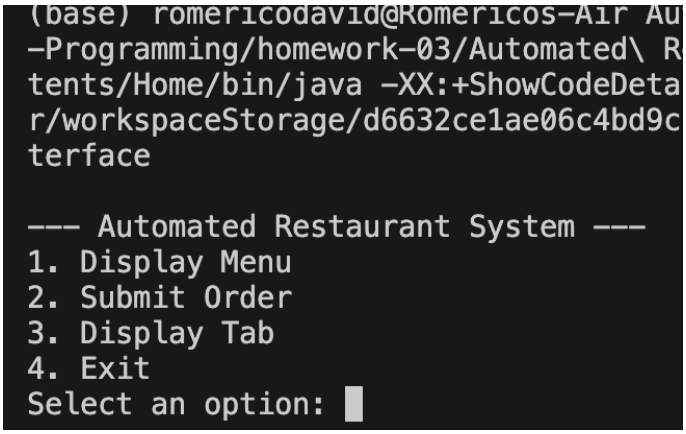
# Romerico David Jr.

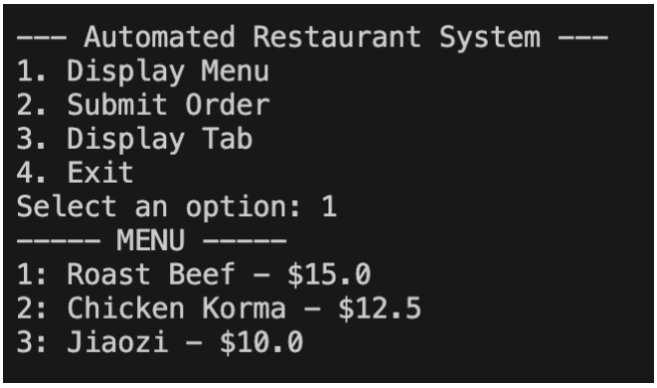
# Documentation

### Screenshots

1. Initial display




2. Display menu




3. Entering a valid order

```
----- MENU -----
1: Roast Beef - $15.0
2: Chicken Korma - $12.5
3: Jiaozi - $10.0

--- Automated Restaurant System ---
1. Display Menu
2. Submit Order
3. Display Tab
4. Exit
Select an option: 2
Enter menu item numbers to order, separated by spaces: 1
Order submitted successfully!
```

4. Displaying tab

```
--- Automated Restaurant System ---
1. Display Menu
2. Submit Order
3. Display Tab
4. Exit
Select an option: 3
----- TAB -----
Roast Beef - $15.5
Total: $15.5
```

5. Entering an invalid order and showing the error

```
--- Automated Restaurant System ---
1. Display Menu
2. Submit Order
3. Display Tab
4. Exit
Select an option: 2
Enter menu item numbers to order, separated by spaces: 0
Error submitting order: Item number 0 does not exist in the menu.
```

6. Entering multiple valid orders

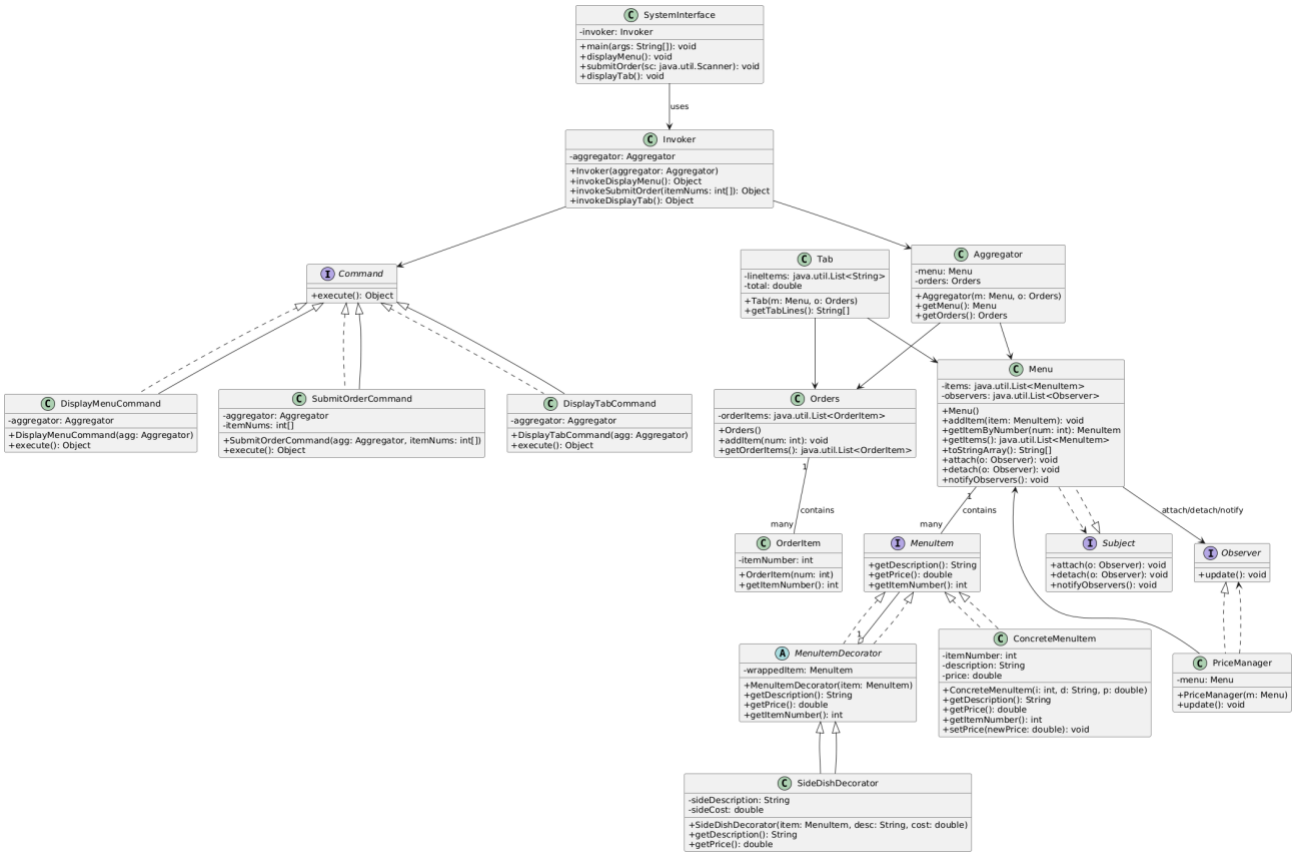
```
--- Automated Restaurant System ---
1. Display Menu
2. Submit Order
3. Display Tab
4. Exit
Select an option: 2
Enter menu item numbers to order, separated by spaces: 1 2 3
Order submitted successfully!

--- Automated Restaurant System ---
1. Display Menu
2. Submit Order
3. Display Tab
4. Exit
Select an option: 3
----- TAB -----
Roast Beef - $16.0
Roast Beef - $16.0
Chicken Korma - $12.5
Jiaozi - $10.0
Total: $54.5
```

