TASK 4: REPORT

Matthew Els u21715191

Dominique da Silva u21629944

Tlhalefo Dikolomela u21507792

Tinashe Austin u21564176

Christopher Katranas u19154853

Contents

[INTRODUCTION 2](#_Toc149895873)

[4.1 RESEARCH 2](#_Toc149895874)

[4.2 DESIGN DECISIONS 4](#_Toc149895875)

[System Requirements (PROJECT SCENARIO) 4](#_Toc149895876)

[Customer Management. 4](#_Toc149895877)

[Order Management. 4](#_Toc149895878)

[Billing 4](#_Toc149895879)

[Satisfaction 4](#_Toc149895880)

[Order 5](#_Toc149895881)

[4.3 WRITE UP OF DESIGN PATTERNS 6](#_Toc149895882)

[4.3.1 BUILDER 6](#_Toc149895883)

[4.3.2 CHAIN OF RESPONSIBILITY 6](#_Toc149895884)

[4.3.3 STATE 6](#_Toc149895885)

[4.3.4 STRATEGY 7](#_Toc149895886)

[4.3.5 DECORATOR 7](#_Toc149895887)

[4.3.6 COMMAND 7](#_Toc149895888)

[4.3.7 TEMPLATE METHOD 8](#_Toc149895889)

[4.3.8 oBSERVER 8](#_Toc149895890)

[4.3.9 8](#_Toc149895891)

[4.3.10 COMPOSITE 8](#_Toc149895892)

[4.4 ASSUMPTIONS 9](#_Toc149895893)

[4.5 SUPPORTING UML DIAGRAMS 10](#_Toc149895894)

# 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). In addition, it did serve as a guideline for us to rather pursue a simulation of the environment, rather than a game-like implementation.

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].

GitHub. (n.d.). *COS214-Project/Docs/DefinitiveStyleGuide.md at main · Multimedia-Overachievers/COS214-Project*. [online] Available at: https://github.com/Multimedia-Overachievers/COS214-Project/blob/main/Docs/DefinitiveStyleGuide.md [Accessed 3 Nov. 2023].

262588213843476 (n.d.). *CodingGuidelines.md*. [online] Gist. Available at: https://gist.github.com/earonesty/ccee25a56be7adeb5f670cf44e5fa479 [Accessed 3 Nov. 2023].

Software Engineering Stack Exchange. (n.d.). *Creating a coding standards document*. [online] Available at: https://softwareengineering.stackexchange.com/questions/196706/creating-a-coding-standards-document [Accessed 3 Nov. 2023].

webmaster (2016). *Embedded C Coding Standard*. [online] Barr Group. Available at: https://barrgroup.com/Embedded-Systems/Books/Embedded-C-Coding-Standard.

‌

‌

‌

‌

# 4.2 DESIGN DECISIONS

The idea for our implementation was to implement a simulation of an actual restaurant, rather than the restaurant tycoon.

This was purely because it was easier to conceptualize, as we have experienced the mundane workflow of a restaurant, compared to a game.

We wanted the system to display the experience of a customer coming to a restaurant, having made a reservation or just walking in. Customers might have a party and tables might have to be joined for the group and now become a single table that needs to be serves, and later the table would have to be split. In addition to that, there are only a finite number of tables available on the restaurant’s floor space. Thereafter mimic the behavior of a real waiter that comes and takes the order.

What happens to the order made once the waiter commands the kitchen to start the preparation phase? The focus was also to implement a theoretical idea of how the food would be assembled and handled in the kitchen environment. The process of being able to build a standard meal, and adding additional specializations to the food, the head chef plating the food with any type of garnishes. Lastly, the kitchen notifying the waiter to serve the customer.

Finally, the customer has to pay and indicate their satisfaction to determine what amount to tip the waiter. This implies getting the bill, and the table to split the bill evenly. If the table is particularly special, they might defer the payment and start a tab. Once the bill has been settled, the process is done and to be started when the next batch of customers arrive and are seated at a table.

## System Requirements (PROJECT SCENARIO)

### 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.
* Participants:
  + Director –
  + Builder –
  + ConcreteBuilder –
  + Product –
* 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.
* Participants:
  + Handler –
  + ConcreteHandlers –
* 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.
* Participants:
  + Context – MaitreD
  + State – TableState
  + ConcreteState – Table
* Application:
  + The state is used in relation to the management of the tables. The behavior of a table object changes based on the internal state, whether the table is available or not.

## 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.
* Participants:
  + Context –
  + Strategy –
  + ConcreteStrategy –
* Application:
  + This pattern is used to interchange the algorithm based on whether a customer is a walk in, or has reserved a table. The MaitreD assigns a table to a customer based on these 2 options.

## 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.
* Participants:
  + Component –
  + ConcreteComponent –
  + Decorator –
  + ConcreteDecorator -
* 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.
* Participants:
  + Invoker –
  + Receiver –
  + Command – Command
  + ConcreteCommand –
* 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.
* Participants:
  + AbstractClass –
  + ConcreteClass –
  + templateMethod() –
  + primitiveOperation() –
* Application:

## 4.3.8 oBSERVER

* Intent:
  + The Observer design pattern's intent is to define a one-to-many dependency between objects so that when one object changes its state, all its dependents are notified and updated automatically.
* Participants:
  + Subject –
  + ConcreteSubject –
  + Observer –
  + ConcreteObserver –
* Application:

## 4.3.9

* Intent:
* Participants:
* Application:

## 4.3.10 COMPOSITE

* Intent:
  + The Composite pattern composes objects into tree structures to represent part-whole hierarchies. Individual and compositions of objects can be treated uniformly.
* Participants:
  + Component – TableState
  + Leaf – Table
  + Composite – TableComposite
* Application:
  + The tree like structure of this pattern was used in order to keep track of compositions of tables (joining of the tables). Should a large number of customers enter the restaurant, for a group function for example, they cannot be seated at an individual table. The floor has a set number of tables. The availability of tables need to be checked and the system needs to calculate how many tables ought to be merge in order to facilitate the large group. Thus, the leafs consists of the single 4 seat tables, indicating that they form part of a node called the TableComposite.

# 4.4 ASSUMPTIONS

* We assume that individual customers do not matter to the system. A collection of customers of the restaurant being seated at a table, will be treated as a singular object. The table will be served by the waiter and not individual customers at the table,
* When splitting a Table’s bill, we assume that the value of the bill will be split evenly between the numbers of customers seated at that particular table.

# 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.