Design a Restaurant Management system

Let's design a restaurant management system.

We'll cover the following

A Restaurant Management System is a software built to handle all restaurant activities in an easy and safe manner. This System will give the Restaurant management power and flexibility to manage the entire system from a single portal. The system allows the manager to keep track of available tables in the system as well as the reservation of tables and bill generation.

System Requirements

We will focus on the following set of requirements while designing the Restaurant Management System:

1. The restaurant will have different branches.
2. Each restaurant branch will have a menu.
3. The menu will have different menu sections, containing different menu items.
4. The waiter should be able to create an order for a table and add meals for each seat.
5. Each meal can have multiple meal items. Each meal item corresponds to a menu item.
6. The system should be able to retrieve information about tables currently available to seat walk-in customers.
7. The system should support the reservation of tables.
8. The receptionist should be able to search for available tables by date/time and reserve a table.
9. The system should allow customers to cancel their reservation.
10. The system should be able to send notifications whenever the reservation time is approaching.
11. The customers should be able to pay their bills through credit card, check or cash.
12. Each restaurant branch can have multiple seating arrangements of tables.

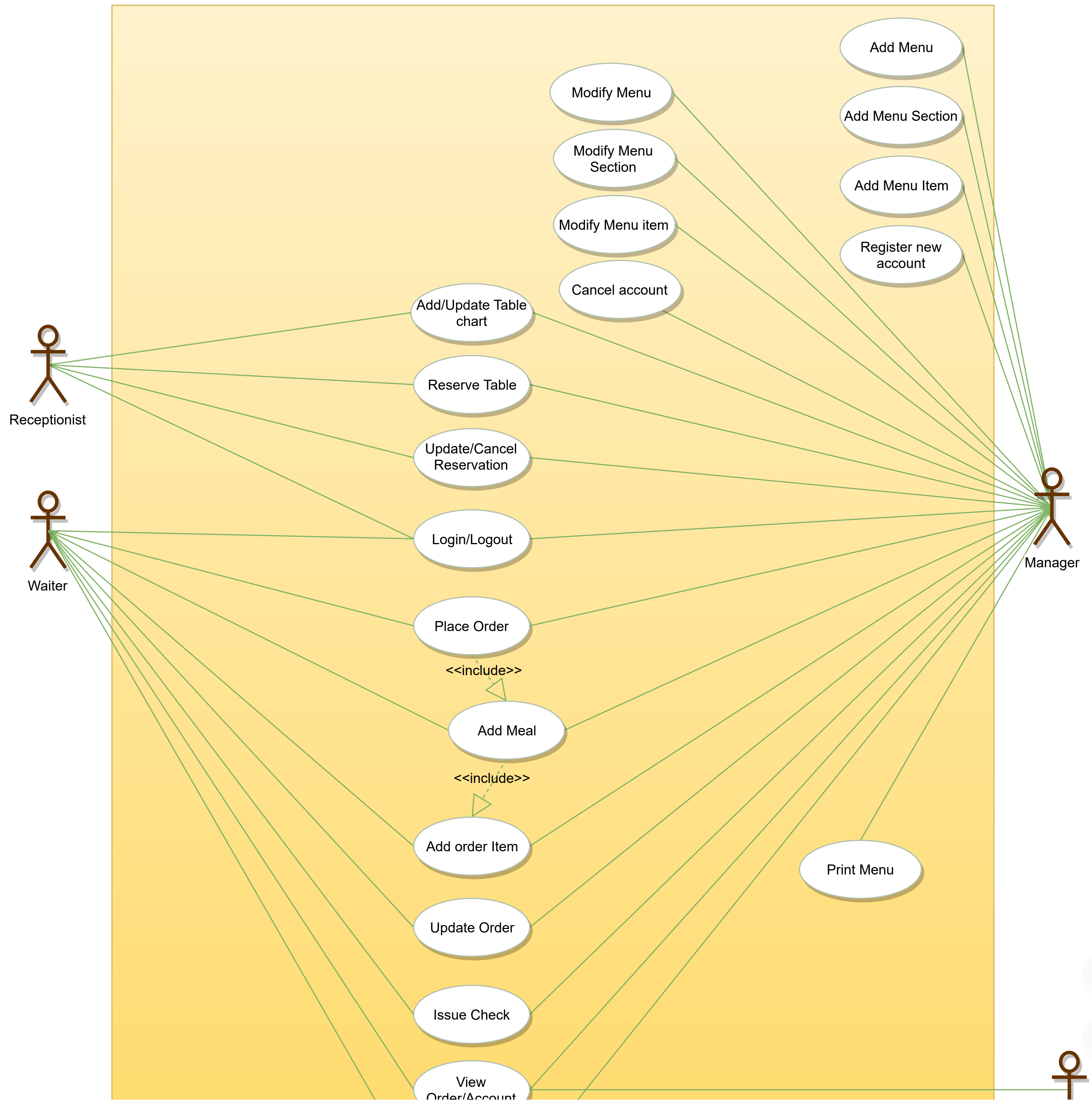
Use case diagram

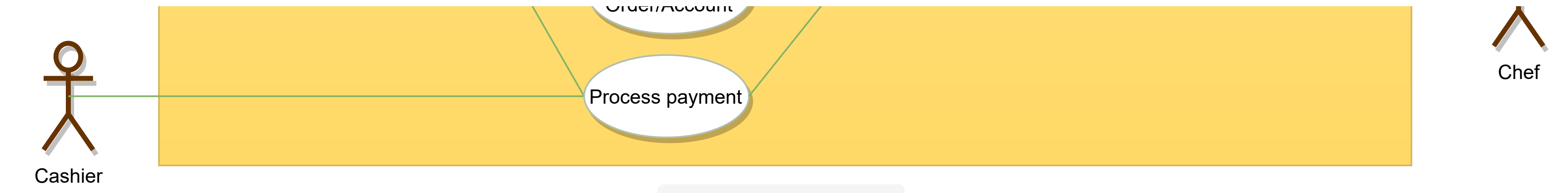
Here are the main Actors in our system:

- **Receptionist:** Mainly responsible for adding and modifying tables and their layout, and creating and canceling table reservations.
- **Waiter:** To take/modify orders.
- **Manager:** Mainly responsible for adding new workers and modifying the menu.
- **Chef:** To view and work on an order.
- **Cashier:** To generate checks and process payments.
- **System:** Mainly responsible for sending notifications about table reservations, cancellations, etc.

Here are the top use cases of the Restaurant Management System:

- **Add/Modify tables:** To add, remove, or modify a table in the system.
- **Search tables:** To search for available tables for reservation.
- **Place order:** Add a new order in the system for a table.
- **Update order:** Modify an already placed order, which can include adding/modifying meals or meal items.
- **Create a reservation:** To create a table reservation for a certain date/time for an available table.
- **Cancel reservation:** To cancel an existing reservation.
- **Check-in:** To let the guest check in for their reservation.
- **Make payment:** Pay the check for the food.



