TASK 4: REPORT

Matthew Els u21715191

Dominique da Silva u21629944

Tlhalefo Dikolomela u21507792

Tinashe Austin u21564176

Christopher Katranas u19154853

Contents

[INTRODUCTION 2](#_Toc149737231)

[4.1 RESEARCH 2](#_Toc149737232)

[4.2 DESIGN DECISIONS 4](#_Toc149737233)

[System Requirements 4](#_Toc149737234)

[Customer Management. 4](#_Toc149737235)

[Order Management. 4](#_Toc149737236)

[Billing 4](#_Toc149737237)

[Satisfaction 4](#_Toc149737238)

[Order 4](#_Toc149737239)

[4.3 WRITE UP OF DESIGN PATTERNS 5](#_Toc149737240)

[4.3.1 BUILDER 5](#_Toc149737241)

[4.3.2 CHAIN OF RESPONSIBILITY 5](#_Toc149737242)

[4.3.3 STATE 5](#_Toc149737243)

[4.3.4 STRATEGY 6](#_Toc149737244)

[4.3.5 DECORATOR 6](#_Toc149737245)

[4.3.6 COMMAND 6](#_Toc149737246)

[4.3.7 TEMPLATE METHOD 6](#_Toc149737247)

[4.3.8 7](#_Toc149737248)

[4.3.9 7](#_Toc149737249)

[4.3.10 7](#_Toc149737250)

[4.4 ASSUMPTIONS 7](#_Toc149737251)

[4.5 SUPPORTING UML DIAGRAMS 8](#_Toc149737252)

# INTRODUCTION

Our (Concrete pass) implementation of the given assignment focuses on the implementation of a restaurant simulation system. Customers are seated at a table, orders are taken by the waiter, and the food is prepared and served.

The primary components we have focused on, in our system, is the floor where customers interact with the waiters and are managed by the staff, as well as the kitchen where food orders are prepared and dispatched.  
Customer satisfaction also plays a role in this simulation, influencing factors such as tipping of their assigned waiter.

Customers also can create customized orders from a menu and may choose to start tabs for deferred payment.

# 4.1 RESEARCH

Similar project were found online and was used as reference as to what patterns to consider in our system design (as referenced below).

GitHub. (n.d.). *Grokking-OOD/object-oriented-design-case-studies/design-a-restaurant-management-system.md at master · wyaadarsh/Grokking-OOD*. [online] Available at: https://github.com/wyaadarsh/Grokking-OOD/blob/master/object-oriented-design-case-studies/design-a-restaurant-management-system.md#class-diagram [Accessed 1 Nov. 2023].

‌

Shahid, M. (2023). *Restaurant-Management-System*. [online] GitHub. Available at: https://github.com/mabbia706/Restaurant-Management-System/tree/master [Accessed 1 Nov. 2023].

‌

www.c-sharpcorner.com. (n.d.). *Food Delivery Application Using Design Patterns*. [online] Available at: https://www.c-sharpcorner.com/article/food-delivery-application-using-with-design-patterns/.

‌

Doyle, W. (2023). *Design Patterns Explained with Food 🥕*. [online] GitHub. Available at: https://github.com/wesdoyle/design-patterns-explained-with-food/tree/main [Accessed 1 Nov. 2023].

‌

Kothari, A. (2018). *Java Builder Design Pattern Example - Java Code Geeks*. [online] Examples Java Code Geeks. Available at: https://examples.javacodegeeks.com/java-development/core-java/java-builder-design-pattern-example/ [Accessed 1 Nov. 2023].

‌

# 4.2 DESIGN DECISIONS

## System Requirements

### Customer Management.

The system had to have a mechanism for the customers to be able to request, whether that is by reservation or by walking in, a seat. It should mimic the role of a Maître D and assign customers to an available table. Wait staff must be assigned to the table as to manage and serve the customers that are seated at the table.

### Order Management.

Waiters should be able to take a table’s order, then communicate to the kitchen. The waiter should also be the one to serve to the table with the meal. Waiters have the role of mediator between the kitchen and the floor.

