17/10/23

COS 214 PROJECT
RESTAURANT SIMULATOR

OFENTSE RAMOTHIBE

BOLOKA KGOPODITHATE

NTANDO MAZIBUKO

MATHENDA MAPHASHA

REALEBOGA MOATSHE

THASHLENE MOODLEY

TABLE OF CONTENTS

INTRODUCTION

This project entails a generic Restaurant Simulator which is designed to provide a simple demonstration of important subsystems that need to exist for the creation of any restaurant and how the subsystems interact with one another to bring about an operational restaurant that caters to the needs of customers in relation to what they can offer the customers.

The simulator highlights important aspects such as decisions that need to be made prior to opening a restaurant such as the maximum number of people it can cater for, the type of food to be sold to customers, and services that can be provided to customers to ensure a successful restaurant. It also highlights the important roles that each person plays, which include but not limited to:- the customer, maître D, waiter, chefs, and the manager.

DESIGN

Design decisions

Functional Requirements

The floor:

- Have and maintain a max capacity.
- Allow the construction and destruction of tables.
- Maintain a list of “customer bills”.
- Allow users to navigate to available tables.
- Allow the maître D to allocate customers to tables.
- Let users merge and split tables if necessary.
- Allow users to view the full floor.
- Make the decision to Reserve a table if booking is made.

Waiters:

Maître D

- Make decisions for merging and splitting tables based on party numbers.
- Make the decision on allocating Reserved tables.
- Make the decision to split tables and link with the corresponding waiter.
- Allocate tables to customers minimizing waste (for party of 4 if table 1(10 seats) and table 2(6 seats) allocate table 2.
- Maintain a queue of parties waiting to be seated. Refer to the floor to validate reservations.
- Allocate a waiter to a table.
- Make the decision to stop allocating tables based on max capacity.
- Maintain a list of waiters.

General waiter

- Indicate the section of floor or table they are assigned.
- Maintain a list of tables with outstanding orders/haven’t ordered yet.
- Maintain a timer for each table to simulate returning “after a while” if indicated by the table.
- Maintain tables bills (tab bill, full bill, split bill)

- Decide on the customer's bill if the bill's expiration date has been reached or has passed.
- Have a concrete time to decide whether to traverse their entire section or individual timers for individual tables.
- Traverse the floor.

The Engine:

- Will be in charge of communicating updates to all timers by notifying classes of changes.

Floor manager:

- Maintain a list of currently active Waiters.
- Maintain a list of currently active Tables.
- Occasionally traverse the floor.

Customer:

- Make decisions of tip based on their current mood.
- Create orders with custom decorators.
- Make decisions on creating and removing “bills” (if paid).
- Notify the table when they are ready to make an order.
- Can set a custom tip.
- Make the decision to pay a bill or split the bill into sub bills (allow the customer to either select the % split or custom split amounts).
- Make a complaint if mood becomes “unsatisfied” / ”extremely unsatisfied”.
- Create mood conditions based on time taken for order/order accuracy/time taken for waiter to return/random mood change.
- Notify the table when they are ready to leave.
- Make the decision to leave if they have not been allocated a table within a time.
- randomly set readyToOrder to true, with bias when waiter comes to check order.

Table:

- allow only tables of 1, tables of 2 and tables of 4.
- Be allocated a waiter.
- Make the decision to notify the waiter that all customers are ready to order/set timer for waiter to return to check if customers are ready to order.
- Be allocated a floor section/sub-section.
- Merge into n tables if notified.
- Split into n tables if notified.
- Maintain a list of outstanding orders and completed orders.
- Maintain a list of currently connected tables in the 4 cardinal directions.
- Change state to “empty” when customers vote to leave.

Kitchen:

- Maintain a list of currently active chefs.
- Maintain a list of chef responsibilities, each to be implemented or passed as part of the chain.

- Allow the interaction of a waiter and head chef over an order or a single dish.
- Allow chefs, managers, and waiters to do rounds.

Head Chef:

- Be the beginning and end of the dish chain.
- Make the decision as to when each of the other stations will receive the dish.
- Notify the waiter when a dish is complete.
- Occasionally traverse the floor.

Commis Chef:

- Determines if the dish they have been passed requires decoration.
- Pass the dish to the next chef as indicated by the dish.
- Pass the dish back to the head chef when/if complete.
- Can adjust preparation based on preferences.

