**Project Description:**

Purchase Assistant program that allows a user to manage supplier and order details.

The program displays a menu with several options, such as entering a new supplier or order, viewing payment details for orders, and viewing a delivery report.

The **menu()** method is used to display the menu options to the user. It prints out the available options and then prompts the user to enter a number corresponding to their chosen option.

The **execute()** method is responsible for executing the user's chosen menu option. It first checks whether the user input is a valid menu option. If the input is valid, the method then executes the corresponding code block based on the user's choice.

Each menu option calls a method to execute a specific functionality. For example, if the user chooses to add a new supplier, the execute() method calls a menu() method to display the relevant options, takes user input for supplier name and contact, creates a new Supplier object, adds it to an ArrayList of suppliers, and writes the updated data to a file.

Similarly, if the user chooses to place a new order, the execute() method displays a list of available suppliers and takes user input for the chosen supplier. It then creates a new Order object, takes user input for the items, quantities, and prices in the order, calculates the total bill, takes user input for the expected delivery date, adds the order to the supplier's ArrayList of orders, and writes the updated data to a file.

If the user chooses to view payment details for orders, the execute() method iterates through all suppliers and their orders, prints out the supplier ID, order ID, and total bill for any undelivered orders.

**Instructions for Purchase Assistant Application:**

The Purchase Assistant Application is a tool to help you manage your purchases. Here are the instructions on how to use the application:

1. When you start the application, you will see a menu with the following options:
   1. Enter new supplier
   2. Enter new order
   3. View payment detail for orders
   4. View delivery report
   5. Exit
2. To select an option, type in the number next to the option and press enter.
3. If you choose option 1, you can enter a new supplier's details, such as their name and contact information.
4. If you choose option 2, you can place a new order by selecting a supplier and entering the order details, such as item information, quantity, and price.
5. If you choose option 3, you can view payment details for all orders that have not yet been delivered.
6. If you choose option 4, you can view a delivery report for all orders that have been delivered.
7. If you choose option 5, the application will exit.

Note: You can only choose options 1-4 if you have previously entered suppliers or orders. If you have not done so, you will need to choose option 1 or 2 first before the other options become available. Also, the application will automatically save your data to a file.

**Solving of problems:**

First there was a problem of choosing appropriate data structure for storing data locally. I solved it by using ArrayList:Collection . First, I created a general arraylist of suppliers. Then, every supplier has orders that have been placed alongwith its info. Inside order, I maintain data regarding price, dates, items contained and payments record. Hence, by achieving this hierarchy, I was able to maintain and access data smoothly.

I was facing a challenge of saving data in the form of objects, so I devised following logic in ‘FileHandling’ class. "FileHandling" which provides two methods for handling files: write() and read(). The methods use ObjectInputStream and ObjectOutputStream classes to read and write data from/to a file, respectively.

The write() method takes no arguments and returns void. It opens an ObjectOutputStream with the filename "data.txt" and writes the data stored in the ArrayList named "MainProg.suppliers" to the file. If an IOException is thrown, the stack trace is printed.

The read() method takes no arguments and returns void. It opens an ObjectInputStream with the filename "data.txt" and reads the ArrayList of Supplier objects stored in the file. It then assigns this ArrayList to the "MainProg.suppliers" variable. If an IOException or ClassNotFoundException is thrown, the stack trace is printed.

|  |  |  |
| --- | --- | --- |
|  | **Example of code used in assignment** | **Comment** |
| **JavaFX** | AnchorPane root = **new** AnchorPane();  root.setPrefHeight(690.0);  root.setPrefWidth(607.0);  Interface inF=**new** Interface();  inF.load(root);  Scene scene = **new** Scene(root);  primaryStage.setScene(scene);  primaryStage.show(); | This code creates an instance of the AnchorPane class, send it in order to populate it, then create a new scene and set it with pane above. |
| **Lambda**  **expressions** | // view suppliers  btnViewSuppliers.setOnAction(a -> {  displaySuppliers(txtAreaSample);  }); | This code uses lambda expressions to trigger a function on click event. |
| **Exceptions** | **try** (ObjectInputStream in = **new** ObjectInputStream(**new** FileInputStream("data.txt"))) {  ArrayList<Supplier> readList = (ArrayList<Supplier>) in.readObject();  MainProg.*suppliers* = readList;  } **catch** (IOException | ClassNotFoundException e) {  e.printStackTrace();  } | This code handles different types of exceptions. |
| **Collection classes** | ArrayList<Supplier> *suppliers*=**new** ArrayList<>(); | This code uses Collection : ArrayList |
| **File handling** | **public** **static** **void** write() {  **try** (ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream("data.txt"))) {  out.writeObject( MainProg.*suppliers* );  } **catch** (IOException e) {  e.printStackTrace();  }    }    **public** **static** **void** read() {  **try** (ObjectInputStream in = **new** ObjectInputStream(**new** FileInputStream("data.txt"))) {  ArrayList<Supplier> readList = (ArrayList<Supplier>) in.readObject();  MainProg.*suppliers* = readList;  } **catch** (IOException | ClassNotFoundException e) {  e.printStackTrace();  }    } | This code deals with file handling. It has two methods for reading & writing that handle object type arraylist of suppliers. |