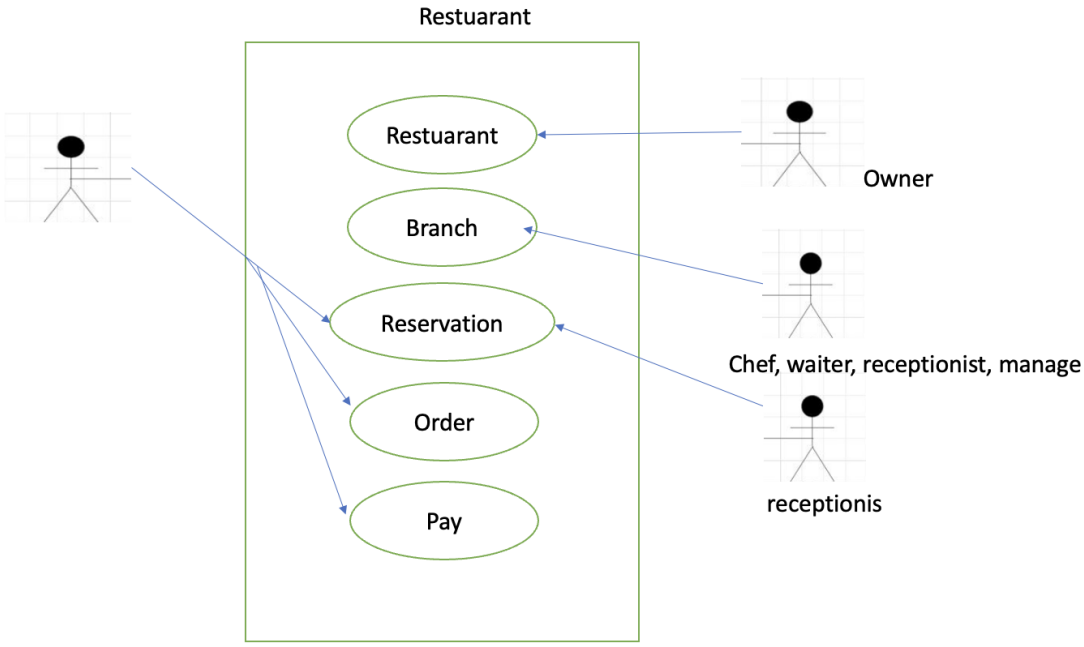
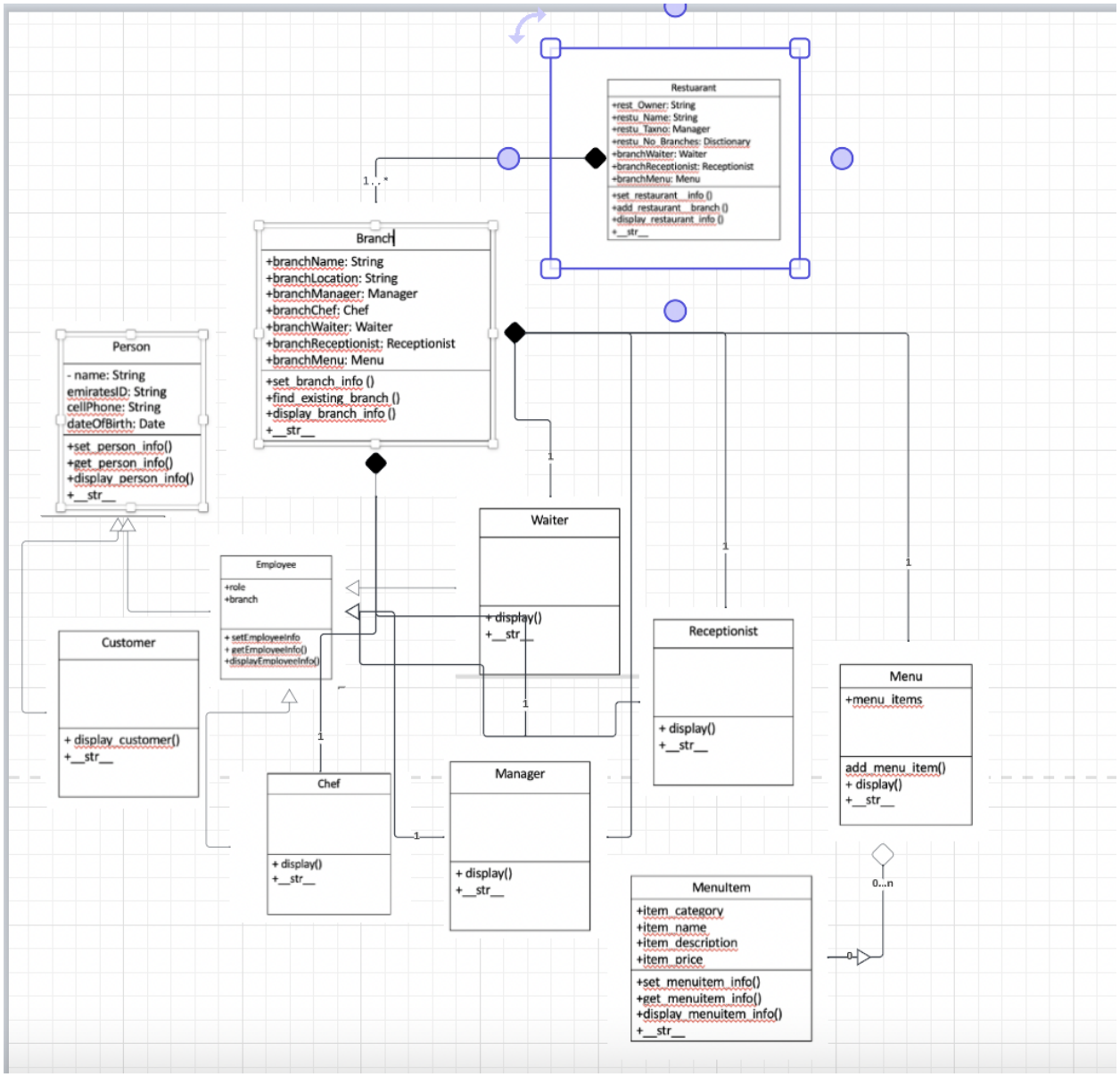
USE CASE DIAGRAM





USE CASE DESCRIPTION

Use Case	**Restaurant**
Trigger	Restaurants has branches selling food
Precondition	Restuarants need to file employees information

Main Scenario	
	The owner opens a Branch and employs workers
	The owner files their infomation
	The system updates

Use Case	**Menu**
Trigger	The owner wants to create and manage the menu for the restaurant.
Precondition	The owner has logged into the restaurant management system
Main Scenario	
	The manager selects the "Menu item" option from the system
	The system presents a form or interface for the manager to input menu details.
	The owner provides information such as item category, name, description, and price for each menu item.
	The new menuitem is created and saved in the Menu

Use Case	**Menu**
Trigger	The owner wants to create and manage the menu for the restaurant.
Precondition	The owner has logged into the restaurant management system
Main Scenario	

	The manager selects the "Menu item" option from the system
	The system presents a form or interface for the manager to input menu details.
	The owner provides information such as item category, name, description, and price for each menu item.
	The new menuitem is created and saved in the Menu

Use Case	**Reservation**
Trigger	The receptionist wants to handle a reservation requested by a customer
Precondition	The customer initiates the reservation process
Main Scenario	
	The customer contacts the restaurant to make a reservation.
	TThe receptionist accesses the reservation system and selects the "Handle Reservation" option.
	The system prompts the receptionist to enter reservation details, including the customer's name, desired time, date, table, and the number of people.
	The system checks the availability of the selected table at the specified time and date.
	The receptionist reserve

Use Case	Order

Trigger	The waiter is assigned to a table with a customer
Precondition	The customer initiates the reservation process
Main Scenario	
	The customer reviews the menu and discusses their preferences with the waiter.
	The waiter accesses the order system and selects the "Place Order" option.
	The system displays the menu, and the waiter adds items selected by the customer to the order.
	For each item, the system prompts the waiter to specify additional details such as quantity or special instructions.
	The kitchen acknowledges the order, and the waiter informs the customer that the order is being prepared.
