|  |  |  |
| --- | --- | --- |
| **LAB 211 Assignment** | **Type:** | **Long Assignment** |
| **Code:** | **01** |
| **LOC:** | **500** |
| **Slot(s):** | **N/A** |

**Title**

FlowerStore Managem

ent

**Background**

* Write a program to manage a flower store. The program implements the terminology of the Object-Oriented Programming (OOP) paradigm. OOP is one of the best choosing ways to design software programs. This program must support a basic security. Roles in the shop include:
* (1) DEV: Managing users
* (2) USER: Managing dealers

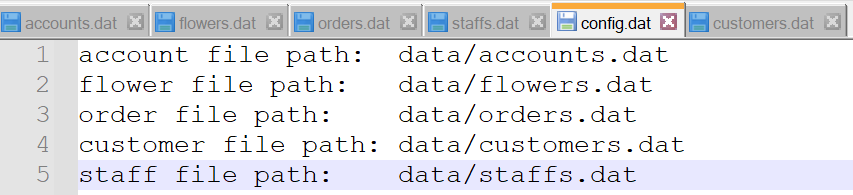


Figure: config.dat file, this file stores all file path of all dat file.

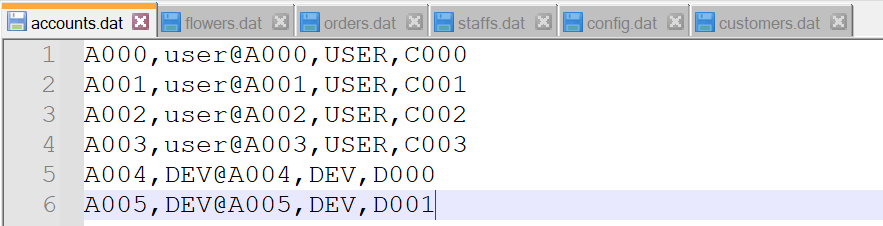


Figure: accounts.dat file including AccountID, Password, ROLE, and CustomerID or StaffId which associates with this AccountID. Each field is separated by a comma.

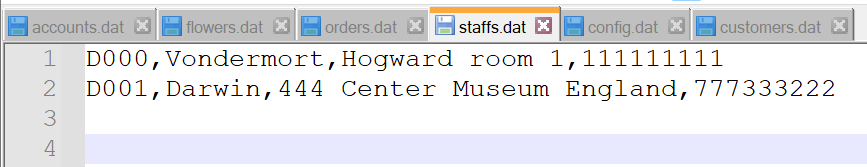


Figure: staff file, the file stores all staff of the shop. Each line includes StaffId, name,address,phone

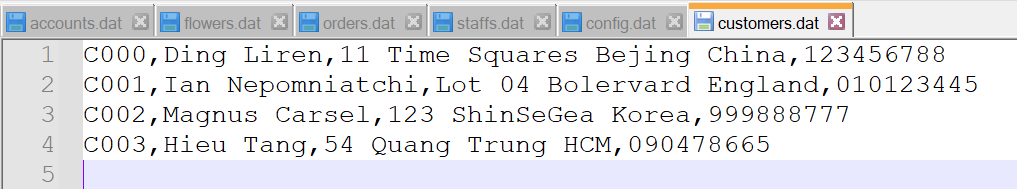


Figure: customer data file. Fields in each line are CustomerId, name, address, phone

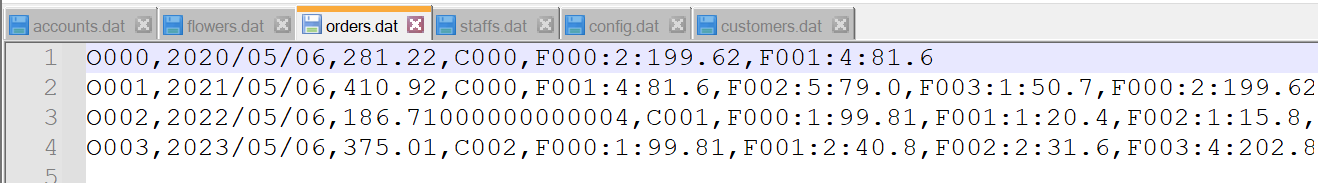


Figure: orders data file. The file contains all order from users such as OrderId, date, totalPrice, FlowerId1:num1:price1,FlowerId2:num2:price2,… The FlowerIdx relates to flower data file as each order can have many flower types.