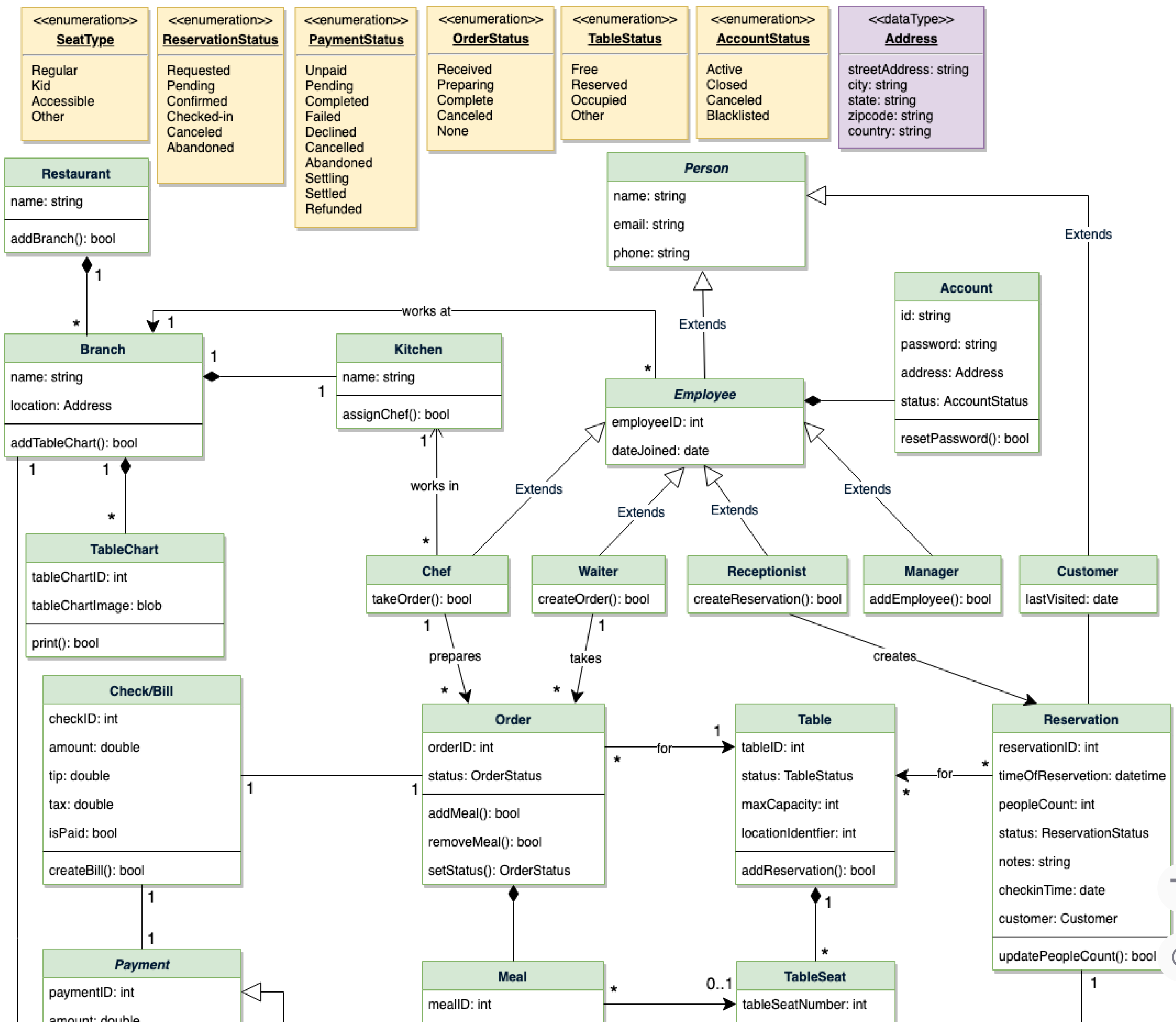


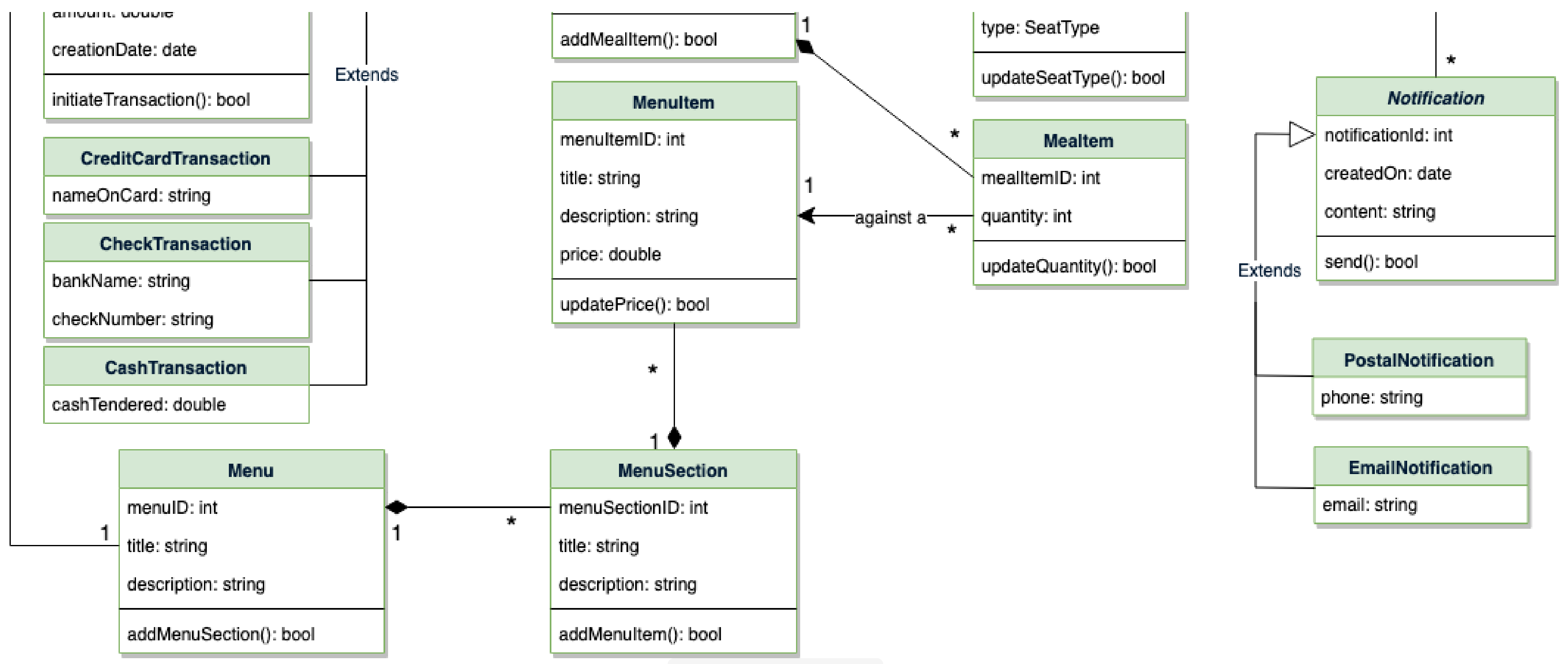
Use case diagram

Class diagram

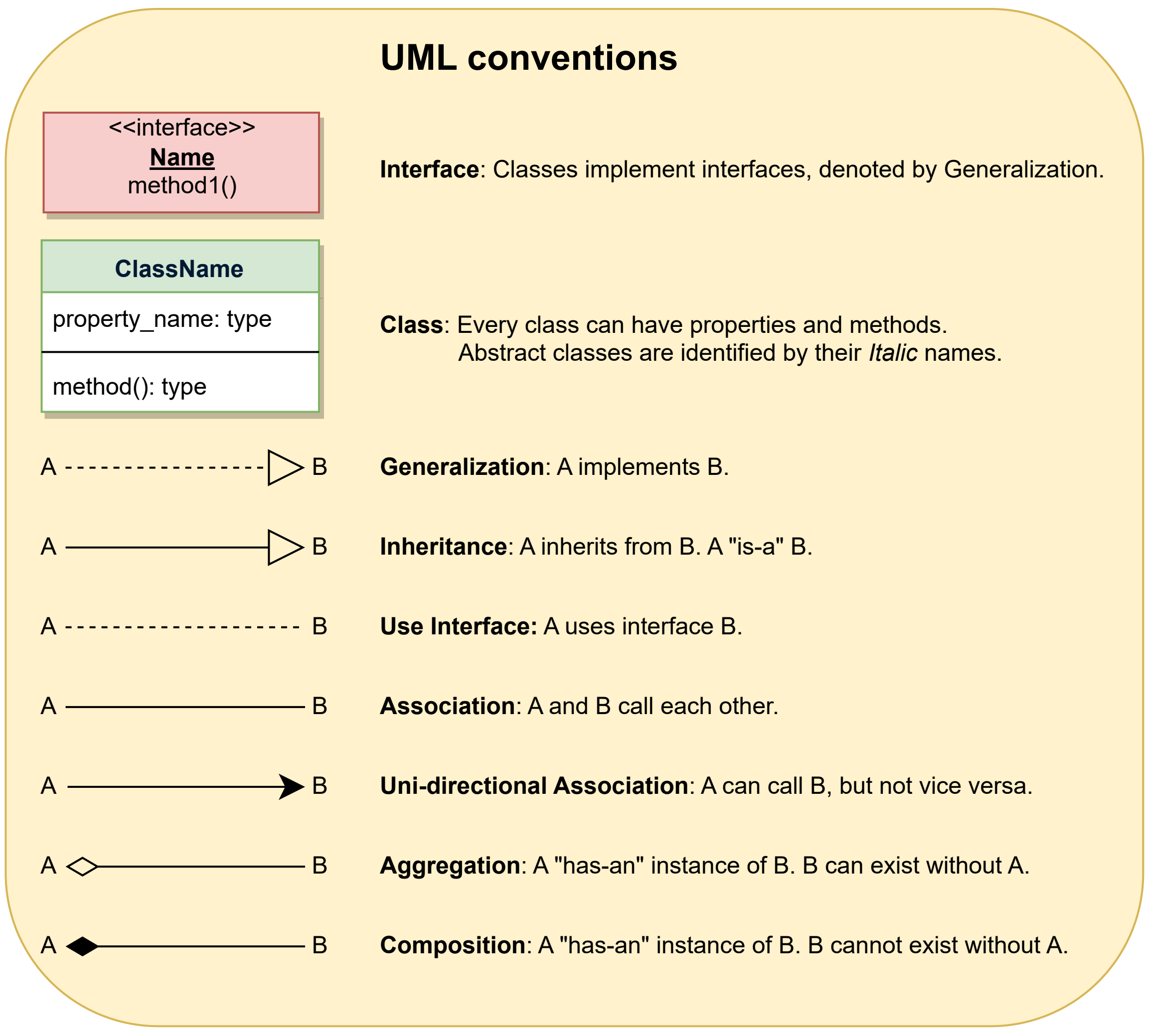
Here is the description of the different classes of our Restaurant Management System:

- **Restaurant:** This class represents a restaurant. Each restaurant has registered employees. The employees are part of the restaurant because if the restaurant becomes inactive, all its employees will automatically be deactivated.
- **Branch:** Any restaurants can have multiple branches. Each branch will have its own set of employees and menus.
- **Menu:** All branches will have their own menu.
- **MenuSection and MenuItem:** A menu has zero or more menu sections. Each menu section consists of zero or more menu items.
- **Table and TableSeat:** The basic building block of the system. Every table will have a unique identifier, maximum sitting capacity, etc. Each table will have multiple seats.
- **Order:** This class encapsulates the order placed by a customer.
- **Meal:** Each order will consist of separate meals for each table seat.
- **Meal Item:** Each Meal will consist of one or more meal items corresponding to a menu item.
- **Account:** We'll have different types of accounts in the system, one will be a receptionist to search and reserve tables and the other, the waiter will place orders in the system.
- **Notification:** Will take care of sending notifications to customers.
- **Bill:** Contains different bill-items for every meal item.