	The customer receives the order

Code

```
import tkinter as tk
from tkinter import ttk
from itertools import count
from tabulate import tabulate

class Person:
    """
    Represents a general person with basic information.
    """

    def __init__(self):
        self._name = None
        self._emirates_id = None
        self._cell_phone = None
        self._date_of_birth = None
```

```
    def set_person_info(self, fullname, emirates_id, phone, date_of_birth):
        """
        Set the information of a person.

        Args:
            fullname (str): The full name of the person.
            emirates_id (str): The Emirates ID of the person.
            phone (str): The cell phone number of the person.
            date_of_birth (str): The date of birth of the person.
        """
        self._name = fullname
        self._emirates_id = emirates_id
        self._cell_phone = phone
        self._date_of_birth = date_of_birth if date_of_birth is not None
else None

    def get_person_info(self):
        """
        Get the information of a person.

        Returns:
            tuple: A tuple containing the person's information (name,
Emirates ID, cell phone, date of birth).
        """
        return self._name, self._emirates_id, self._cell_phone,
self._date_of_birth

    def display_person_info(self):
        """
        Generate a formatted string representing the person's information.

        Returns:
            str: A formatted string containing the person's information.
        """
        return f'Full Name: {self._name}, Emirates ID: {self._emirates_id},
Phone: {self._cell_phone}, Date Of Birth: {self._date_of_birth}'

    def __str__(self):
        """
        Generate a string representation of a person.
```

```
        Returns:
            str: A string containing the person's information.
        """
        return f'Full Name: {str(self._name)}, ID:
{str(self._emirates_id)}, Phone: {str(self._cell_phone)}, Date Of Birth:
{str(self._date_of_birth)}'
class Employee(Person):
    """
    Represents an employee of the hospital with additional department and
room information.
    """

    def __init__(self):
        super().__init__()
        self.role = ""
        self.branch = ""

    def set_employee_info(self, fullname, emirates_id, phone, dateofbirth,
role, branch):
        """
        Set the information of an employee, including personal information,
department, and room.

        Args:
            fullname (str): The full name of the employee.
            emirates_id (str): The Emirates ID of the employee.
            phone (str): The phone number of the employee.
            dateofbirth (str): The date of birth of the employee in
"YYYY-MM-DD" format.
            role (str): The department where the employee works.
            branch (str): The room where the employee is assigned.
        """
        super().set_person_info(fullname, emirates_id, phone, dateofbirth)
        self.role = role
        self.branch = branch

    def get_employee_info(self):
        """
```

```
        Get the employee information, including personal information,
department, and room.

        Returns:
            tuple: A tuple containing personal information and
employee-specific details
                   (name, Emirates ID, phone, insurance, date of birth,
department, room).
        """
        return super().display_person_info(), self.role, self.branch

    def display_employee_info(self):
        """
        Generate a formatted string representing an employee's information.

        Returns:
            str: A formatted string containing the employee's information.
        """
        return f'{super().display_person_info()}, Department: {self.role},
Room: {self.branch}'

    def __str__(self):
        """
        Generate a string representation of an employee.

        Returns:
            str: A string containing the employee's information.
        """
        person_info, role, branch = self.get_employee_info()
        return f'{str(person_info)}, Department: {str(role)}, Room:
{str(branch)}'




class Waiter(Employee):
    """
    Represents a waiter, inheriting from the Employee class.
    """
```

```
    def __init__(self):
        super().__init__()

    def display_waiter_info(self):
        """
        Generate a formatted string representing the waiter's information.

        Returns:
            str: A formatted string containing the waiter's information.
        """
        return f"{super().display_employee_info()}"

    def __str__(self):
        """
        Generate a string representation of a waiter.

        Returns:
            str: A string containing the waiter's information.
        """
        return f'{str(super().display_employee_info())}'

class Order:
    """
    Represents an order, including information about the waiter, customer,
menu item, order number, price, and discount.
    """

    def __init__(self):
        self.waiter = Waiter()
        self.table = None
        self.menu_item = None
        self.num_order = None
        self.price = None
        self.discount = None
        self.order_list = {}

    def set_waiter(self, waiter):
        """
        Set the waiter for the order.
