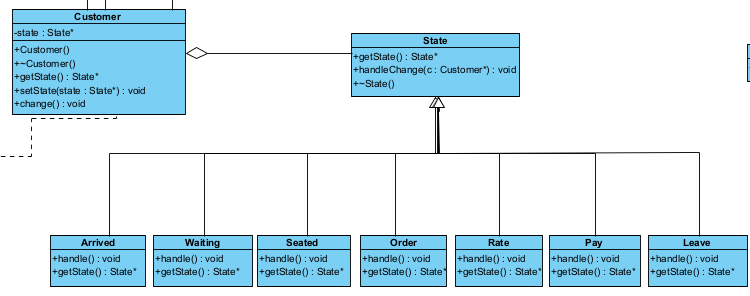
**COS 214 Task 4 Report**

**Task 4.2 – Design Decisions with reasoning**

1. Object-Oriented Design: The system seems to be designed using an object-oriented approach. This is evident from the different entities identified such as Customers, Waiters, Tables, and Kitchen which can be represented as classes in the system.
2. Customizability: The system allows for a high degree of customizability. This is seen in the ability for customers to customize their orders and request special cooking instructions. This design choice enhances the user experience and makes the simulation more realistic.
3. Modularity: The system is divided into distinct modules such as Customer Management, Waiter Management, Table Management, Kitchen Management, and Inventory Management. This design choice allows for better organization of code and easier maintenance and updates.
4. Communication between modules: There’s a clear communication protocol between different parts of the system. For example, waiters pass orders from customers to the kitchen and then serve the order when it’s ready. This design choice ensures smooth operation of the restaurant.
5. Error Handling and Reporting: The system is designed to handle unexpected scenarios and provide error messages. This design choice improves the robustness of the system and helps in troubleshooting issues.
6. Inventory Management: The system keeps track of ingredients used for dishes and updates the inventory accordingly. This design choice is crucial for avoiding situations where a dish cannot be prepared due to lack of ingredients.
7. Customer Satisfactions: The system allows customers to rate their experience at the restaurant. This design choice provides valuable feedback that can be used to improve the service.
8. Testing and Debugging:
   1. Unit Testing: The decision to have each person do unit testing on their own code ensures that individual components of the system work as expected before they are integrated. This can help catch and fix bugs early in the development process.
   2. Testing Framework: Creating a testing framework helps ensure the correctness of the code. It provides a structured way to write and run tests, making it easier to verify that the system behaves as expected under different conditions.
   3. Code Review Process: Having an admin approve merges on GitHub rather than allowing all users to automatically merge enforces a code review process. This can help catch potential issues and improve code quality.
9. Documentation:
   1. System Documentation: Documenting the system, its components, assumptions, and method of utilization helps users understand how to use the system and developers understand how to maintain or extend it.
   2. Diagram Tracking: Keeping track of all diagrams (like UML Class, Activity etc.) that helped simplify complex coding scenarios provides a visual representation of the system. This can make it easier to understand the system’s structure and behaviour.

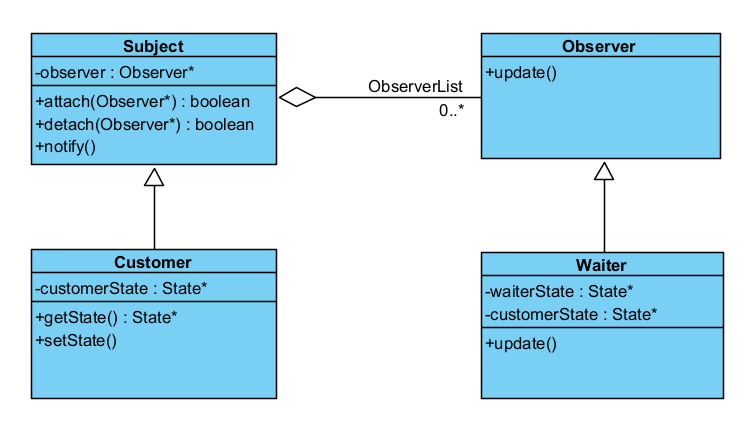
**Task 4.3 - A writeup showing how all the patterns have been used, also note what problem the pattern solved in your implementation**