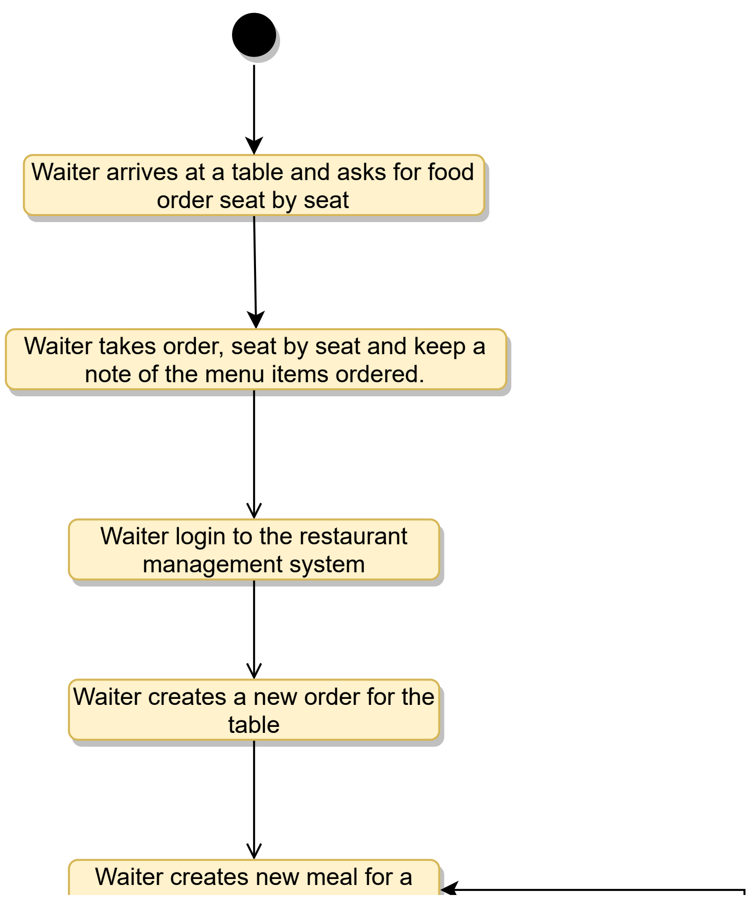


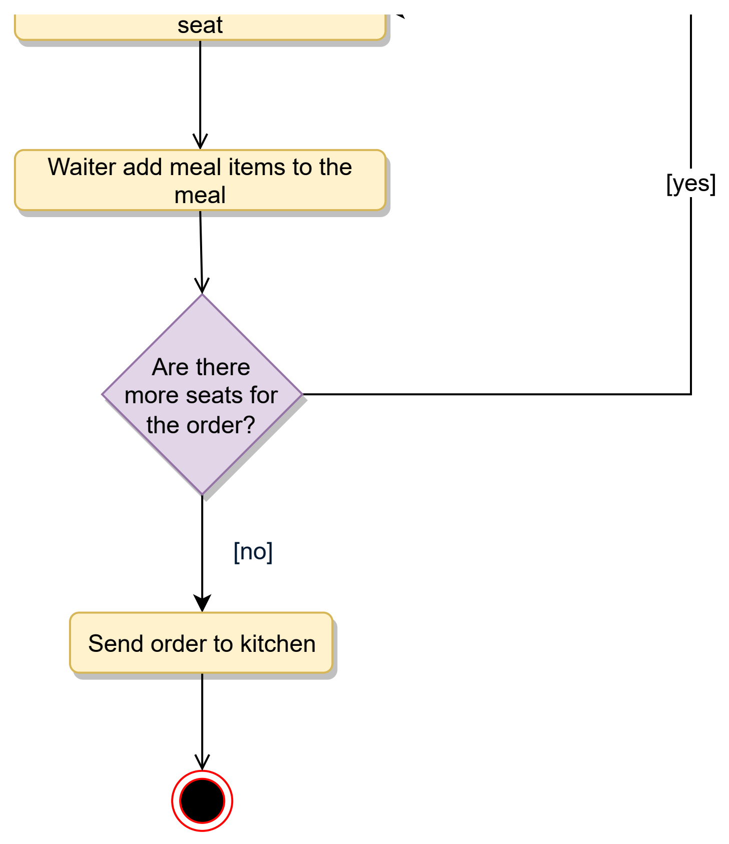
Class diagram


UML conventions
<<interface>>
Name
method1()
Interface: Classes implement interfaces, denoted by Generalization.
ClassName
property_name: type
method(): type
Class: Every class can have properties and methods.
Abstract classes are identified by their Italic names.
A B
Generalization: A implements B.
A B
Inheritance: A inherits from B. A "is-a" B.
A B
Use Interface: A uses interface B.
A B
Association: A and B call each other.
A B
Uni-directional Association: A can call B, but not vice versa.
A B
Aggregation: A "has-an" instance of B. B can exist without A.
A B
Composition: A "has-an" instance of B. B cannot exist without A.