```

```
        Args:
            waiter (Waiter): The waiter for the order.
        """
        self.waiter = waiter

    def set_table(self, customer):
        """
        Set the customer for the order.

        Args:
            customer (Customer): The customer for the order.
        """
        self.customer = customer

    def set_menu_item(self, menu_item):
        """
        Set the menu item for the order.

        Args:
            menu_item: The menu item for the order.
        """
        self.menu_item = menu_item

    def set_num_order(self, num_order):
        """
        Set the order number.

        Args:
            num_order: The order number.
        """
        self.num_order = num_order

    def set_price(self, price):
        """
        Set the price for the order.

        Args:
            price: The price for the order.
        """
        self.price = price
```

```
def set_discount(self, discount):
    """
    Set the discount for the order.

    Args:
        discount: The discount for the order.
    """
    self.discount = discount

def get_waiter(self):
    """
    Get the waiter for the order.

    Returns:
        Waiter: The waiter for the order.
    """
    return self.waiter

def get_table(self):
    """
    Get the customer for the order.

    Returns:
        Customer: The customer for the order.
    """
    return self.customer

def get_menu_item(self):
    """
    Get the menu item for the order.

    Returns:
        The menu item for the order.
    """
    return self.menu_item

def get_num_order(self):
    """
    Get the order number.
```

```
        Returns:
            The order number.
        """
        return self.num_order

    def get_price(self):
        """
        Get the price for the order.

        Returns:
            The price for the order.
        """
        return self.price

    def get_discount(self):
        """
        Get the discount for the order.

        Returns:
            The discount for the order.
        """
        return self.discount

    def add_items(self, waiter, table, item, item_no, value):
        """
        Add items to the order.

        Args:
            item: The menu item.
            item_no: The item number.
            value: The value (price) of the item.

        Returns:
            dict: The updated order list.
        """
        self.waiter  = waiter
        self.table = table
        self.menu_item = item
        self.num_order = item_no
```

```
        self.price = value

        self.order_list[waiter, table]=[item, item_no, value]
        return self.order_list

    def display_order_info(self):
        """
        Display information about the order.

        Returns:
            list: A list of strings containing information about each order
item.
        """
        return [f"OrderItem(number={num}, menu_item={item[0]},
num_item={item[1]}, price={item[2]})" for num, items in
self.order_list.items() for item in items]

    def total_price(self,file,waiter,table):
        """
        Calculate the total price of the order.

        Returns:
            float: The total price.

        """
        """self.order_list = file
        self.waiter = waiter
        self.table = table
        for key, value in file.items():
            if key[0] == waiter and key[1]==table:
                for item in valaue:
                    total=sum(item[1]*item[2])"""
        calculate_total = lambda items: sum(item[1] * item[2] for item in
items)

        # Find the relevant items in the order_list
        order_items = next((value for key, value in file.items() if key[0]
== waiter and key[1] == table), [])

        # Calculate the total using the lambda function
```

```
        total = calculate_total(order_items) if order_items else 0



        return total

    def __str__(self):
        """
        Generate a string representation"""




class Chef(Employee):
    """
    Represents a chef, inheriting from the Employee class.
    """

    def __init__(self):
        super().__init__()

    def display_chef_info(self):
        """
        Generate a formatted string representing the chef's information.

        Returns:
            str: A formatted string containing the chef's information.
        """
        return f"{super().display_employee_info()}"

    def __str__(self):
        """
        Generate a string representation of a chef.

        Returns:
            str: A string containing the chef's information.
        """
        return f'{str(super().display_employee_info())}'
class Customer(Person):
    """
```

```
    Represents a customer, inheriting from the Person class.
    """

    def set_customer_info(self, fullname, emirates_id, phone,
date_of_birth):
        """
        Set the information of a customer.

        Args:
            fullname (str): The full name of the customer.
            emirates_id (str): The Emirates ID of the customer.
            phone (str): The cell phone number of the customer.
            date_of_birth (str): The date of birth of the customer.
        """
        super().set_person_info(fullname, emirates_id, phone,
date_of_birth)

    def display_customer_info(self):
        """
        Generate a formatted string representing the customer's
information.

