In our project, we've implemented various design patterns to improve the organization and functionality of our codebase. Discussed below is our list of patterns and how we used for various processes within our system.

*Composite*

Our tables follow the Composite pattern with AbstractTable as the Component, Table as the Leaf and CombinedTable as the Composite. This functionality allows us to combine tables if groups of customers exceed the maximum number of people allowed at one table.

*State*

A few classes follow the State pattern to indicate when certain actions should be performed:

* TableState (State), with AbstractTable as its Context, has the following ConcreteStates that indicate which actions the customers at the Table are ready to take: NotOccupied, Waiting, NotReadyToOrder, ReadyToOrder and PayBill.
* CustomerState (State), with Customer as its Context, has the following ConcreteStates that are influenced by the service/food: Happy, Neutral, Angry.
* OrderStatus (State), with Order as its Context, has the following ConcreteStates that indicate the state of a customer’s order: Received, Processing, Ready.

*Iterator (Floor and Employee)*

To ensure a consistent program flow, without using concurrency, we use the Iterator design pattern to control the flow of activities by the Employees. Whether that’s chefs preparing food or waiters checking on orders in the kitchen or checking on customers. The employees will perform tasks based each iteration on it by the Floor.

* With AbstractTable as the Aggregate that we will iterate over and Table as a class that implements its interface (Concrete Aggregate)
* A generic Iterator interface and a specific iterator that implements the interface for the Table iteration process. (ConcreteIterator)

*Bridge*

The Department class follows the Bridge pattern by bridging the Employees to their respective departments. Department is the Implementor, with KitchenDepartment and FloorDepartment as the ConcreteImplementors, that create a bridge to Employee (Abstraction) with Chef, Waiter and Manager as RefinedAbstractions.

*Memento*

Memento will be used to keep record of all Bills for the restaurant’s bookkeeping. Bill (Originator) stores the order and calculates the total cost for the bill. BillMemento (Memento) then stores the state of the bill. All of these bills are then stored in BillCaretaker for future reference.

*Observer*

Observer is used to model how the Waiters will keep track of the Orders they take and notify the Kitchen about. KitchenNotifier is the Subject with OrderSubject as its ConcreteSubject. WaiterObserver is the Observer with Waiter as its ConcreteObserver.

*Chain of Responsibility*

The Chef class makes use of Chain of Responsibility to indicate the different Chefs available to work on Orders. Chef is the Handler while GrillChef, FryerChef and HeadChef are the ConcreteHandlers. This will pass a dish onto various chefs for each step in the cooking process necessary for each dish.

*Strategy*

Chef also makes use of Strategy as GrillChef, FryerChef and HeadChef (ConcreteStrategies) use similar algorithms inherited from the Chef (Strategy) class. The various implementations of the Chef class will produce specific implementations of the cook() function unique to their role within the cooking process.

*Template Method*

The Chef class can also be considered a Template as the skeleton of Chef is used to define its children classes. With Chef as the AbstractClass and GrillChef, FryerChef and HeadChef as the ConcreteClasses.

*Decorator*

We use the Decorator pattern to add variety to our dishes by allowing customers to choose different types of pizzas, burgers and add toppings to their pizzas. The Food class is our Component. This signifies a generic dish for which we will use decorators to make into speciality dishes like Pizza, Pasta, etc.

*Builder*

The Builder pattern is used to create dishes. AbstractDishBuilder is the Builder, dishBuilder is the ConcreteBuilder, Chef is the Director and Food will be our Product.

*Facade (Engine)*

We will use the Façade pattern as our central engine for tying all sections of the our code, the main sections being the Employees, Customers, Kitchen and Floor into a single class to reduce the complexities of using all of our many projected classes at once.