## Flooring Mastery Assessment

Contents

[Architectural Guidance 2](#_Toc128409550)

[Rules of Flooring Mastery 2](#_Toc128409551)

[Requirements 2](#_Toc128409552)

[Orders Files & Format 2](#_Toc128409553)

[Tax File & Format 3](#_Toc128409554)

[Products File & Format 3](#_Toc128409555)

[User Stories 5](#_Toc128409556)

[Display Orders 5](#_Toc128409557)

[Add an Order 5](#_Toc128409558)

[Edit an Order 6](#_Toc128409559)

[Remove an Order 6](#_Toc128409560)

[Export All Data 6](#_Toc128409561)

[Quit 6](#_Toc128409562)

[Stretch Goal: Export Data 7](#_Toc128409563)

[Additional Guidance 7](#_Toc128409564)

[Tips 8](#_Toc128409565)

[Criteria 8](#_Toc128409566)

## Architectural Guidance

Use N-tier development and MVC principles in structuring code to handle products, taxes, and orders appropriately. Your application should follow the MVC pattern presented in the course.

Use unit tests and integration tests to ensure that your application's data layers and business logic (service layer) code is covered.

## Rules of Flooring Mastery

1. We are using an enterprise MVC architecture for this project, so your code must be organized into reasonable classes. You will draw a UML class diagram and a high-level flowchart before proceeding with writing code.
2. For an enterprise architecture, the code must have layers:

|  |  |
| --- | --- |
| Package / Class | Contents |
| model package | Classes that have data members (properties). |
| dao package | Classes that are responsible for persisting data. |
| controller package | Classes that orchestrate the program. |
| view package | Classes that interact with the user. |
| service package | Service layer components |
| UserIO class | Handle all console IO for the user. |

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

## Requirements

Sample Data Files: [**SampleFileData.zip**](https://the-software-guild.s3.amazonaws.com/bootcamp-v2.6/oop-java/mastery/SampleFileData.zip)

The application will use three separate file formats for information.

### Orders Files & Format

1. All orders are stored within an Orders folder.
2. A new order file is created for each sales day.
3. The file name is guaranteed to be unique for that day because the date is part of the file name. When creating new order files the file name should be in this format: Orders\_MMDDYYYY.txt.
4. With this format, the order file for August 21st, 2025 would be named Orders\_08212025.txt.
5. The order file should contain a header row.

Data rows will contain the following fields and should be in this order:

|  |  |
| --- | --- |
| **Data** | **Datatype** |
| OrderNumber | Integer |
| CustomerName | String |
| State | String |
| TaxRate | BigDecimal |
| ProductType | String |
| Area | BigDecimal |
| CostPerSquareFoot | BigDecimal |
| LaborCostPerSquareFoot | BigDecimal |
| MaterialCost | BigDecimal |
| LaborCost | BigDecimal |
| Tax | BigDecimal |
| Total | BigDecimal |

Sample line from an orders file:

1,Ada Lovelace,CA,25.00,Tile,249.00,3.50,4.15,871.50,1033.35,476.21,2381.06

Orders\_06012013.txt is a sample file which is included in the sample download above.

Some of the order fields are calculations:

* MaterialCost = (Area \* CostPerSquareFoot)
* LaborCost = (Area \* LaborCostPerSquareFoot)
* Tax = (MaterialCost + LaborCost) \* (TaxRate/100)
  + Tax rates are stored as whole numbers
* Total = (MaterialCost + LaborCost + Tax)

### Tax File & Format

The tax information found in Data/Taxes.txt. This file will contain the following fields:

1. StateAbbreviation – String
2. StateName – String
3. TaxRate – BigDecimal

Here is a sample line from the taxes file:

TX,Texas,4.45

### Products File & Format

The current product information can be found in Data/Products.txt. It contains the following fields:

1. ProductType – String
2. CostPerSquareFoot – BigDecimal
3. LaborCostPerSquareFoot – BigDecimal

Here is a sample line from the products file:

Tile,3.50,4.15

## User Stories

The UI should start with 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. Export All Data

\* 6. Quit

### Display Orders

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

### Add an Order

To add an order will query the user for each piece of order data necessary:

1. **Order Date** – Must be in the future
2. **Customer Name** – May not be blank and is limited to characters [a-z][0-9] as well as periods and comma characters. "Acme, Inc." is a valid name.
3. **State** – Entered states must be checked against the tax file. If the state does not exist in the tax file, we cannot sell there. If the tax file is modified to include the state, it should be allowed without changing the application code.
4. **Product Type** – Show a list of available products and pricing information to choose from. Again, if a product is added to the file it should show up in the application without a code change.
5. **Area** – The area must be a positive decimal. Minimum order size is 100 sq ft.