Dish:

- Have cooking decorators.
- Maintain a reference to the customer that ordered it(create Dish “state” after cooking, undercooked, perfect, overcooked may affect customer mood differently, implemented randomly to allow variation with bias)
- allow attribute “time taken” to allow variation of customer mood if too much time is taken.

Menu:

- Allow customers to decorate a dish using side dishes, garnish and toppings.
- Allow customers to remove from dishes based on allergens/preferences.
- Manage the creation/destruction of dishes.
- Assign unique preparation times based on item type (starters have shorter times etc.)

Bill:

- Have the ability to adjust cost based on the presence of a tab.
- Allow more than one way to pay (card, cash) (may require chain to determine change to return)
- Make the decision to split into multiple bills when prompted.
- be dynamically allocated based on the orders, decorations and omitted items.
- account for customer tips given based on satisfaction.

Design patterns

The requirement is to accurately model how a customer can make an order, have it prepared and sent back, including the services available to the customer such as laying a complaint against bad service.

Composite

How the pattern was used:

- Manages tables, allowing them to be merged when there isn't enough space for a party's size.
    Merging tables is a common practice in restaurants, enabling customers to sit together comfortably.

Problem the pattern solved:

- The pattern allows you to treat individual tables and groups of merged tables uniformly, simplifying the code that deals with tables.

Memento

How the pattern was used:

- When a customer creates a tab, they can pay for a bill at a later date.

Problem the pattern solved:

- The restaurant can keep record of customers tabs and retrieve them when the customer is ready to pay or when the tab is close to the expiration date.

Strategy

How the pattern was used:

- The customer has the option to pay either by card or cash.

Problem the pattern solved:

- The customer would be able to choose which payment method they prefer to use to pay for their order.

Iterator

How the pattern was used:

- Provides seamless iteration through the tables on the floor. It allows other functions to access and iterate through the tables without needing to understand how the tables are connected.

Problem the pattern solved:

- It allows other functions to access and iterate through the tables without needing to understand how the tables are connected. This design pattern simplifies the interaction with

tables and enhances code modularity, making it easier to add or change functionalities related to tables.

Observer

How the pattern was used:

- The customer notifies the table when they are ready to order. The table then notifies the waiter to collect the order from the customer and send it to the kitchen.
- The head chef notifies the waiter when the order is ready for collection.

Problem the pattern solved:

- This ensures that the waiter knows the state of the customer without constantly checking the customer but waiting to be notified by the table.
- The waiter does not have to keep checking for the customer's order but gets notified when the order is ready for collection.

State

How the pattern was used:

- The customer can choose to cancel their order, on the condition that the order has not been prepared and is still on the queue.
- The pattern allowed the MaitreD to allocate tables to customers according to the status of the table. If the table was occupied, or reserved and the customer did not make the reservations, the MaitreD would then allocate any available table to the customer if the floor has not reached maximum capacity.

Problem the pattern solved:

- We used the pattern to maintain the state of the order to decide whether the customer can cancel the order without paying or not.
- Since the tables have states, the MaitreD would be able to know which tables to allocate to which customer. If a customer reserved a table, they would be allocated a table that has its state set to "reserved", if we have a new customer that has not made reservations, the MaitreD would allocate a table set to "available". This also allows us to know when we have reached full capacity and we cannot cater for more customers.

Chain of responsibility

How the pattern was used:

- The customer's order ideally comprises of a beverage, starter, main dish, and dessert. Each chef is responsible for a specific part of the order. Therefore, the customer's order is passed from the head chef to the other chefs (bartender for beverages, commis chef for starters and main dish, and pastry chef for dessert) where each chef handles their part of the order. The order gets sent back to the head chef for reviewing.

Problem the pattern solved:

- It ensures that the customer's order goes to the head chef both at the beginning and end of the preparation process.

Decorator

How the pattern was used:

- The customer can add:

  - garnish to their beverage

  - side dishes to their starter and main meal

  - toppings to their dessert

- Customer can add a custom tip to the already existing bill.

Problem the pattern solved:

- Customer has a pool of options and can make multiple combinations.
- Customer has the freedom of specifying the tip amount.

Template

How the pattern was used:

- The maître d and general waiter are both waiters however they have different responsibilities that they must carry out towards the customers. Either the maître d or general waiter gets called depending on what the customer needs.

Problem the pattern solved:

- With the design pattern, we were able to separate the responsibilities of the maître d and the general waiter, and according to what the customer needs the correct waiter would get called.

