An application that can read and write flooring orders for TSG Corp.

- N-tier/MVC architecture, including the use of a service layer
- Interfaces
- Spring dependency Injection
- Unit Testing

Draw a UML class diagram and high-level flowchart before proceeding with writing code.

Layers:

- Model package
  - Contain classes that have data members (properties).
- dao package
  - Contain classes that are responsible for persisting data.
- Controller package
  - Contain classes that orchestrate the program.
- View package
  - Contain classes that interact with the user.
- Service package
  - Contain the service layer components.
- UserIO class
  - Along with the view component, Handle all console IO for the user.

3 separate file formats for information:

- Backup
  - DataExport.txt

```
OrderNumber,CustomerName,State,TaxRate,ProductType,Area,CostPerSquareFoot,LaborCostPerSquareFoot,MaterialCost,LaborCost,Tax,Total,OrderDate
1,Ada Lovelace,CA,25.00,Tile,249.00,3.50,4.15,871.50,1033.35,476.21,2381.06,06-01-2013
```

- Data
  - Products.txt

```
ProductType,CostPerSquareFoot,LaborCostPerSquareFoot
Carpet,2.25,2.10
```

  - Taxes.txt

```
State,StateName,TaxRate
TX,Texas,4.45
```

- Orders
  - Orders_06012013.txt

```
OrderNumber,CustomerName,State,TaxRate,ProductType,Area,CostPerSquareFoot,LaborCostPerSquareFoot,MaterialCost,LaborCost,Tax,Total
1,Ada Lovelace,CA,25.00,Tile,249.00,3.50,4.15,871.50,1033.35,476.21,2381.06
```

  - Orders_06022013.txt
  - Orders_mmddyyyy.txt

Orders:

- All orders are stored within an Orders folder.
- New order file is created for each sales day.
- Date is part of the file name.
    - Orders_MMDDYYYY.txt.
- File contains:
    - A header row.
    - Data rows:
        - OrderNumber – Integer
        - CustomerName – String
        - State – String
        - TaxRate – BigDecimal
        - ProductType – String
        - Area – BigDecimal
        - CostPerSquareFoot – BigDecimal
        - LaborCostPerSquareFoot – BigDecimal
        - MaterialCost – BigDecimal
            - (Area * CostPerSquareFoot)
        - LaborCost – BigDecimal
            - (Area * LaborCostPerSquareFoot)
        - Tax – BigDecimal
            - (MaterialCost + LaborCost) * (TaxRate/100)
        - Total – BigDecimal
            - (MaterialCost + LaborCost + Tax)

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

Tax:

- Found in Data/Taxes.txt
- Contain the following fields:
    - StateAbbreviation – String
    - StateName – String
    - TaxRate – BigDecimal

```
TX,Texas,4.45
```

Products:

- Found in Data/Products.txt.
- Contain the following fields:
    - ProductType – String
    - CostPerSquareFoot – BigDecimal
    - LaborCostPerSquareFoot – BigDecimal

```
Tile,3.50,4.15
```

User Stories: menu to prompt the user

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   * <<Flooring Program>>
   * 1. Display Orders
   * 2. Add an Order
   * 3. Edit an Order
   * 4. Remove an Order
   * 5. Export All Data
   * 6. Quit
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Menu:

1. Display orders
    a. Ask the user for a date : Date
    b. Display the orders for that date.
    c. If no orders exist for that date,
        i. Display an error message: **NoOrderException**
        ii. Return the user to the main menu.
2. Add an order
    a. Query the user for each piece of order data necessary:
        i. Order Date: Date
            1. Must be in the future
        ii. Customer Name: String
            1. May not be blank,
            2. Allowed to contain [a-z][0-9] as well as periods and comma characters. "Acme, Inc." is a valid name.
        iii. State: String: 2 char,
            1. Entered states must be checked against the tax file.
            2. If the state does not exist in the tax file we cannot sell there.
            3. Tax file modification should be allowed without changing the application code.
        iv. Product Type: int from list of products
            1. List of available products and pricing information to choose from.
            2. If a product is added to the file it should show up in the application without a code change.
        v. Area: BigDecimal (Scale = 2, HALF_UP)
            1. The area must be a positive decimal.
            2. Minimum order size is 100 sq ft., no max
    b. 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). String
        i. If yes, the data will be added to in-memory storage.
        ii. If no, simply return to the main menu.
    c. Generate an order number for the user based on the next available order # INY

3. Edit an order
    a. Ask the user for a date and order number. Date, int
    b. If the order exists for that date,
        i. Ask the user for each piece of order data but display the existing data.
        ii. If the user enters something new,
            1. Replace that data;
        iii. if the user hits Enter without entering data,
            1. Leave the existing data in place.
    c. If the order does not exists for that date,
        i. Throw **NoOrderException** , return to main menu
    d. Data allowed to be changed:
        i. CustomerName
        ii. State
        iii. ProductType
        iv. Area
    e. If state, product type, or area are changed,
        i. Order will need to be recalculated.
    f. Order date may not be changed!
    g. Display a summary of the new order information
        i. Prompt for whether the edit should be saved.
        ii. If yes, replace the data in the file then return to the main menu.
        iii. If no, do not save and return to the main menu.
4. Remove an order
    a. Ask for the date and order number.
    b. If it exists,
        i. the system should display the order information
        ii. Prompt the user if they are sure.
        iii. If yes, it should be removed from the list.
    c. If the order does not exists for that date,
        i. Throw **NoOrderException** , return to main menu
5. Export all data
    a. Save all active orders files to a file called DataExport.txt within a Backup folder.
    b. Overwrite the data within the file with the latest active order information.
    c. 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.
    d. Save all orders from different files to the backup file
6. Quit
    a. Exit the application.