The remaining fields are calculated from the user entry and the tax/product information in the files.

Show a summary of the order once the calculations are completed and prompt the user as to whether they want to place the order (Y/N).

If yes, the data will be added to in-memory storage.

If no, simply return to the main menu.

The system should generate an order number for the user based on the next available order # (so if there are two orders and the max order number is 4, the next order number should be 5).

### Edit an Order

Edit will ask the user for a date and order number. If the order exists for that date, it will ask 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 the user hits Enter without entering data, it will leave the existing data in place. For example:

Enter customer name (Ada Lovelace):

If the user enters a new name, the name will replace Ada Lovelace. If the user hits enter without entering any data, it will leave the data as-is.

Only certain data is allowed to be changed:

* CustomerName
* State
* ProductType
* Area

If the state, product type, or area are changed, the order will need to be recalculated. Order date may not be changed!

After querying for each editable field display a summary of the new order information and prompt for whether the edit should be saved. If yes, replace the data in the file then return to the main menu. If no, do not save and return to the main menu.

### 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.

### Export All Data

When the user selects this option, the system should export all existing data to a file that exists independently of the application itself.

### Quit

On quit, exit the application.

## Stretch Goal: Export Data

**This step is optional.** Complete this step only after all other steps work as required.

Selecting this option should save all active orders files to a file called *DataExport.txt* within a *Backup* folder. Exporting the order data multiple times should overwrite the data within the file with the latest active order information. However, unlike other order files, an order's line item in this DataExport file should also include the date in MM-DD-YYYY format, and the file's header should reflect this addition.

For example, here is a line of an example export, showcasing an order on June 12th, 2013:

1,Ada Lovelace,CA,25.00,Tile,249.00,3.50,4.15,871.50,1033.35,476.21,2381.06,06-12-2013

## Additional Guidance

* Any time a user enters invalid data, the system should ask them again until they enter valid data. A user should not be able to crash the application with invalid data.
* We are using a layered architecture for this project. Thus your code must be organized into reasonable well-named classes and split into correct packages.
* Please submit your project plan in addition to your code. This can be flowcharts, outlines, pseudocode, or any other documentation and planning technique.
* You must unit test your DAO and ServiceLayers.

## Tips

1. Build this application following the process outlined in the *Agile Approach Checklist for Console Applications* (as specified in the Rules of Flooring Mastery section above). This process will help you approach the project in a logical and orderly manner.
2. If you get stuck, ask an instructor for general guidance on what you should be looking for. The instructor will not write the code for you, but advice is always free.
3. There are places where this specification has been left open to interpretation: **this is intentional**. When on the job you will frequently have to ask clarifying questions to business stakeholders. Be sure to ask an instructor for general guidance on any features you do not understand.
4. Try to work on pieces of functionality in isolation and test them before moving on. Working on too large of a chunk at once leads to difficulties in troubleshooting and poor separation of concerns.
5. Think, then code. If you do not have a plan for each workflow, you will not be able to complete this project.
6. Use prior code samples and assessments as inspiration for this project. At this point in the course, you should be aware that there are patterns for creating a layered data-driven application.

## Criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Criteria** | **Meets Expectations** | **Needs Improvement** | **No Credit** |
| 1 | Specifications | ***20*** | ***10*** | ***0*** |
| 2 | Code Style | ***10*** | ***5*** | ***0*** |
| 3 | DI and MVC | ***5*** | ***3*** | ***0*** |
| 4 | I/O Operations | ***10*** | ***5*** | ***0*** |
| 5 | Data Types | ***5*** | ***3*** | ***0*** |
| 6 | Service Layer | ***5*** | ***3*** | ***0*** |
| 7 | Lambdas and Streams | ***5*** | ***3*** | ***0*** |
| 8 | Unit Tests | ***5*** | ***3*** | ***0*** |
| 9 | Data Storage | ***5*** | ***3*** | ***0*** |
| 10 | Input Validation | ***5*** | ***3*** | ***0*** |
| 11 | Exception Handling | ***5*** | ***3*** | ***0*** |
| 12 | Basic Concepts | ***5*** | ***3*** | ***0*** |
| 13 | Intermediate Concepts | ***5*** | ***3*** | ***0*** |
| 14 | Application Architecture: | ***5*** | ***3*** | ***0*** |