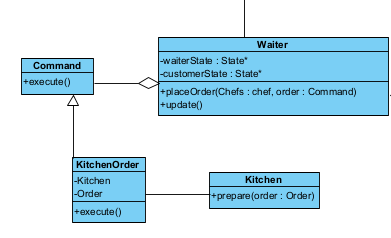
**The State design Pattern**



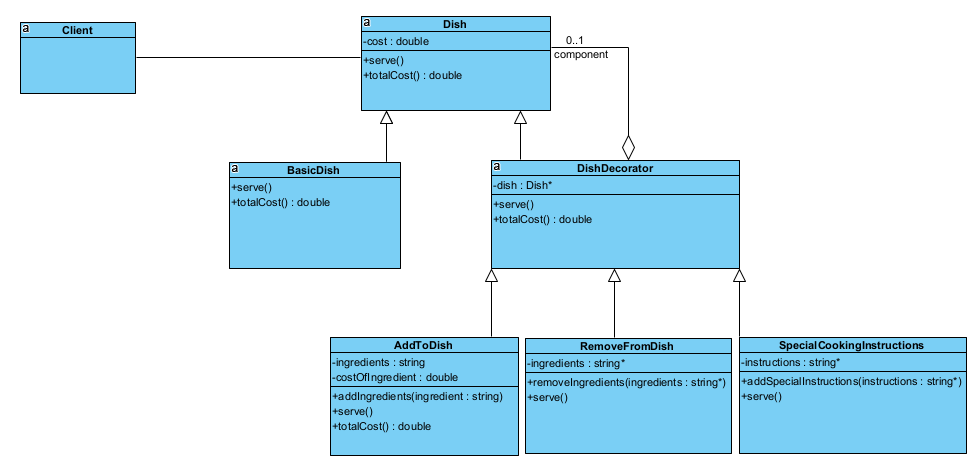
The state pattern allows us to alter the behaviour of certain functions depending on how the customer is feeling at a certain moment in time. This easily solves the issue of modelling a customer’s behaviour and is easily extendible meaning more states can be added if necessary.