Visitor

How the pattern was used:

- The manager, head chef and general waiter use this pattern to visit tables to ensure that the customer is happy and good service has been provided.

Problem the pattern solved:

- The manager, head chef, and the general waiter can perform different operations to the customers. The manager collects any complaints made by the customer, the head chef checks the customer's satisfactory level with their order and the waiter checks the customer's satisfactory level with their service.

Façade, Singleton & Mediator

How the pattern was used:

- The Façade acts as the mediator between the client and the subsystems (the kitchen and the floor) by passing the client's requests to the correct subsystem. As we need only one instance of this interface, we used a singleton to enforce this property.

Problem the pattern solved:

- The interface becomes easier to use for the client as it only needs to interact with the façade and, the façade delegates the client's request to appropriate subsystem objects.
- There is a single point of communication which makes it easier to interact with the system
- Using the singleton design pattern on Façade ensures that only one instance of façade is created.

Prototype

How the pattern was used:

- Used to create tables.

Problem the pattern solved:

- Instead of calling the Table constructor each time we want a new table, we clone the one that already exists.

RESEARCH

Creation of the menu

The menu items for the restaurant are similar to the "American Food" menu items seen the Restaurant tycoon 2 game.

The starters are onion rings, chicken wings and chips. The main dishes are cheeseburgers, hot dogs, mac and cheese, BBQ ribs, chicken tenders, cheesesteak, BBQ steak and buffalo wings. The beverages on offer are coke, sprite, coke zero, bubblegum milkshakes and strawberry milkshakes.

The decorator design pattern was used in the customization of the menu items that the customer would order.

The customer has the option to add the coleslaw, mushroom sauce, or salad custom additions to a menu item that they ordered. The reasoning behind using the decorator design pattern for the menu class was the need of a customer to be able to dynamically change their order hence it provides a flexible alternative to subclassing for extending functionality.

Features to add for the customer

Tab system

A tab is a running account of a customer's drink and food purchases over one or many visits. It can be kept on paper or digitally through a point-of-sale system. Tabs save waiters time by consolidating multiple transactions of a single customer into one bill. It also allows for the separation of the bill in cases where a large group wants to pay for their drinks and food individually.

Allow a customer to open a tab in their name. However, they lose the freedom of choosing a method of payment as tabs need to be strictly paid with a credit or debit card to avoid customers leaving without paying the bill. Even in cases where they can settle their bill in days, their card information is essential so a follow up can be made.

Reservation procedure

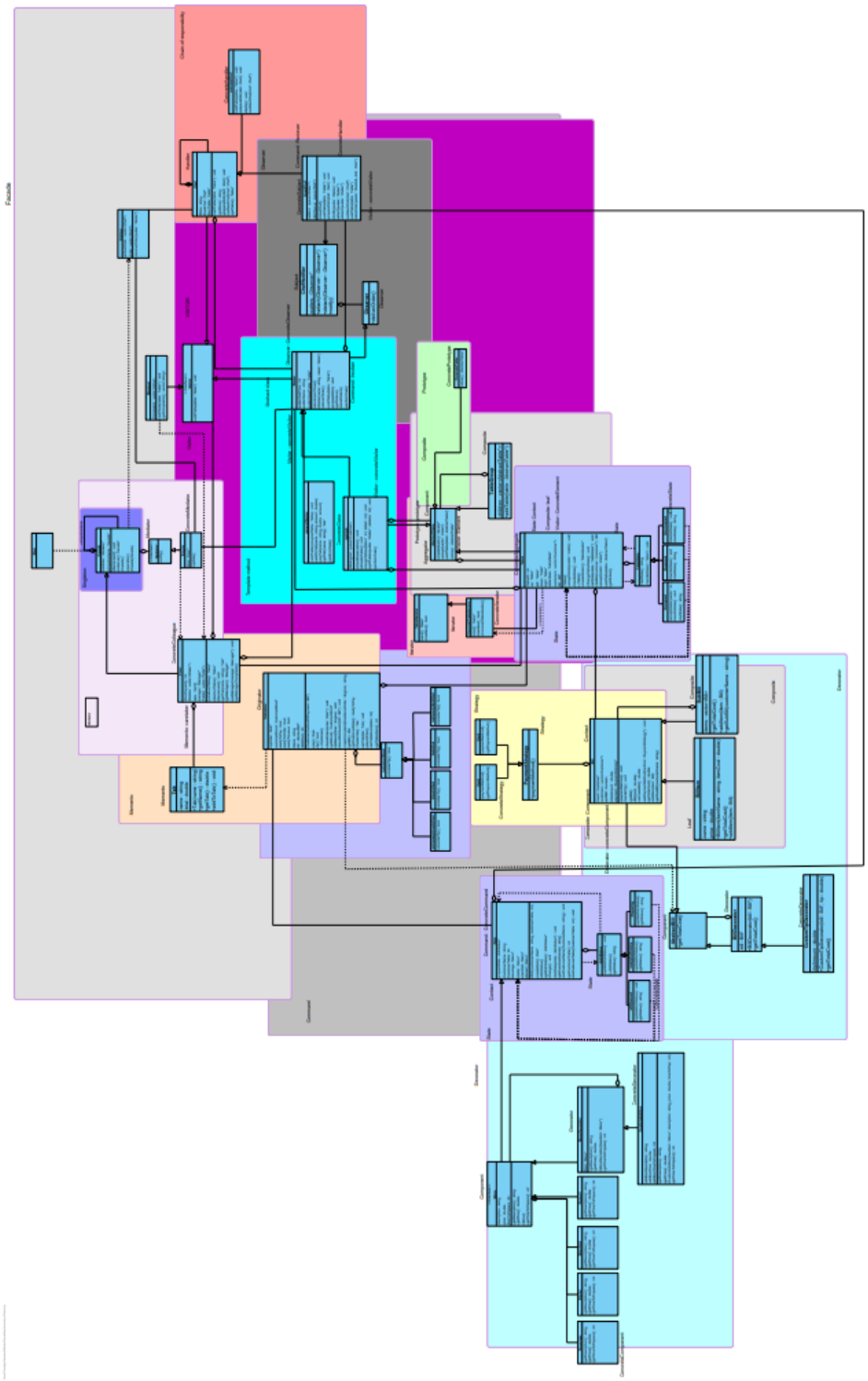
Information needed from a customer:
- Name
- Number of guests
- Date and time