Follow Agile Approach Checklist for Console Applications

Design Steps:

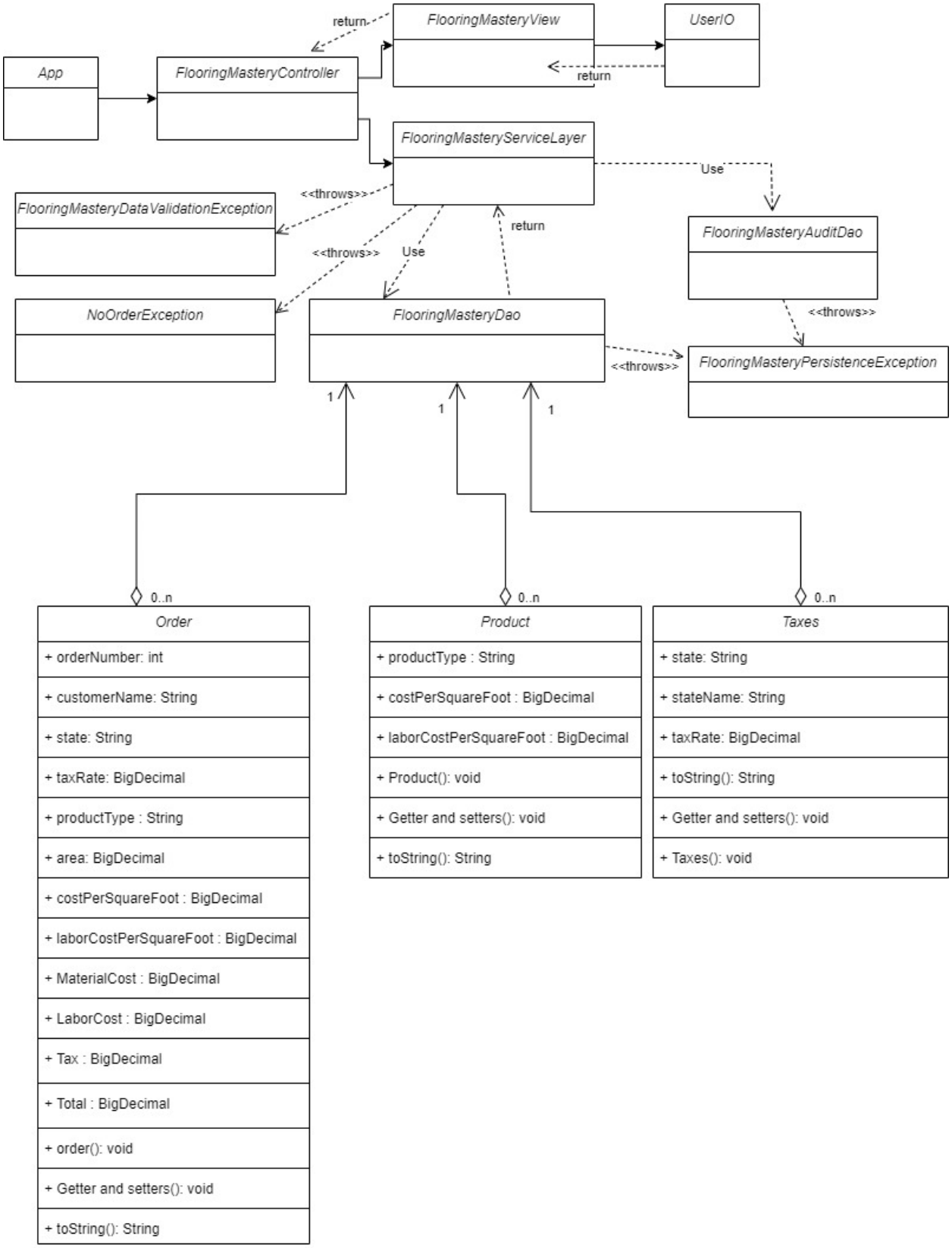
0. Create user stories from the problem statement and/or requirements.
    - 1.Display Orders
    - 2. Add an Order
    - 3. Edit an Order
    - 4. Remove an Order
    - 5. Export All Data
    - 6. Quit
1. Analyze the problem statement and identify the required classes.
    - Order(DTO)
    - Product(DTO)
    - Taxes(DTO)
    - FlooringMasteryDao(data access object)
    - UserIO (helper used by the View class to interact with the console)
    - FlooringMasteryView class (used by Controller to handle user interaction)
    - FlooringMasteryServiceLayer
    - FlooringMasteryController (this class orchestrates the program)
    - App (this class has a main method)
2. Flesh out the classes by defining properties and methods for each.
    - Order
        - Constructor()
        - Getter and setters()
        - toString()
        - Variables:
            - OrderNumber – Integer
            - CustomerName – String
            - State – String
            - TaxRate – BigDecimal
            - ProductType – String
            - Area – BigDecimal
            - CostPerSquareFoot – BigDecimal
            - LaborCostPerSquareFoot – BigDecimal
            - MaterialCost – BigDecimal: (Area * CostPerSquareFoot)
            - LaborCost – BigDecimal: (Area * LaborCostPerSquareFoot)
            - Tax – BigDecimal: (MaterialCost + LaborCost) * (TaxRate/100)
            - Total – BigDecimal: (MaterialCost + LaborCost + Tax)
    - Product
        - Constructor()
        - Getter and setters(), toString()
        - Variables:
            - ProductType -String
            - CostPerSquareFoot - BigDecimal
            - LaborCostPerSquareFoot - BigDecimal

