Unit 5: Mastery

Mastery Project 1: Flooring Mastery Requirements

Mastery Project 1: Flooring Mastery Requirements

Goal

The goal of this mastery project is to create an application that reads and writes flooring orders for SG 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 provide sample data and allow for reading order information from the console user. This data should not be saved in between runs of the application. 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.

Our company receives product and tax information from suppliers and the state, so we do not know the format of those files yet; until we can get you the files, use interfaces and mock repositories. Use the Application Configuration file to switch from file mode to mock mode. The customer state and product type entered by a user must match items in the data. In mock mode, these data items can be hard-coded, but in production mode, it must read from the files. (Hint: interface the repositories)

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

We do not know the sample order for products and taxes, but they will have the following fields:

ProductType string

CostPerSquareFoot decimal

LaborCostPerSquareFoot decimal

StateAbbreviation string

StateName string

TaxRate decimal

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. Quit

*

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

Display orders will ask the user for a date and load the orders.txt file for that date if it exists. If the date does not exist, it will display an error message and return the user to the main menu.

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 (and saved to file in production mode). 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 will ask the user for a date and order number. If the order exists in the file 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, and otherwise it will leave the name as is.

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 prompt them again until they enter valid data. Validate all reasonable data. Errors should be logged to the file system in a log.txt file.

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 flowchart each workflow before proceeding to write code.

2. As an enterprise architecture, we must layer our code. The Models class library may only contain classes that have data members (properties), the data class library may only contain classes that read and write data to files (or hold them in memory in test mode), the operations class library commands the data classes to load and save data and performs all calculations, and the UI console application is in charge of all display and user input validation (in the case of validating state/product entry, it will need to ask the operations layer if the data is valid).

3. Your projects will be:

   FlooringProgram.UI

   FlooringProgram.BLL

   FlooringProgram.Models

   FlooringProgram.Data

   FlooringProgram.Tests

   BLL is where all business logic and calculations will occur. Models is for your DTOs and interface definitions. Data is for file manipulation. You must use interfaces to connect your BLL to the Data layer. Make both a File repository and a Mock Data repository for each file.

4. You will work with a partner on this project. You may not write 100% of the code for the project if partnered. You are expected to spend 50% of the time "driving."

5. You should try to unit test where it makes sense to do so.

6. You may ask other teams to view their code. If you do ask another team to view their code, you must explain to the instructor's satisfaction what their code is doing before you may copy it.

Here are some tips

1. Work on pieces of functionality in isolation. For example, when adding an order, write the UI piece to prompt and gather data first and then once that is working, worry about hooking it up to the operations and data layer.

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.

And some research:

1. You will need to look up how to work with application configuration files for the Dev/Prod settings.

2. If your team finishes early, I want you to make your data layer injectable into the Operations layer using a framework like Unity or Ninject (research, choose, and implement).