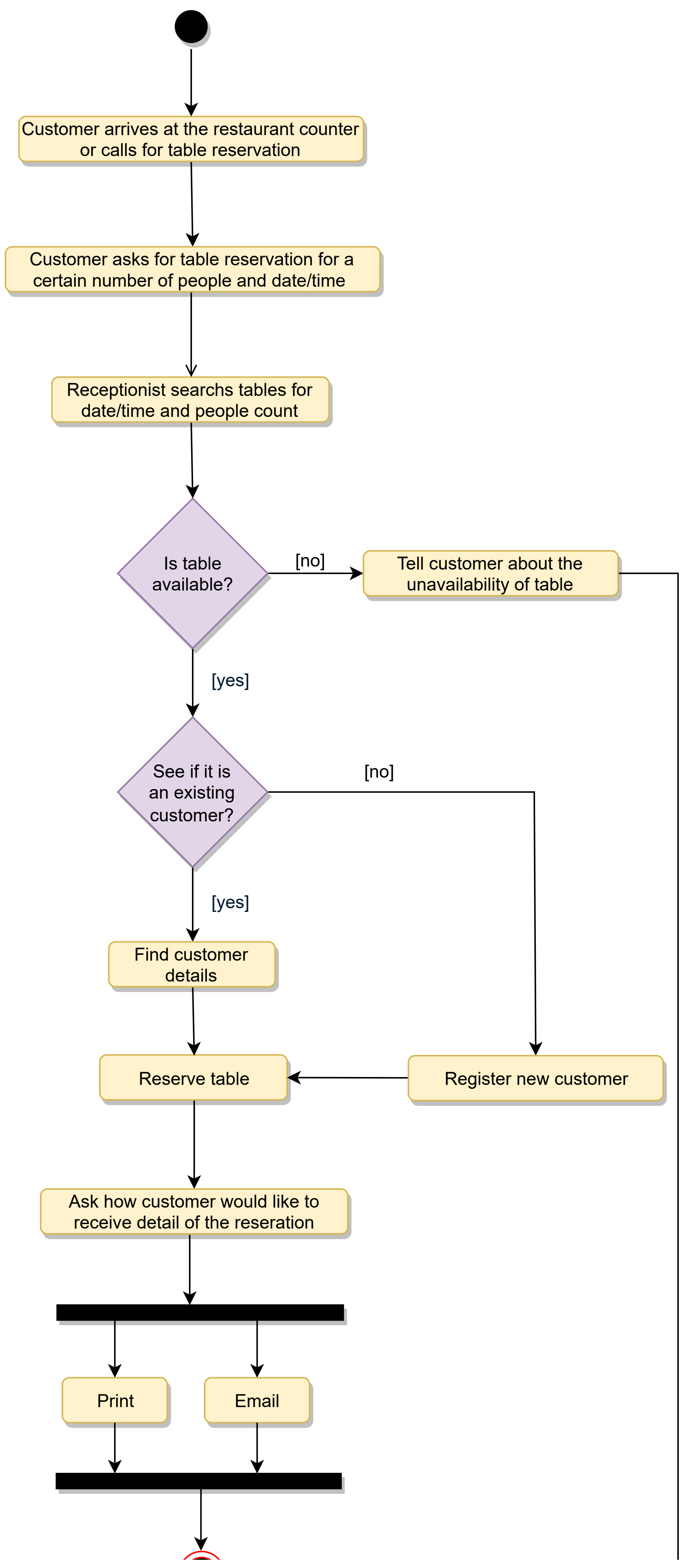
Activity diagrams

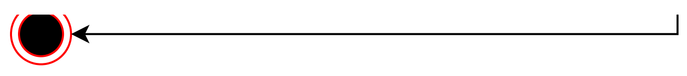
Place order: Any waiter can perform this activity. Here are the steps to place an order:


Waiter arrives at a table and asks for food order seat by seat
Waiter takes order, seat by seat and keep a note of the menu items ordered.
Waiter login to the restaurant management system
Waiter creates a new order for the table
Waiter creates new meal for a