- Taxes
    - Constructor()
    - Getter and setters()
    - toString()
    - Variables:
        - State,
        - StateName,
        - TaxRate
- FlooringMasteryDao:
    - Constructor()
    - displayOrders(Date date)
    - addOrder(Order order)
    - editOrder(Date date, int orderNumber)
    - removeOrder(Date date, int orderNumber)
    - exportData()
    - saveOrder(Order order, date)
    - loadProducts()
    - loadTaxes()
    - loadOrders()
    - saveOrders()
    - Variables:
        - Filenames
        - Delimiter
        - Map taxes <String state, Taxes>
        - Map products <String ProductType, Product>
        - Map orders <int orderID, Orders>
- UserIO:
    - void print(String message);
    - String readString(String prompt);
    - int readInt(String prompt);
    - int readInt(String prompt, int min, int max);
    - double readDouble(String prompt);
    - double readDouble(String prompt, double min, double max);
    - float readFloat(String prompt);
    - float readFloat(String prompt, float min, float max);
    - long readLong(String prompt);
    - long readLong(String prompt, long min, long max);

- FlooringMasteryView:
    - Constructor()
    - Variables:
        - io
    - Display main menu
    - Display banners
        - Display orders
        - Add an order
        - Edit an order
        - Remove an order
        - Export all data
        - Quit
    - Get new order information
        - Order date
        - Customer name
        - Stature
        - Product type
        - area
    - Success/result banner
        - orderAdded
        - orderEdit
        - removeOrder
        - exportData
    - Display order(list <orders>)
    - Exit banner
    - Error banner
    - Unknown command banner
    - Prompt the user “are you sure”
        - Y/N
- FlooringMasteryServiceLayer:
    - Constructor()
    - Variables
        - Dao
        - Auditdao
    - validateOrderData
    - createOrder
    - getAllOrders
    - getOrder(date, orderId)
    - removeOrder(date, orderId)
    - exportData()

- FlooringMasteryController
    - Constructor()
    - Variables
        - view
        - service
    - run()
    - getMenuSelection()’
    - displayOrders()
    - addOrder()
    - editOrder()
    - removeOrder()
    - exportData()
    - exitMessage();
    - unknownCommand();
- App
    - Spring commands
- Exceptions
    - FlooringMasteryPersistenceException
    - NoOrderException
    - FlooringMasteryDataValidationException

3. Create a flowchart for the user interaction process



4. Create file format to match the identified domain object(s)
    a. Backup
        i. <OrderNumber>,<CustomerName>,<State>,<TaxRate>,<ProductType>, <Area>,<CostPerSquareFoot>,<LaborCostPerSquareFoot>, <MaterialCost>,<LaborCost>,<Tax>,<Total>,<OrderDate>
    b. Data: product
        i. <ProductType>,<CostPerSquareFoot>,<LaborCostPerSquareFoot>
    c. Data: taxes
        i. <State>,<StateName>,<TaxRate>
    d. Orders
        i. <OrderNumber>,<CustomerName>,<State>,<TaxRate>,<ProductType>, <Area>,<CostPerSquareFoot>,<LaborCostPerSquareFoot>, <MaterialCost>,<LaborCost>,<Tax>,<Total>

Construction Steps: See code

5. Create a menu system with stubbed-out code for each menu choice.
    a. Ex: User presses the choice to add an address ⇒ prints "AddAddress: To Be Implemented."
    b. In the execute method of AddressBookController
6. Pick a user story to implement. And implement it
    a. Ex:
    - Create the Address class (domain object)
    - Create the AddressBookDao class:
        - ArrayList to hold the Address objects as a class-level variable
        - Implement the addAddress(...) method
    - Add code into the AddressBookController to:
        - Instantiate AddressBookDao object for storing Address information
        - Read in address information from the user
        - Create a new Address object
        - Put address information from the user into the Address object
        - Add the new Address to the AddressBookDao
7. Repeat step 6 for all user stories