        Returns:
            str: A formatted string containing the customer's information.
        """
        return f"{super().display_person_info()}"

    def __str__(self):
        """
        Generate a string representation of a customer.

        Returns:
            str: A string containing the customer's information.
        """
        return f'{str(super().display_person_info())}'

class Receptionist(Employee):
    """
    Represents a receptionist, inheriting from the Employee class.
    """
```

```
    def __init__(self):
        super().__init__()

    def display_receptionist_info(self):
        """
        Generate a formatted string representing the receptionist's
information.

        Returns:
            str: A formatted string containing the receptionist's
information.
        """
        return f"{super().display_employee_info()}"

    def __str__(self):
        """
        Generate a string representation of a receptionist.

        Returns:
            str: A string containing the receptionist's information.
        """
        return f'{str(super().display_employee_info())}'

class Reservation:
    """
    Represents a reservation with information about the receptionist,
customer, table, time, date, and number of people.
    """

    def __init__(self):
      self.reservation_receptionist = Receptionist()
      self.customer_name = Customer()
      self.table = None
      self.time = None
      self.date = None
      self.num_people = None
      self.no_reservation = {}
      self.counter = count(start=1)
```

```
    def set_reservation_info(self, receptionist, customer, table, time,
date, num_people):
        """
        Set the information of a reservation.

        Args:
            receptionist (Receptionist): The receptionist handling the
reservation.
            customer (Customer): The customer making the reservation.
            table: The table reserved.
            time: The reservation time.
            date: The reservation date.
            num_people: The number of people for the reservation.
        """
        self.reservation_receptionist = receptionist
        self.customer_name = customer
        self.table = table
        self.time = time
        self.date = date if date is not None else None
        self.num_people = num_people

    def get_reservation_info(self):
        """
        Get the information of a reservation.

        Returns:
            tuple: A tuple containing reservation information.
        """
        return (
            self.reservation_receptionist,
            self.customer_name,
            self.table,
            self.time,
            self.date,
            self.num_people
        )

    def display_reservation_info(self):
        """
```

```
        Generate a formatted string representing the reservation's
information.

        Returns:
            str: A formatted string containing the reservation's
information.
        """
        return f"Reservation Information:\nTable: {self.table}\nTime:
{self.time}\nDate: {self.date}\nCustomer Name:
{self.customer_name.display_customer_info()}\nNumber of People:
{self.num_people}\nReceptionist: {self.reservation_receptionist}"

    def add_reservation(self, reservation):
        """
        Add a reservation to the list of reservations.

        Args:
            reservation: The reservation to add.
        """
        num = next(self.counter)
        self.no_reservation[num] = reservation

    def __str__(self):
        """
        Generate a string representation of a reservation.

        Returns:
            str: A string containing the reservation's information.
        """
        return f"Reservation Information:\nTable: {self.table}\nTime:
{self.time}\nDate: {self.date}\nCustomer Name:
{self.customer_name.display_person_info()}\nNumber of People:
{self.num_people}\nReceptionist:
{self.reservation_receptionist.display_person_info()}"




class Manager(Employee):
    """
```

```
    Represents a manager, inheriting from the Employee class.
    """

    def __init__(self):
        super().__init__()

    def display_manager_info(self):
        """
        Generate a formatted string representing the manager's information.

        Returns:
            str: A formatted string containing the manager's information.
        """
        return f"{super().display_employee_info()}"

    def __str__(self):
        """
        Generate a string representation of a manager.

        Returns:
            str: A string containing the manager's information.
        """
        return f'{str(super().display_employee_info())}'




class MenuItem:
    """
    Represents a menu item with information about its category, name,
description, and price.
    """

    def __init__(self):
        self.item_category = None
        self.item_name = None
        self.item_description = None
        self.item_price = None
```

```
    def set_menu_item(self, item_category, item_name, item_description,
item_price):
        """
        Set the information of a menu item.

        Args:
            item_category: The category of the menu item.
            item_name: The name of the menu item.
            item_description: The description of the menu item.
            item_price: The price of the menu item.