Customer complaints system

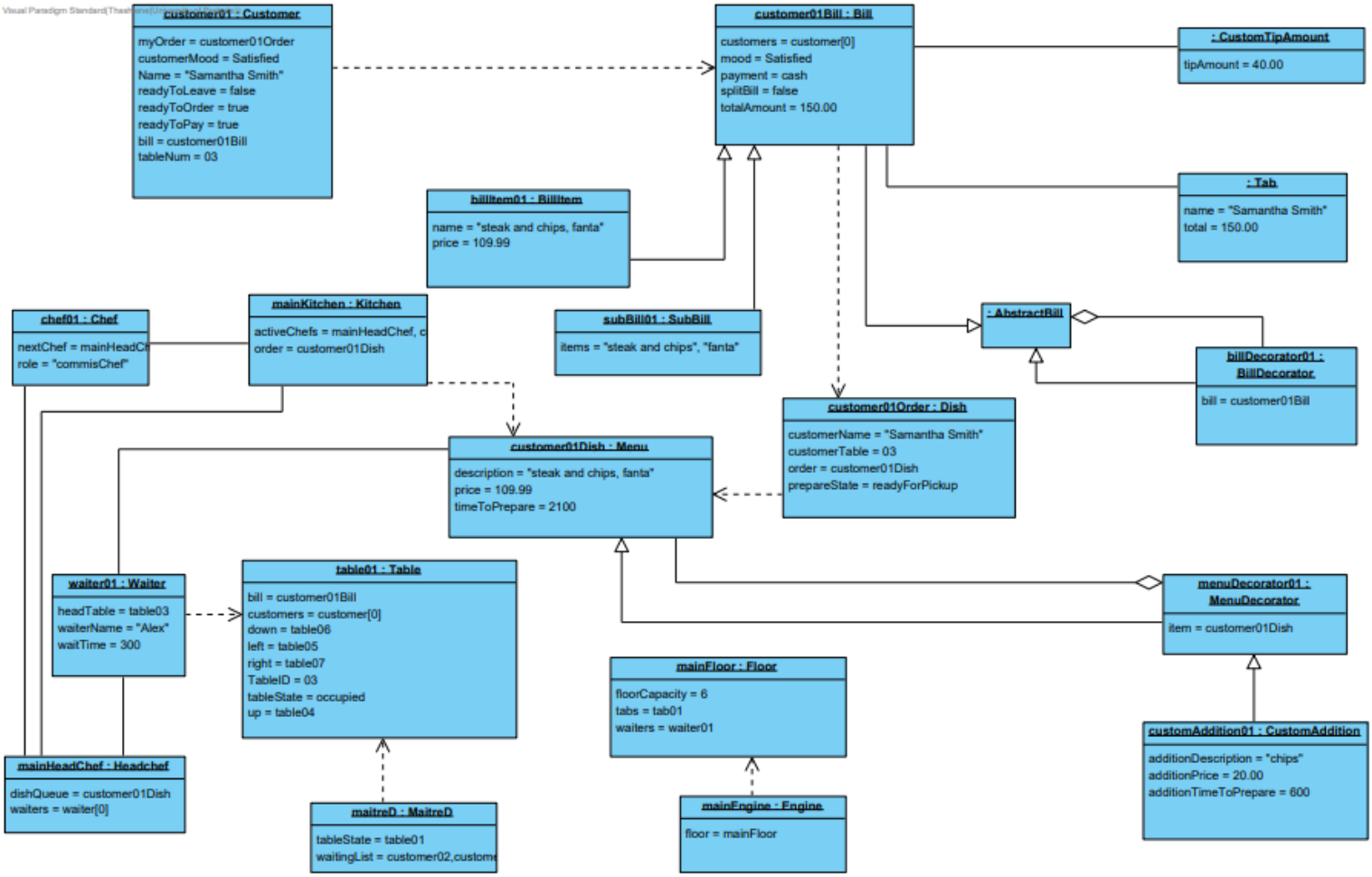
The best person to handle a complaint in a restaurant is the manager. The chain of command is comprised of the waiter, manager, and the owner of the restaurant respectively. Depending on how difficult the situation is, the customer can ask for the manager directly, skipping the waiter. However, only the manager can escalate the problem to the owner of the restaurant.