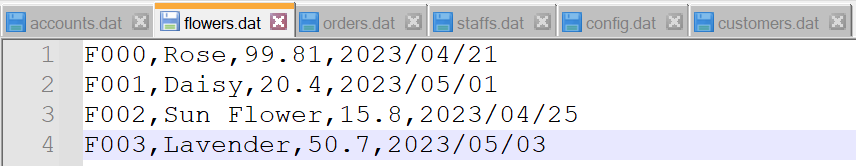


Figure: flowers.dat file, this file contains data for Flower including FlowerId, Name, unitPrice, importDate

* All users must be login to system to carry out appropriate management activities based on his/her role.
* Depend on users role, there will have menu with respect to them.

Views:Only display and get user input, send request to Controller ( resemble as front end)

**Logout**

Login

**Logout**

Role=USER

**Login**

**Login**

Role=DEV

DevMenu

CustomerMenu

Result,statusCode

Request,payload

Controllers: Process Request and return results and statusCode for views, similar to back end.

Connect with Models to get actual data and perform operations on them.

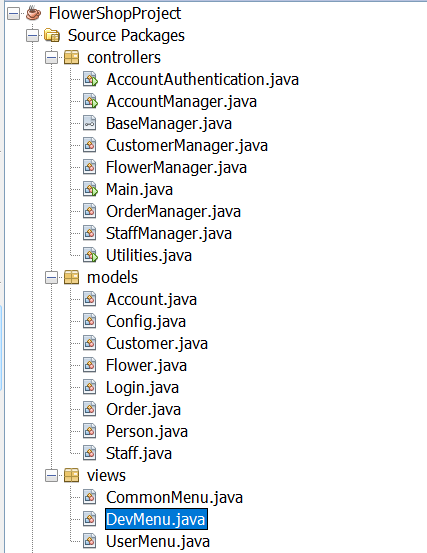
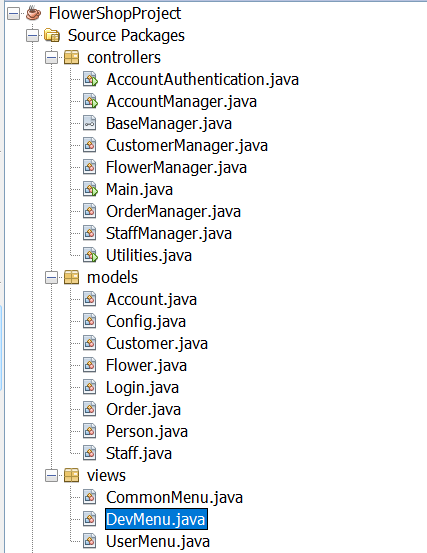
AccountManager

CustomerManger

StaffManager

OrderManager

FlowerManger



Models: Contain simple classes which mainly store data

Account

Customer

Flower

Order

Staff

In this assignment, we will use the Set<Flower> structure to store a collection of objects since it is a unique flower species.

Each menu choice should invoke an appropriate function to perform the selected menu item. Your program must display the menu after each task and wait for the user to select another option until the user chooses to quit the program.

