: While you may talk to other students about the assignment, all code must be written individually. Any sharing of assignment code will likely lead to an unpleasant outcome.

Good Luck