        """
        self.item_category = item_category
        self.item_name = item_name
        self.item_description = item_description
        self.item_price = item_price

    def get_item_info(self):
        """
        Get the information of a menu item.

        Returns:
            tuple: A tuple containing menu item information.
        """
        return self.item_category, self.item_name, self.item_description,
self.item_price

    def display_item_info(self):
        """
        Generate a formatted string representing the menu item's
information.

        Returns:
            str: A formatted string containing the menu item's information.
        """
        return f"Menu Item Information:\nCategory:
{self.item_category}\nName: {self.item_name}\nDescription:
{self.item_description}\nPrice: ${self.item_price:.2f}"

    def __str__(self):
        """
```

```
        Generate a string representation of a menu item.

        Returns:
            str: A string containing the menu item's information.
        """
        return f"Menu Item Information:\nCategory:
{self.item_category}\nName: {self.item_name}\nDescription:
{self.item_description}\nPrice: ${self.item_price:.2f}"


class Menu:
    """
    Represents a menu with a collection of menu items.
    """

    def __init__(self):
        self.menu_items = {}

    def add_menu_item(self, menu_item):
        """
        Add a menu item to the menu.

        Args:
            menu_item: The menu item to add.
        """
        self.menu_items[menu_item.item_name] = menu_item

    def display_menu(self):
        """
        Display the menu items.

        Returns:
            list: A list of strings containing information about each menu
item.
        """
        return [item.display_item_info() for item in
self.menu_items.values()]

    def __str__(self):
        """
```

```
        Generate a string representation of the menu.

        Returns:
            str: A string containing the menu's information.
        """
        return f"Menu:\n{tabulate([(item.item_category, item.item_name,
item.item_price) for item in self.menu_items.values()],
headers=['Category', 'Name', 'Price'])}"



class Branch:
    """
    Represents a branch of the restaurant with information about its
location, staff, and menu.
    """

    def __init__(self):
        self.branch_Name = None
        self.branch_Location = None
        self.branch_manager = Manager()
        self.branch_chef = Chef()
        self.branch_waiter = Waiter()
        self.branch_receptionist = Receptionist()
        self.branch_Menu = Menu()
        self.existing_branches_dict = {}

    def set_branch_info(self, name, location, manager, chef, waiter,
receptionist, menu):
        """
        Set the information of a branch, including name, location, manager,
chef, waiter, receptionist, and menu.

        Args:
            name (str): The name of the branch.
            location (str): The location of the branch.
            manager (Manager): The manager of the branch.
            chef (Chef): The chef of the branch.
            waiter (Waiter): The waiter of the branch.
            receptionist (Receptionist): The receptionist of the branch.
```

```
            menu (Menu): The menu of the branch.
        """
        try:
            # Try to find an existing branch with the same name and
location
            existing_branch = self.find_existing_branch(name, location)

            if existing_branch:
                # Branch with the same name and location exists, update
existing branch's staff information
                existing_branch["branch_Manager"] = manager
                existing_branch["branch_Chef"] = chef
                existing_branch["branch_Waiter"] = waiter
                existing_branch["branch_Receptionist"] = receptionist
            else:
                # Create a new branch if no existing branch is found
                new_branch = {
                    "branch_Name": name,
                    "branch_Location": location,
                    "branch_Manager": manager,
                    "branch_Chef": chef,
                    "branch_Waiter": waiter,
                    "branch_Receptionist": receptionist,
                    "branch_Menu": menu
                }

                # Add the new branch to the class-level
existing_branches_dict
                key = f"{name}_{location}"
                self.existing_branches_dict[key] = new_branch

        except Exception as e:
            print(f"An error occurred: {e}")

    def find_existing_branch(self, name, location):
        """
        Check if a branch with the same name and location already exists.

        Args:
            name (str): The name of the branch.
```

```
            location (str): The location of the branch.

        Returns:
            dict or None: The existing branch if found, otherwise None.
        """
        # Assuming you have a class-level dictionary of existing branches,
you can use a unique key for each branch
        key = f"{name}_{location}"
        return self.existing_branches_dict.get(key)

    def display_branch_info(self):
        """
        Display information for each branch using tabulate.
