When our program is first started, Main will display the menu choices as follows as a case switch block:

1. Display all menu items
2. Submit an order
3. Display tab
4. Pay tab
5. Show ordering stats
6. Exit

Display all menu items (1): Main will call the public static function ‘displayMenu’ in SystemInterface which will call the ‘invokeDisplayMenu’ function of the Invoker class. This function will then use the CommandFactory class to create the ‘DisplayMenu’ command of the ‘CommandDisplayMenu’ class and then call the execute function of that command. CommandDisplayMenu implements the Command interface which includes two functions: execute and display. The execute function uses the locally declared Aggregator to get the menu output and then calls display. Display takes the menu output and prints it to the console.

Submit an order (2): Main will call the public static function ‘submitOrder’ in SystemInterface which will call the ‘invokeSubmitOrder’ function of the Invoker class. This function will then use the CommandFactory class to create the ‘SubmitOrder’ command of the ‘CommandSubmitOrder’ class and then call the execute function of that command. CommandSubmitOrder implements the Command interface which includes two functions: execute and display. The execute function uses the locally declared Aggregator to get the add the order to the OrderItem list and then calls display. Display takes the order and prints it to the console.

Display tab (3): Main will call the public static function ‘displayTab’ in SystemInterface which will call the ‘invokeDisplayTab’ function of the Invoker class. This function will then use the CommandFactory class to create the ‘DisplayTab’ command of the ‘CommandDisplayTab’ class and then call the execute function of that command. CommandDisplayTab implements the Command interface which includes two functions: execute and display. The execute function creates a new Tab object, gets the current tab, and then calls display. Display takes the tab output and prints it to the console.

Pay tab (4): Main will call the public static function ‘payTab’ in SystemInterface which will call the ‘invokePayTab’ function of the Invoker class. This function will then use the CommandFactory class to create either the ‘PayTabCash’ command of the ‘CommandPayTabCash’ class, or the ‘PayTabCard’ command of the ‘CommandPayTabCard’ class. Either way, it will then call the execute function of that command. CommandPayTab implements the Command interface which includes two functions: execute and display. The execute function uses the locally declared pay function of the PaymentStartegy object to calculate the total cost and clear the current order items. The execute then calls the display function to print payment confirmation to console.

Show ordering stats (5): Main will call the public static function ‘showStats’ in SystemInterface which will call the ‘invokeShowStats’ function of the Invoker class. This function will then use the CommandFactory class to create the ‘ShowStats’ command of the ‘CommandShowStats’ class and then call the execute function of that command. CommandShowStats implements the Command interface which includes two functions: execute and display. The execute function creates a new Tab object, gets the stats of all ordered items, and then calls display. Display takes the tab’s stat output and prints it to the console.

Exit (0): Main exits loop and terminates.

This program incorporates the singleton, command, and strategy design patterns. The singleton design pattern is incorporated with the Aggregator class. The class implements the ‘getInstance’ function which allows a global point of access to a single object. This is beneficial since the aggregator is used across many classes and every class relies on using the same object. This also avoids the need for passing a single aggregator object between classes. The command design pattern is used to implement all the menu choices. This is beneficial as it decouples the sender of a request from the receiver. And since the structure of the commands are all similar, a single command factory can easily organize the objects. Lastly, the strategy design pattern is used for the different ways to pay a user’s tab (with cash or card). The ‘PaymentStrategy’ class defines the family of similar payment algorithms. This is beneficial because it ensures that the classes implementing the strategy remain consistent and can be easily extended to include more options (i.e., payment with a giftcard).