**Program Specifications**

# Build the FlowerStore project with 2 menus as follows(50 LOC):

|  |  |
| --- | --- |
| UserMenu | DevMenu |
| If user enter account and password that matches with an account in accounts.dat file and role=USER, show this menu.  **Constrains:**   * Id of Account must be started with letter “A” followed by exactly 3 digits. * Password must contain at least a character, a digit, and a special character, the minimum length should be greater or equal 8. | If user enter account and password that match with an account in account.dat file and role=DEV, show this menu.  **Constrains:**   * Id of Account must be started with letter “A” followed by exactly 3 digits. * Password must contain at least a character, a digit, and a special character, the minimum length should be greater or equal 8. |
|  | |

# Update a Profile (50 LOC)

|  |
| --- |
| UserMenu && DEV MENU |
| If choose the option, update a customer or staff which associates with this account, later new information should be store in Customers.dat file with updated data.  Before update:    Figure. Current account is A000 which connects to customer C000    Figure. Before update of C000  After exit, the data will be saved into dat file:    Figure. After update, C000 changes with new information.  **Constrains:**   * Phone number must contain exactly 9 or 11 digits. |

# View Flower List (50 LOC)

|  |
| --- |
| UserMenu && DEV MENU |
| If user chooses the option, show data of all flower  **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... |

# View Flower to Cart (50 LOC)

|  |
| --- |
| USER MENU |
| In this option, you will add 3 times.  1st time: add F000 with quantity of 1  2nd time: add F001 with quantity of 1  3rd time: add F000 with quantity of 2  Then, the result in View Order looks like this:    Because F000 is the FlowerId of Rose 🡪 then we have 3. Similarly, F001 is the FlowerId of Daisy 🡪 then we have 1 in quantity.  **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... * Order ID is automatically calculated and must not conflict with existing one. * Date of order is current date in local time * You also need to calculate price of each flower as well as total price of the order. * BuyerId is the ID of customer who buy these flower. It can be searched using current Account |

# View Order (50 LOC)

|  |
| --- |
| USER MENU |
| * If users choose to view their order. Show this table with following information:   + Order ID: while account still logging into the system, whenever the user add Flower to Cart, there will be only 1 Order. The Order ID will start with letter “O” and 3 digits ( Refer to Orders.dat to see more). Therefore, if user add to Cart, automatically create an OrderId which is not conflict with existing ones.   + Date: Date when user create the Order   + Buyer ID: Customer ID who are using this account.   + Flower Name: Name of each flower in the order.   + Quantity: Number of flower of each type.   + Price: Price of each flower ( num\*unitPrice)   + Total: sum of all price.   **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... * Order ID is automatically calculated and must not conflict with existing one. * Date of order is current date in local time * You also need to calculate price of each flower as well as total price of the order. * BuyerId is the ID of customer who buy these flowers. It can be searched using current Account |

# View Order (100 LOC)

|  |
| --- |
| DEV MENU |
| * In this function, users are asked to input whether sort order by Flower Count(“count”), Order Date “date”, or Order Total (“price”). Sorting type is either ascending(“asc”) or descending(“desc”). * In addition, users can choose fromDate(yyyy/MM/dd) to toDate(yyyy/MM/dd) limit result in a timeframe. * More examples:         **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... * Other information must retrieve correctly from the system. |

# Add Flower(50 LOC)

|  |
| --- |
| DEV MENU |
| * When user choose this option, we will create new flower and add to the flowerList.     Figure. Before adding a new flower.    Figure. After adding a new flower.  **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... * FlowerId will be calculated and must not conflict with existing one. |
|  |

# Modify Flower(50 LOC)

|  |
| --- |
| DEV MENU |
| * When user choose this option, we will modify new flower by its id.       Figure. Before modifying flower.    Figure. After modifying flower.  **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... |

# remove Flower(50 LOC)

|  |
| --- |
| DEV MENU |
| * When user choose this option, we will remove the flower by its id.       Figure. Before removing flower.    Figure. After removing flower.  **Constrains:**   * You must format printing table as sample code, border with ‘#’ symbols, align right... |

**Hints:**

The above specifications are only basic information; you must perform a requirements analysis step and build the application according to real requirements.

The lecturer will explain the requirement only once in the first slot of the assignment.