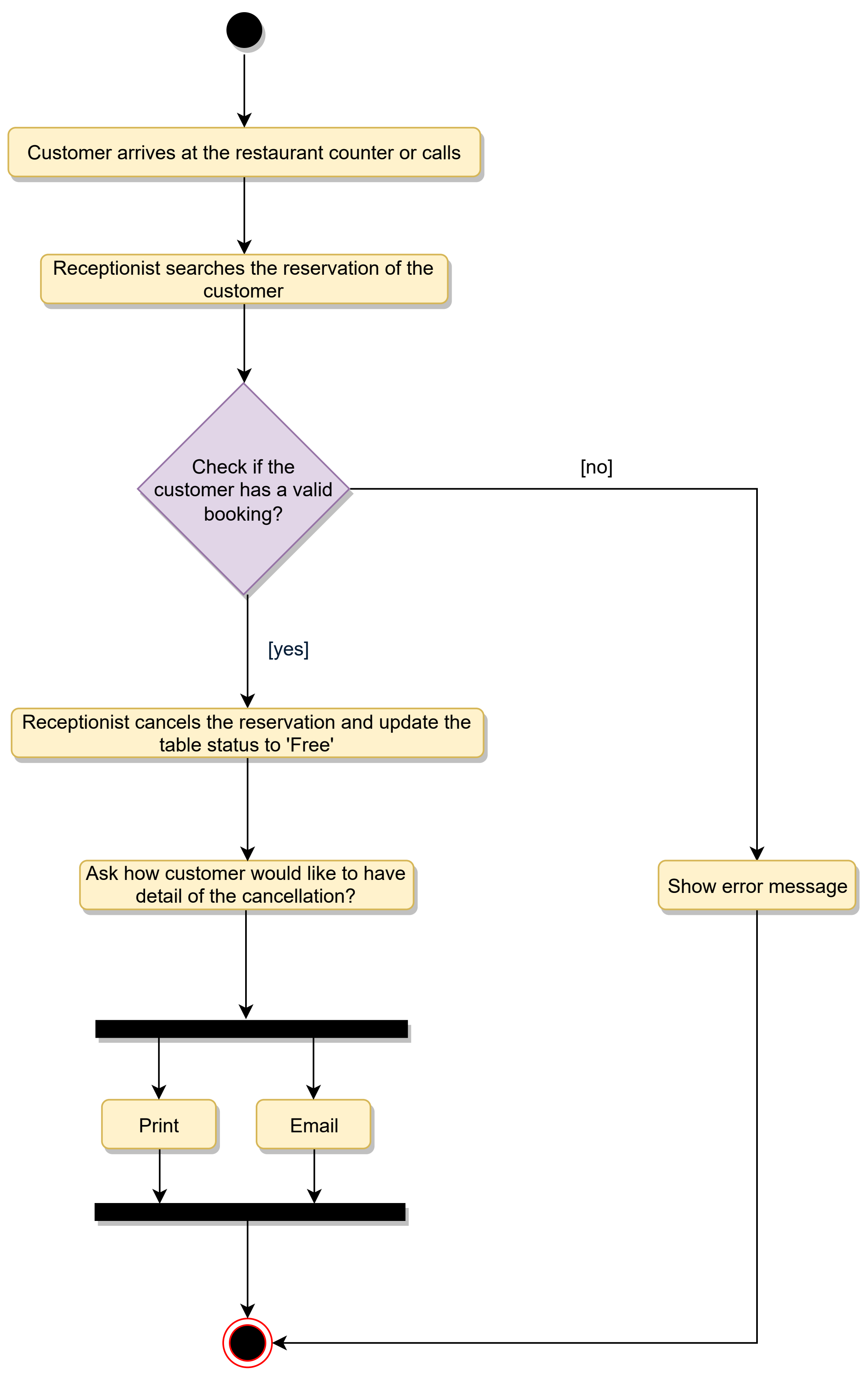

seat
Waiter add meal items to the meal
[yes]
Are there more seats for the order?
[no]
Send order to kitchen

Make a reservation: Any receptionist can perform this activity. Here are the steps to make a reservation:


Customer arrives at the restaurant counter or calls for table reservation
Customer asks for table reservation for a certain number of people and date/time
Receptionist searchs tables for date/time and people count
Is table available?
[no]
Tell customer about the unavailability of table
[yes]
See if it is an existing customer?
[no]
[yes]
Find customer details
Reserve table
Register new customer
Ask how customer would like to receive detail of the reseration
Print
Email



Cancel a reservation: Any receptionist can perform this activity. Here are the steps to cancel a reservation:


Customer arrives at the restaurant counter or calls
Receptionist searches the reservation of the customer
Check if the customer has a valid booking?
[no]
[yes]
Receptionist cancels the reservation and update the table status to 'Free'
Ask how customer would like to have detail of the cancellation?
Show error message
Print
Email

Code

Here is the high-level definition for the classes described above.

