Assignment - Design Pattern

Design Patterns used in the project:

1. Bridge pattern
2. Factory Pattern
3. Facade pattern
4. Iterator Pattern
5. Visitor Pattern

Classes, Interfaces, and their responsibilities:

1. Facade class - To create login for the user and implement the facade design pattern.
2. ProductIterator class - To implement the iterator design pattern and to display the menu as needed for the user using an iterator.
3. ProductMenu interface - To implement the bridge design pattern to affect the load menu option in the PTBS system. It displays the product list differently as per the user input.
4. Product class - To implement the factory design pattern which enables the subclasses to decide which class to instantiate.
5. ReminderVisitor class - To implement the visitor pattern which allows the user to go back to the previous step, by encapsulating the operation we want to perform multiple times.

Step-by-Step solution guide:

Prerequisites: Please add all the necessary SellerInfo, BuyerInfo, and ProductInfo text files as needed for the functioning of the code.

1. **Login**: The login is done using the facade class.
    - Takes username and password from the command line and matches if a valid buyer or seller exists in the SellerInfo and BuyerInfo files.
    - If such as user exists, it'll further create a new User class object and goes to print the products list as specified by the user.
    - Else, it exits the code stating - Invalid user

2. **Print Menu:** Bridge pattern came in handy here. The user has the below choices:

If it's a buyer:

- Type 1 to print the whole products list
- Type 2 to print only the meat products list
- Type 3 to print only the produce product list
- This loop continues unless the user types 0. I've achieved this using the Visitor design pattern by enclosing the function I need.

If it's a seller:

- Type 1 to print the whole products list
- Type 2 to print only the meat products list
- Type 3 to print only the produce product list
- Type 4 to add any items to the ProductInfo text file
- This loop continues unless the user types 0. I've achieved this using the Visitor design pattern by enclosing the function I need.

3. **Printing specific menu**: I've achieved this using the bridge pattern.
    - If the user is a buyer, the showmenu method in the facade created a Buyer class object which in turn invokes either MeatProductMenu or ProduceProductMenu depending on what the user wants to see.

4. **Trading**: I've created the trading in Trading class which has the below methods.
    - setTrade: which allows the seller to set a price for a product
    - makeTrade: which allows the buyer to bid a price for a product of his/her choice. If the bid price is greater than or equal to the price set by the seller, then the buyer wins the bid.