Java OO Concepts: Mastery

Flooring Mastery Project: Phase 1 - Retail Interface

Flooring Mastery Project: Phase 1 - Retail Interface

Objective

The goal of this mastery project is to create an application that allows for reading and writing flooring orders for SWC Corp.

Requirements

The application will have a configuration file to set the mode to either Test or Prod. If the mode is Test, then the application should not save any order data - order data should only be stored in memory. If the mode is prod, then the application should read and write order information from a file called Orders_MMDDYYYY.txt

An Order consists of an order number, customer name, state, tax rate, product type, area, cost per square foot, labor cost per square foot, material cost, labor cost, tax, and total.

Taxes and Product Type information can be found in the Data/Taxes.txt and Data/Products.txt files. The customer state and product type entered by a user must match items in the data. This information is read in from the file in both production and test mode.

Orders_06012013.txt is a sample row of data for one order.

For the UI, it should create a menu to prompt the user for what they would like to do:

```
**************************************************************************

*                          Flooring Program

*

* 1. Display Orders

* 2. Add an Order

* 3. Edit an Order

* 4. Remove an Order

* 5. Save Current Work

* 6. Quit

*

**************************************************************************
```

Display Orders

Display orders will query the user for a date and will display the orders for that date. If no orders exist for that date, it will display an error message and return the user to the main menu.

Add an Order

Add an order will query the user for each piece of order data. At the end it will display a summary of the data entered and ask the user to commit (Y/N). If yes, the data will be written to the orders list. If no the data will be discarded and the user returned to the main menu.

The system should generate an order number for the user based on the next # in the file (so if there are two orders, the next order should be #3)

Edit an Order

Edit will query the user for a date and order number. If the order exists for that date it will query the user for each piece of order data but display the existing data. If the user enters something new it will replace that data, if they hit enter without entering data it will leave the existing data in place. For example:

```
Enter customer name (Wise):
```

If the user enters a new name, the name will replace Wise, otherwise it will leave it as-is.

Remove an Order

For removing an order, the system should ask for the date and order number. If it exists the system should display the order information and prompt the user if they are sure. If yes it should be removed from the list.

Anytime a user enters invalid data, the system should ask them again until they enter valid data.

As in our previous labs, all existing order data will be read in from files when the program starts and will be written to the files when the program exits. This program also allows the user to save their current work when requested (see menu option 5 above).

Here are the rules:

1. We are using an enterprise architecture for this project. Thus your code must be organized into reasonable classes. You are to draw up a class model and flow chart each workflow before proceeding with writing code.

2. As an enterprise architecture, we must layer our code. The "model" package may only contain classes that have data members (properties), the "data" package may only contain classes that read and write data to files (or hold them in memory in test mode), the "controller" package contains classes that orchestrate the program and perform all business calculations, and you will use the ConsoleIO library to handle all IO for the user.

3. You will work with a partner/team on this project. You may not write 100% of the code for the project if partnered. Everyone is expected to spend an equal amount of time ‘driving’.

4. You must unit test where it makes sense to do so.

5. You may ask other teams to view their code - you may not simply copy and paste another teams solution to a problem into your project.

6. At the beginning and end of each day, we will do a stand-up meeting to review what has been accomplished, what is left to do, and places where we are getting stuck.

Here are some tips:

1. Build this application following the process outlined in the Agile Approach Checklist for Console Applications.

2. If you get stuck, ask the instructor for general guidance on what you should be looking for. In this project the instructor will not write the code for you, but advice is always free.