7. Quitting program

```
--- Automated Restaurant System ---
1. Display Menu
2. Submit Order
3. Display Tab
4. Exit
Select an option: 4
Goodbye!
```

## Class Diagram




# Design Patterns

**Command Pattern**

- **Where Used**: The three main commands (Display Menu, Submit Order, Display Tab) are encapsulated as command objects.
- **Why**: This separates the user interface (SystemInterface) from the business logic that executes the commands making the system flexible and easy to extend.
- **Benefits**: This allows adding new commands without changing existing code significantly which promotes decoupling between UI and logic and adheres to the Open/Closed principle.

**Observer Pattern**

- **Where Used**: The Menu is a Subject and PriceManager acts as an Observer. When Menu changes (like after orders are submitted), it notifies PriceManager and it can then adjust pricing.
- **Why**: This lets us dynamically update parts of the system (like pricing, availability) when changes occur without tight coupling.
- **Benefits**: Flexible extension of behavior because Observers can be added or removed without changing the Subject. The Subject also doesn't know the details of how Observers respond.

**Decorator Pattern**

- **Where Used**: SideDishDecorator wraps a MenuItem to add additional features or costs.
- **Why**: Adds optional extras to a base menu item without altering the original class.
- **Benefits**: Complies with the Open/Closed principle because you can add new features (sides) at runtime and I can combine multiple decorators.

## Walkthrough of Design

1. **Initialization**:
   SystemInterface.main() sets up an Aggregator with a Menu and Orders. The menu is populated with a base ConcreteMenuItems. An PriceManager observer is attached to the Menu.
2. **Displaying the Menu**:
   The user selects "1: Display Menu." The SystemInterface.displayMenu() calls invoker.invokeDisplayMenu() which executes a DisplayMenuCommand and returns a string array of menu lines.
3. **Submitting an Order**:
   The user selects "2: Submit Order" and enters item numbers. The SystemInterface.submitOrder() retrieves these numbers and calls invoker.invokeSubmitOrder(itemNums). SubmitOrderCommand checks each item number, if any don't exist, it returns an error. If all are valid, the items are added to Orders and Menu notifies PriceManager which adjust prices if conditions are met.
4. **Displaying the Tab**:
   The user selects "3: Display Tab." The SystemInterface.displayTab() calls invoker.invokeDisplayTab(). The DisplayTabCommand uses Menu and Orders to construct a Tab and returns the tab lines. The UI prints them, showing each ordered item and the total cost.
5. **Dynamic Updates (Observer)**:
   After multiple orders, Menu calls notifyObservers(). PriceManager responds by adjusting the price of certain items. The next time the menu or tab is displayed, the updated prices are shown automatically.
6. **Enhanced Items (Decorator)**:
   At runtime, a ConcreteMenuItem is wrapped with a SideDishDecorator. The displayed menu and tab will reflect those descriptions and higher prices without modifying ConcreteMenuItem itself.