UML DIAGRAMS (after implementation)

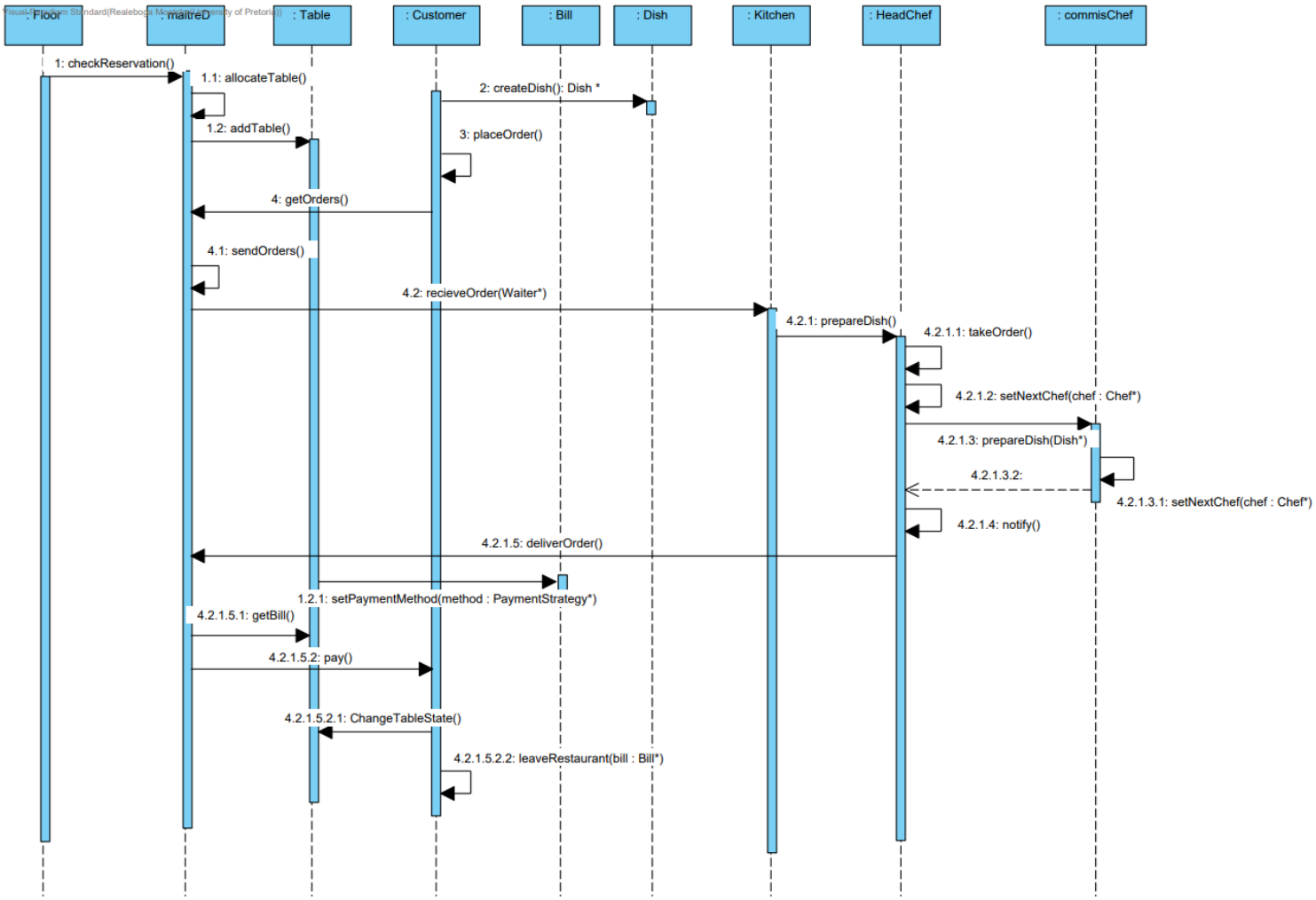
Class diagram



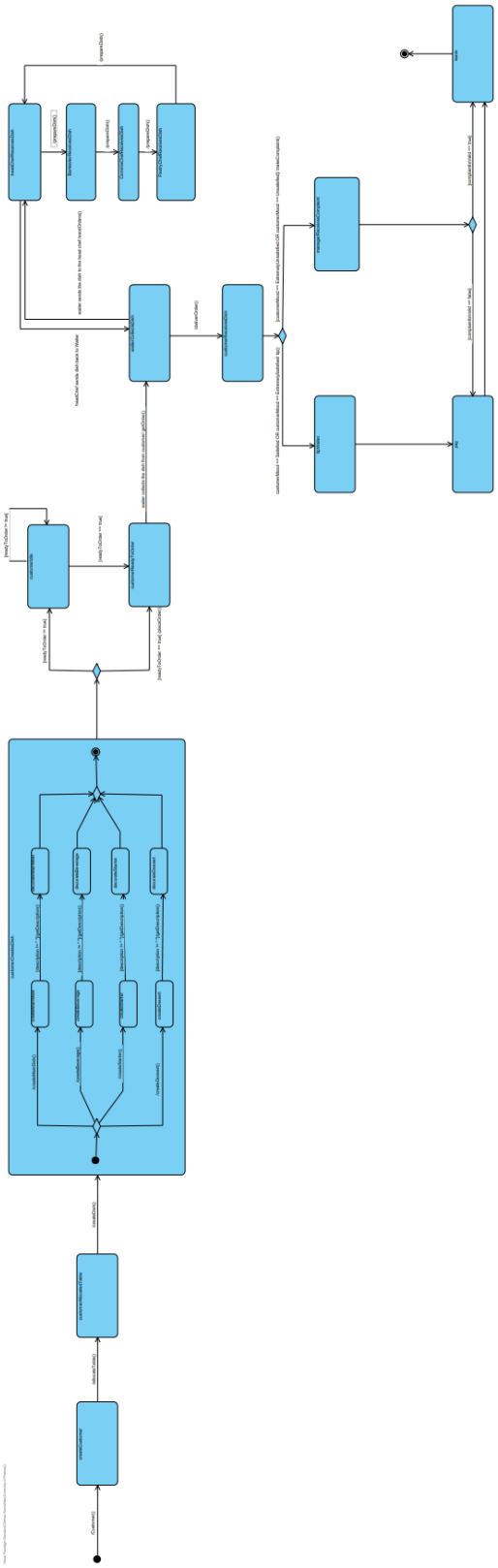
Object diagram



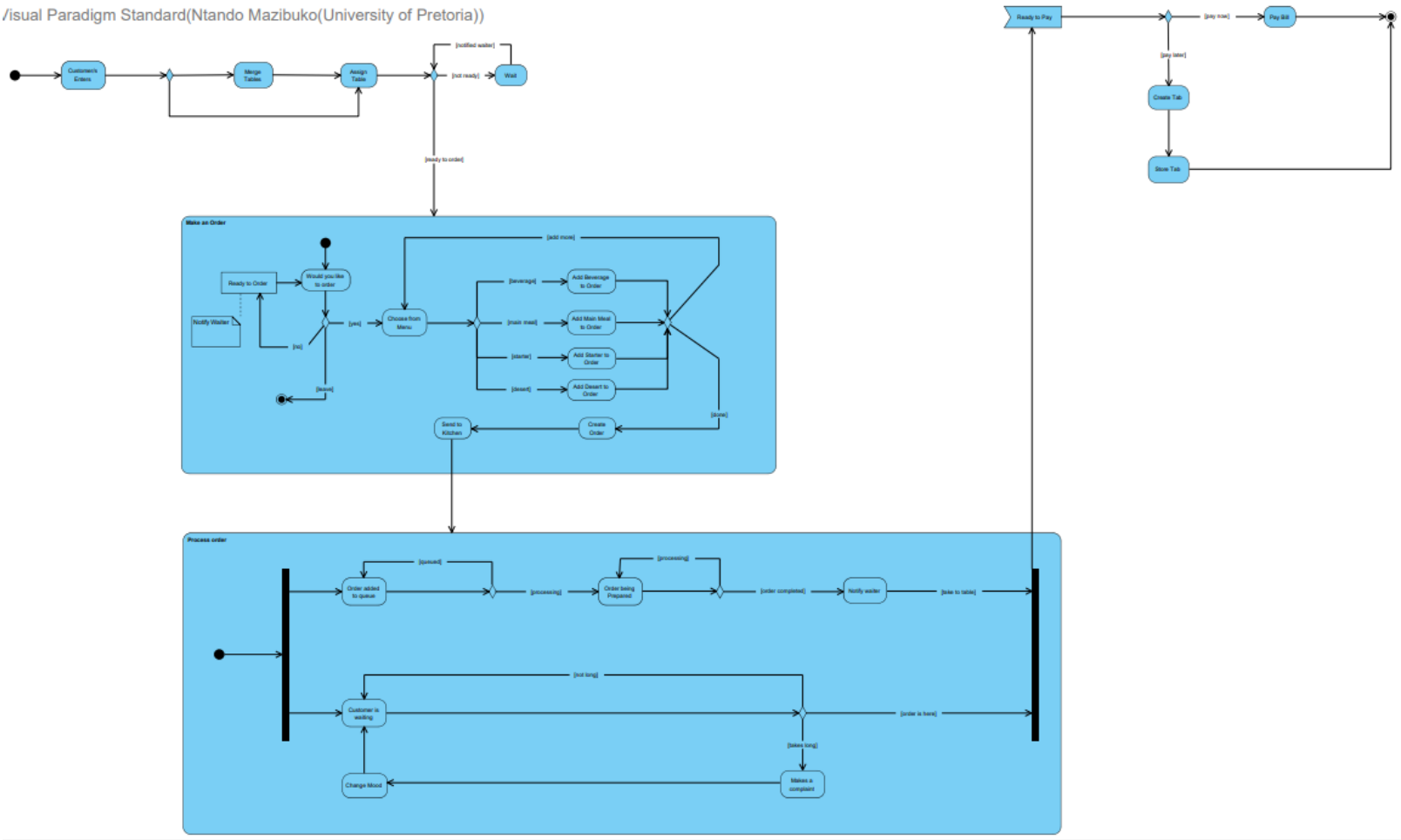
Sequence diagram



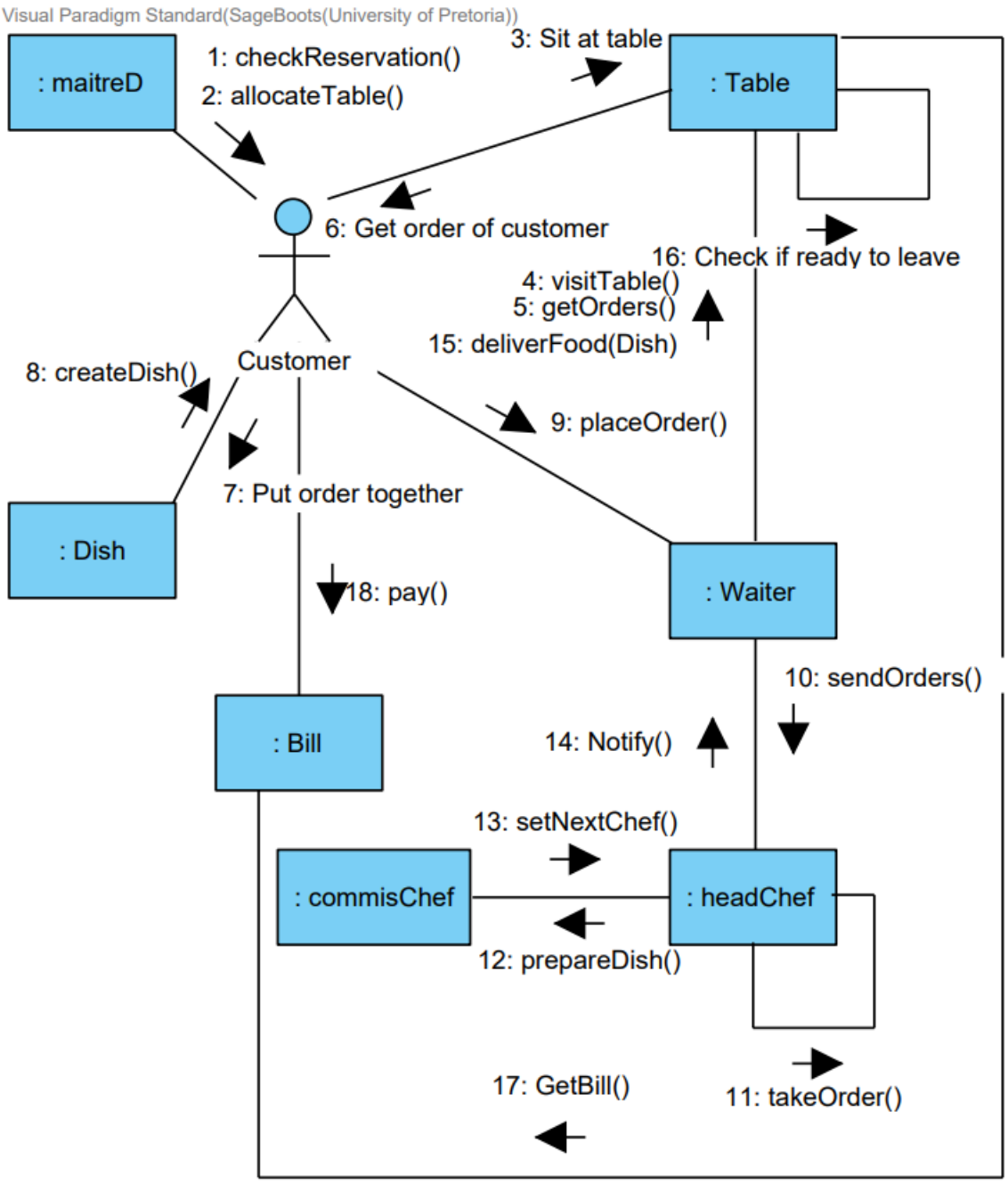
State diagram



Activity diagram



Communication diagram



EXPLANATION OF SYSTEM

Overview:

The system is effectively a simulator that uses an engine class and multithreading to simulate time passing by using a sleeping thread.