Enums, data types, and constants: Here are the required enums, data types, and constants:

Java | Python

```
class ReservationStatus(Enum):
  REQUESTED, PENDING, CONFIRMED, CHECKED_IN, CANCELED, ABANDONED = 1, 2, 3, 4, 5, 6


class SeatType(Enum):
  REGULAR, KID, ACCESSIBLE, OTHER = 1, 2, 3, 4


class OrderStatus(Enum):
  RECEIVED, PREPARING, COMPLETED, CANCELED, NONE = 1, 2, 3, 4, 5


class TableStatus(Enum):
  FREE, RESERVED, OCCUPIED, OTHER = 1, 2, 3, 4


class AccountStatus(Enum):
  ACTIVE, CLOSED, CANCELED, BLACKLISTED, BLOCKED = 1, 2, 3, 4, 5
```

```
class PaymentStatus(Enum):
  UNPAID, PENDING, COMPLETED, FILLED, DECLINED, CANCELLED, ABANDONED, SETTLING, SETTLED, REFUNDED = 1, 2, 3, 4,


class Address:
  def __init__(self, street, city, state, zip_code, country):
    self.__street_address = street
```

Account, Person, Employee, Receptionist, Manager, and Chef: These classes represent the different people that interact with our system:

Java Python

```
# For simplicity, we are not defining getter and setter functions. The reader can
# assume that all class attributes are private and accessed through their respective
# public getter methods and modified only through their public methods function.


class Account:
  def __init__(self, id, password, address, status=AccountStatus.Active):
    self.__id = id
    self.__password = password
    self.__address = address
    self.__status = status

  def reset_password(self):
    None


# from abc import ABC, abstractmethod
class Person(ABC):
  def __init__(self, name, email, phone):
    self.__name = name
    self.__email = email
    self.__phone = phone


# from abc import ABC, abstractmethod
class Employee(ABC, Person):
  def __init__(self, id, account, name, email, phone):
```

Restaurant, Branch, Kitchen, TableChart: These classes represent the top-level classes of the system:

Java Python

```
class Kitchen:
  def __init__(self, name):
    self.__name = name
    self.__chefs = []

  def assign_chef(self, chef):
    None


class Branch:
  def __init__(self, name, location, kitchen):
    self.__name = name
    self.__location = location
    self.__kitchen = kitchen

  def add_table_chart(self):
    None


class Restaurant:
  def __init__(self, name):
    self.__name = name
```

```
        self.__branches = []

    def add_branch(self, branch):
        None
```

Table, TableSeat, and Reservation: Each table can have multiple seats and customers can make reservations for tables:

Java Python

```
class Table:
    def __init__(self, id, max_capacity, location_identifier, status=TableStatus.FREE):
        self.__table_id = id
        self.__max_capacity = max_capacity
        self.__location_identifier = location_identifier
        self.__status = status
        self.__seats = []

    def is_table_free(self):
        None

    def add_reservation(self):
        None

    def search(self, capacity, start_time):
        # return all tables with the given capacity and availability
        None


class TableSeat:
    def __init__(self):
        self.__table_seat_number = 0
        self.__type = SeatType.REGULAR

    def update_seat_type(self, seat_type):
        None
```

Menu, MenuSection, and MenuItem: Each restaurant branch will have its own menu, each menu will have multiple menu sections, which will contain menu items:

Java Python

```
class MenuItem:
    def __init__(self, id, title, description, price):
        self.__menu_item_id = id
        self.__title = title
        self.__description = description
        self.__price = price

    def update_price(self,  price):
        None


class MenuSection:
    def __init__(self, id, title, description):
        self.__menu_section_id = id
        self.__title = title
        self.__description = description
        self.__menu_items = []

    def add_menu_item(self, menu_item):
        None


class Menu:
    def __init__(self, id, title, description):
        self.__menu_id = id
        self.__title = title
    self.__description = description
```

Order, Meal, and MealItem: Each order will have meals for table seats:

Java Python

```
class MealItem:
  def __init__(self, id, quantity, menu_item):
    self.__meal_item_id = id
    self.__quantity = quantity
    self.__menu_item = menu_item

  def update_quantity(self, quantity):
    None


class Meal:
  def __init__(self, id, seat):
    self.__meal_id = id
    self.__seat = seat
    self.__menu_items = []

  def add_meal_item(self, meal_item):
    None


class Order:
  def __init__(self, id, status, table, waiter, chef):
    self.__order_id = id
    self.__OrderStatus = status
    self.__creation_time = datetime.datetime.now()

    self.__meals = []
```

Back

Design a Hotel Management System

Next

Design Chess

✓ Mark as Completed