### Billing

The implemented system, should at the end of the dining experience be able to generate a bill for the table. The table should be able to request for bill splitting, i.e. should be able to pay the bill in multiple payments. In addition to that, the table should be able to defer the payment, offering to pay the bill at a later date.

### Satisfaction

Customers should be able to rate their dining experience or complain. Should the experience be rated badly, there will be implications for the tipping of the waiter. Complaints should be handled by the manager.

### Order

Customer should be able to build their own order, selecting items from a set menu. Customers should also be able to specify any additions and preparation methods for their requested food. The order is taken by the waiter and delivered to the kitchen. The kitchen will notify the waiter when the food is done and thereafter the waiter will serve the table their order.

# 4.3 WRITE UP OF DESIGN PATTERNS

## 4.3.1 BUILDER

Intent:

The Builder design pattern's basic intent is to separate the construction of a complex object from its representation, allowing the same construction process to create different representations.

Application:

A pizzeria has many different types of pizzas (representations), however the basic construction remains the same up until toppings are to be chosen for the final product.  
Thus we have chosen the builder design pattern to be able to create many different types of pizzas while simplifying the construction process.

## 4.3.2 CHAIN OF RESPONSIBILITY

Intent:

The Chain of Responsibility design pattern is intended to create a chain of objects, where each object can process a request and decide whether to pass it to the next object in the chain or to stop processing it. It allows you to decouple the sender of a request from its receiver, providing multiple objects the opportunity to handle the request.

Application:

With this in mind, the decision was made to pass the order made by the table through a chain. Multiple classes have to handle the request in different ways.  
In our design the order made (the request made by the customer table) is passed from the customer table, to the waiter. The waiter then passes it to the kitchen, and eventually the order is passed to the head chef to plate, and then sent back to the waiter to serve the table with their meal.

## 4.3.3 STATE

Intent:

The basic intent of the State design pattern is to allow an object to alter its behavior when it’s internal state changes. The object appears to change its class, enabling it to adapt to different situations, and effectively encapsulates the state-specific behavior into separate state objects. This pattern helps achieve cleaner and more maintainable code by reducing conditional statements and promoting flexibility in handling state transitions.

Application:

## 4.3.4 STRATEGY

Intent:

The basic intent of the Strategy design pattern is to define a family of interchangeable algorithms, encapsulate each one, and make them interchangeable. This pattern allows a client to choose an algorithm from a family of algorithms at runtime, without altering the code that uses the algorithm. It promotes flexibility and ensures that the algorithm can vary independently from clients that use it.

Application:

## 4.3.5 DECORATOR

Intent:

The Decorator design pattern is to attach additional responsibilities to an object dynamically, providing a flexible and reusable way to extend its functionality. This pattern allows objects to be extended with new behaviors or features without modifying their core structure.

Application:

## 4.3.6 COMMAND

Intent:

The intent of the Command design pattern is to encapsulate a request as an object, thereby allowing for parameterization of clients with requests, queuing of requests, and providing support for undoable operations. This pattern separates the sender of a request from the receiver, enabling decoupling between the two, and allowing for the easy addition of new commands. It promotes flexibility and extensibility by turning requests into standalone objects that can be manipulated and executed in a more controlled and abstract way.

Application:

## 4.3.7 TEMPLATE METHOD

Intent:

Define the skeleton of an algorithm in a method, allowing certain steps of the algorithm to be implemented by subclasses. This pattern enables the reusability of the algorithm's structure while allowing variations in the implementation of specific steps. It promotes a consistent algorithm structure across multiple subclasses, ensuring that certain steps are defined by the template method, while other steps can be customized by individual subclasses.

Application:

## 4.3.8

Intent:

Application:

## 4.3.9

Intent:

Application:

## 4.3.10

Intent:

Application:

# 4.4 ASSUMPTIONS

# 4.5 SUPPORTING UML DIAGRAMS

(The idea is to include all the design patterns under this heading and refer to the figures in 4.3)

(We can just instant reverse our code and use a snipping tool to get each pattern’s UML class diagram)

Possibly also include a link to the entire system design UML Class diagram.