**Observer**



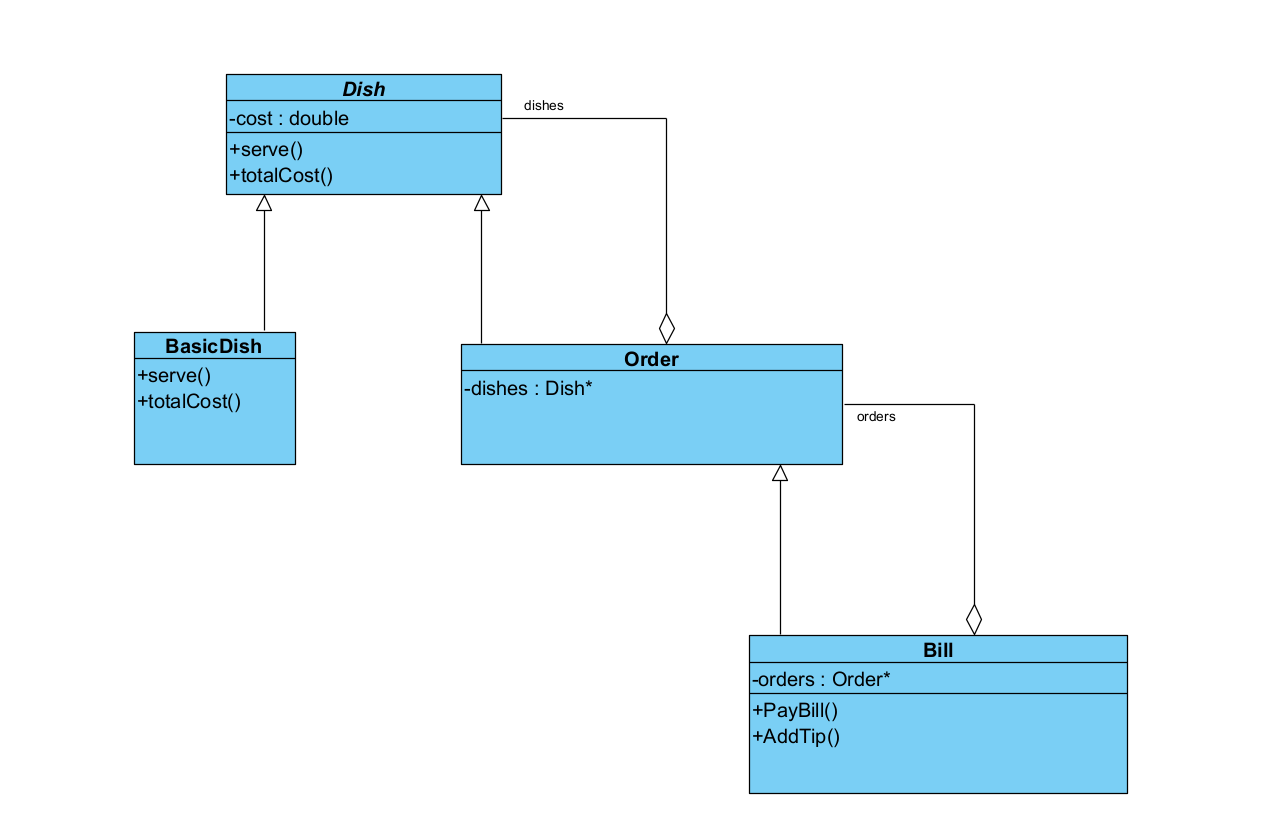
**Command**

**Decorator**



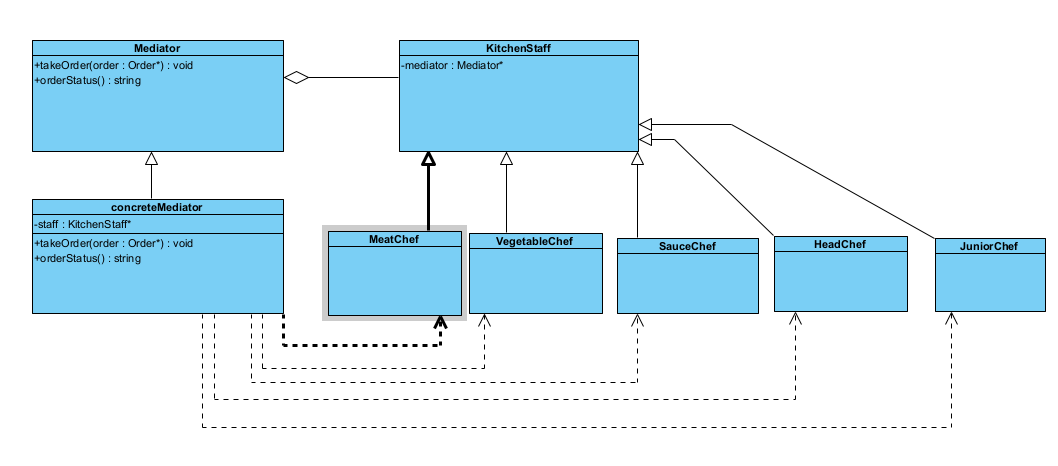
**Memento**

**Composite**

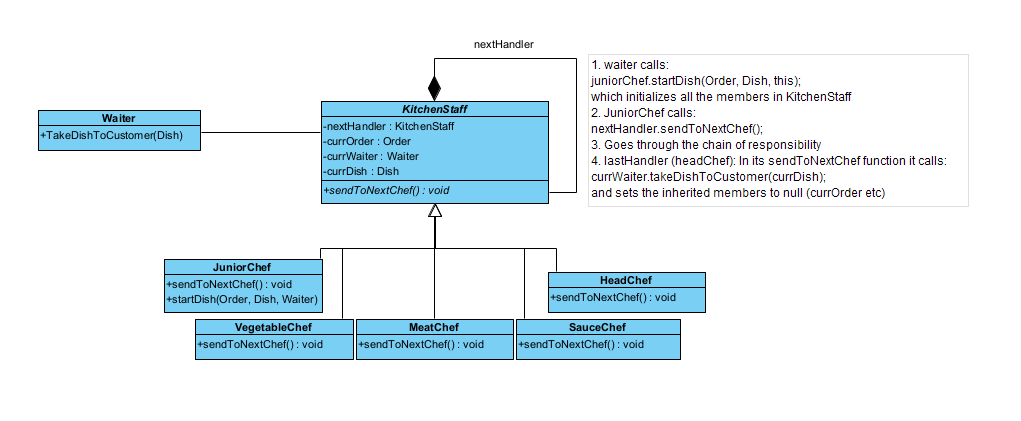


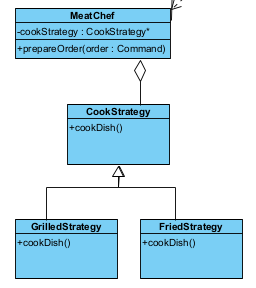
**Prototype**

**Mediator**



**Chain Of Responsibility**



**Strategy**

**Singleton**

