Ravyn: Recommendation Algorithm, Purchase History, ShopperList

```
class RecommendationAlgorithm:
   methods:
      function generateRecommendations(List<String> purchaseHistory):
       # recommendation logic here
      print("Generating recommendations...")
       for (String item : purchaseHistory)
          print("Consider buying: " + item)

   function displayShoppingBudget():
      #  adv functionality for pulling budget from database
      print("Shopping Budget: " + shoppingBudget)

   function displayPurchaseHistory()
      print("Purchase History: " + purchaseHistory)
```

—----------

```
Class PurchaseHistory:
    Methods:
      Function displayHistory():
         #  adv functionality for pulling history from database
         print(“Purchase History:” + pastItems)

      Function updateHistory(List<String> newItems):
         pastItems = newItems
```

—----------

```
Class ShopperList:
    Methods:
      Function displayList():
         #  adv functionality for pulling current shopping list from database
         print(“Grocery List:” + currentItems)

      Function updateList(List<String> newItems):
         currentItems = currentItems.append(newItems)
```

```
Amado:  Coupon History, window and gui
class CouponHistory:
   #Store the coupon usage records
   methods:
     initialize:
         coupon_list = EmptyList
     function record_coupon_usage( coupon_details):
         #Append coupon_details to coupon_list

      function viewhistcoupon_list (coupon_list );
         For each coupon in this.couponList:
              print("Print coupon details...")
—----------------------------------------------------------------------------------------------------
class Window:
   methods:
      function open(list):
         this.content = list
      function close():
         this.content = None
      function display():
         if this.content is not None:
            # Implementation to display the content in the window
            print("Displaying content:", this.content)
         else:
            print("No content to display.")
      function move():
         # Implementation to move the window
         # This could be related to navigation within the app interface
      function handleEvent(event):
         if event == "button_click":
            print("Button clicked. Performing action...")
         elif event == "list_interaction":
            print("Interacting with the list. Performing action...")
         else:
            print("Unhandled event.")
—----------------------------------------------------------------------------------------------------
class GUI:
   methods:
      function userAccount(user: User):
         # This method displays user account information
         print("User Account Information:")
         print(f"Username: {user.username}")
         print(f"Email: {user.email}")

      function couponControl():
         # Implementation for coupon control GUI
         # This method displays available coupons
         # It allows users to apply or remove coupons from their cart
         # It may also display coupon usage history or details
         # Implementation details can be added here based on the coupon control functionality
```

Sneha: Price data, Price tracking history, Price tracking reasoning

```
class Product:
    attributes:
        - productID
        - name
        - currentPrice
        - description
        - ...

    methods:
        function addPriceToHistory(newPrice):
            timestamp = getCurrentTimestamp()
            PriceHistory.add({'timestamp': timestamp, 'price': newPrice})

        function getCurrentPrice():
            return this.currentPrice
—----------------------------------------------------------------------------------------------------
class PriceTracker:
    attributes:
        - productID
        - PriceHistory  # List to store historical prices and timestamps

    methods:
        function addPriceToHistory(newPrice):
            timestamp = getCurrentTimestamp()
            this.PriceHistory.add({'timestamp': timestamp, 'price': newPrice})

        function checkHistory():
            return this.PriceHistory
—----------------------------------------------------------------------------------------------------
class PriceAnalyzer:
    attributes:
        - productID
        - PriceHistory
        - reasoning  # String to store reasoning for price changes

    methods:
        function analyzePriceChange():
            latestPrice = getLastPriceFromHistory()
            previousPrice = getSecondToLastPriceFromHistory()

            if latestPrice < previousPrice:
                this.reasoning = "Price decreased, possible sale or promotion."
            elif latestPrice > previousPrice:
                this.reasoning = "Price increased, potential inflation or high demand."
            else:
                this.reasoning = "Price remained unchanged."

        function getReasoning():
            return this.reasoning
```

Sid:SearchService

```
Class SearchService:
    Methods:
        Function findCheapestStoreToBuyAllItemsAt() -> String:
            cheapestStore = None
            # Logic to find the cheapest store to buy all items
            if len(shopperList) > 0 and len(stores) > 0:
                # Placeholder logic to find the cheapest store (you can replace this with actual logic)
                cheapestStore = stores[0]
                print("Cheapest store found:", cheapestStore)
            else:
                print("Shopper list or stores is empty. Cannot find the cheapest store.")
            return cheapestStore
```

```
Class GroceryStore:
    Methods:
        Function fetchPriceFromDatabase(item: String) -> float:
            # Simulated method to retrieve item price from an SQL database
            # Assume connection to an SQL database and fetching prices
            # Placeholder prices for demonstration

            # Example connection and query using SQL (pseudocode)
            connection = establishSQLConnection()
            query = "SELECT price FROM items WHERE item_name = " + item

            try:
                # Execute the SQL query
                result = executeSQLQuery(connection, query)

                # Assume fetching the price from the result
                if result.isNotEmpty():
                    # Access the price from the result set (pseudocode)
                    price = result.getPrice()  # Retrieve the price from the result set
                else:
                    print("Item not found in the database.")
                    price = 0.0  # Placeholder value if item is not found
            except SQLConnectionException as e:
                print("SQL Connection Error:", e)
                price = 0.0  # Placeholder value in case of connection error

            # Close the SQL connection
            closeSQLConnection(connection)

            return price
```

```
Class DietaryPreferencesFilter:
    Methods:
        Function selectPreference(preference: String):
            # Toggle the selection of a dietary preference
            if preference in selectedPreferences:
                selectedPreferences[preference] = not selectedPreferences[preference]
            else:
                selectedPreferences[preference] = True
```

```
Function applyFilters() -> List<Product>:
    # Apply selected dietary filters to the product list
    filteredProducts = queryDatabaseForFilteredProducts(selectedPreferences)
    return filteredProducts
```

Hiwot:

Shopping budget, budget history , budget

Shopping budget

Attributes:
Class shopping: shopping budget

Method:
- monthlyBudget: double
setBudget(budget: double): void
addItemToBudget(item: String):
checkBudget(): boolean

budget history
Attributes:
UserID: String
- Amount: Float
Date: Date

Method:
+ viewPastBudgets()
: List<Budget>
+ setBudget(): Float
+ getBudget(): Budget
+ getUserInfo(): user
+ amount() List<purchase>

Budget
Attributes:
- month: String
- exceeded: boolean

Class ShoppingBudget:

Methods:
setBudget(budget: double) -> void:

monthlyBudget = budget

addItemToBudget(item: String) -> void:

items.add(item)

checkBudget() -> boolean:

```
        totalSpent = sum of amount in budgetHistory
        return totalSpent <= monthlyBudget

Budget HIstory

    viewPastBudgets() -> void:
        for history in budgetHistory:
            print("Date: " + history.getDate() +
                ", Amount: " + history.getAmount() +
                ", Exceeded: " + history.getBudget().isExceeded())

    setBudget() -> double:


    getBudget() -> double:

        return monthlyBudget

    getUserInfo() -> List of BudgetHistory:

        return budgetHistory

  Class Budget:
    Attributes:
        month: String
        exceeded: boolean


    Constructor(userId: String, amount: double, date: Date, budget: Budget) -> BudgetHistory:
```