This engine controls the timers for multiple types of objects in the system which includes the amount of time a waiter takes before traversing the floor, where the waiter checks to see if a table is prepared to order or not.

The engine controls the customer satisfaction in that, if the waiter takes too long to return, that affects the customers mood and the tip possible.

The engine also controls the kitchen in the sense that it allows dishes to have a time taken attribute where the longer a dish takes to make, the worse the customers mood gets.

The system also contains a "MaitreD" that manages table allocations and the splitting and merging of tables.

The floor of the restaurant also allows the capabilities for customers to create tab objects such that customers can add and remove from their tab as an object.

The customers have the capability to combine n bills at their table or separate the bill among n people with each person having the ability to create or add to a tab using their current sub-bill.

This is all encapsulated in a "facade" class which allows the interaction of the user and the subclasses in which the subclasses are not aware of the facade but can interact with it to simplify the user's view of the system.

LINKS

Git repository

Diagrams

REFERENCES

1. https://restaurant-tycoon-2.fandom.com/wiki/Foods#American_Food
2. https://www.webstaurantstore.com/blog/3660/bar-tabs-and-preauthorization.html
3. https://www.jotform.com/form-templates/restaurant-reservation-form
4. https://assets.website-files.com/602da0632e0ff07c4548b93b/62578b9caead2d3e47282088_Customer%20Complaint%20Policy.pdf