        """
        for key, branch_info in self.existing_branches_dict.items():
            table = [
                ["Property", "Value"],
                *[(property_name, property_value) for property_name,
property_value in branch_info.items()]
            ]
            table_str = tabulate(table, headers="firstrow",
tablefmt="pretty")
            print(f"\n{table_str}")

    def __str__(self):
        """
        Generate a string representation of a branch.

        Returns:
            str: A string containing the branch's information.
        """
        return f"Branch Information:\nManager:
{self.branch_manager.display_manager_info()}\nChef:
{self.branch_chef.display_chef_info()}\nWaiter:
{self.branch_waiter.display_waiter_info()}\nReceptionist:
{self.branch_receptionist.display_receptionist_info()}"

class Restaurant:

    def __init__(self):
```

```
        self.restu_No_Branches = {}
        self.rest_Owner = None
        self.restu_Name = None
        self.restu_Taxno = None


    def set_restaurant_info(self, owner, name, taxno):
        """
        Sets the information for the restaurant.

        Args:
            owner (str): The owner's name.
            name (str): The name of the restaurant.
            taxno (str): The tax number of the restaurant.

        Returns:
            tuple: A tuple containing owner, name, and tax number.
        """
        self.rest_Owner = owner
        self.restu_Name = name
        self.restu_Taxno = taxno
        self.counter= count(start=1)


    def add_restaurant_branch(self, branch):
        """
        Adds a branch to the restaurant.

        Args:
            branch (Branch): The branch to be added.
        """
        key = next(self.counter)
        self.restu_No_Branches[key] = branch
        return self.restu_No_Branches


    def display_restaurant_info(self):
        """
        Displays the information for the restaurant.
```

```
        Returns:
            str: A formatted string containing the restaurant information.
        """
        table = [
            ["Attribute", "Value"],
            ["Owner", self.rest_Owner],
            ["Restaurant Name", self.restu_Name],
            ["Tax Number", self.restu_Taxno],
            ["Number of Branches",self.restu_No_Branches]
        ]

        table_str = tabulate(table, headers="firstrow", tablefmt="pretty")

        return f"\n{table_str}"

    def __str__(self):
        """
        Generates a string representation of the restaurant.

        Returns:
            str: A string containing the restaurant information.
        """
        return "Restaurant Information:\n" + tabulate([["Owner",
restaurant.rest_Owner], ["Restaurant Name", restaurant.restu_Name], ["Tax
Number", restaurant.restu_Taxno], ["Number of Branches",
restaurant.restu_No_Branches]], headers="firstrow", tablefmt="pretty")

class ReservationForm:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Reservation Form")

        # Create an instance of the Reservation class
        self.reservation = Reservation()
        self.counter= count(start=1)



        self.receptionist_label = tk.Label(self.root, text="Receptionist:")
        self.receptionist_label.grid(row=0, column=0, padx=5, pady=5)
```

```
        self.receptionist_entry = tk.Entry(self.root)
        self.receptionist_entry.grid(row=0, column=1, padx=5, pady=5)

        self.fullname_label = tk.Label(self.root, text="Customer Full
Name:")
        self.fullname_label.grid(row=1, column=0, padx=5, pady=5)
        self.fullname_entry = tk.Entry(self.root)
        self.fullname_entry.grid(row=1, column=1, padx=5, pady=5)

        self.emiratesID_label = tk.Label(self.root, text="Customer Emirates
ID:")
        self.emiratesID_label.grid(row=2, column=0, padx=5, pady=5)
        self.emiratesID_entry = tk.Entry(self.root)
        self.emiratesID_entry.grid(row=2, column=1, padx=5, pady=5)

        self.phone_label = tk.Label(self.root, text="Customer Phone
number:")
        self.phone_label.grid(row=3, column=0, padx=5, pady=5)
        self.phone_entry = tk.Entry(self.root)
        self.phone_entry.grid(row=3, column=1, padx=5, pady=5)

        self.datebirth_label = tk.Label(self.root, text="Customer Date of
Birth:")
        self.datebirth_label.grid(row=4, column=0, padx=5, pady=5)
        self.datebirth_entry = tk.Entry(self.root)
        self.datebirth_entry.grid(row=4, column=1, padx=5, pady=5)

        self.table_label = tk.Label(self.root, text="Table Number:")
        self.table_label.grid(row=5, column=0, padx=5, pady=5)
        self.table_entry = tk.Entry(self.root)
        self.table_entry.grid(row=5, column=1, padx=5, pady=5)

        self.time_label = tk.Label(self.root, text="Time of Arrival:")
        self.time_label.grid(row=6, column=0, padx=5, pady=5)
        self.time_entry = tk.Entry(self.root)
        self.time_entry.grid(row=6, column=1, padx=5, pady=5)

        self.date_label = tk.Label(self.root, text="Date of Arrival:")
        self.date_label.grid(row=7, column=0, padx=5, pady=5)
        self.date_entry = tk.Entry(self.root)
```

```
        self.date_entry.grid(row=7, column=1, padx=5, pady=5)

        self.numpeople_label = tk.Label(self.root, text=" Number Of
People:")
        self.numpeople_label.grid(row=8, column=0, padx=5, pady=5)
        self.numpeople_entry = tk.Entry(self.root)
        self.numpeople_entry.grid(row=8, column=1, padx=5, pady=5)

        self.submit_button = tk.Button(self.root, text="Submit",
command=self.submit)
        self.submit_button.grid(row=9, column=0, pady=10)

        self.clear_button = tk.Button(self.root, text="Clear",
command=self.clear)
        self.clear_button.grid(row=9, column=1, pady=10)

        self.tree = ttk.Treeview(self.root, columns=('Receptionist',
'Customer', 'Table', 'Time', 'Date', 'NumPeople'))
        self.tree.grid(row=11, column=0, columnspan=2, pady=10)

        self.tree.heading('#0', text='ID')
        self.tree.heading('Receptionist', text='Receptionist')
        self.tree.heading('Customer', text='Customer')
        self.tree.heading('Table', text='Table')
        self.tree.heading('Time', text='Time')
        self.tree.heading('Date', text='Date')
        self.tree.heading('NumPeople', text='NumPeople')

        self.tree.scrollbar = ttk.Scrollbar(self.root, orient=tk.VERTICAL,
command=self.tree.yview)
        self.tree.configure(yscroll=self.tree.scrollbar.set)
        self.tree.scrollbar.grid(row=11, column=6, sticky='ns')

        # Hide the default ID column




        self.root.mainloop()
```

```
   def submit(self):
            # Get values from the entries
            receptionist = self.receptionist_entry.get()
            fullname = self.fullname_entry.get()
            emirates_id = self.emiratesID_entry.get()
            phone = self.phone_entry.get()
            dateofbirth = self.datebirth_entry.get()
            table = self.table_entry.get()
            time = self.time_entry.get()
            date = self.date_entry.get()
            num_people = self.numpeople_entry.get()

            # Set reservation information
            customer_info = f"{fullname} - {emirates_id} - {phone} -
{dateofbirth}"
            self.reservation.set_reservation_Info(receptionist,
customer_info, table, time, date, num_people)



            self.tree.insert("", "end", text=f"Reservation
{next(self.counter)} ", values=(receptionist, customer_info, table, time,
date, num_people))



            self.receptionist_entry.delete(0, tk.END)
            self.fullname_entry.delete(0, tk.END)
            self.emiratesID_entry.delete(0, tk.END)
            self.phone_entry.delete(0, tk.END)
            self.datebirth_entry.delete(0, tk.END)
            self.table_entry.delete(0, tk.END)
            self.time_entry.delete(0, tk.END)
            self.date_entry.delete(0, tk.END)
            self.numpeople_entry.delete(0, tk.END)

   def clear(self):

            self.receptionist_entry.delete(0, tk.END)
            self.fullname_entry.delete(0, tk.END)
            self.emiratesID_entry.delete(0, tk.END)
            self.phone_entry.delete(0, tk.END)
```

```
            self.datebirth_entry.delete(0, tk.END)
            self.table_entry.delete(0, tk.END)
            self.time_entry.delete(0, tk.END)
            self.date_entry.delete(0, tk.END)
            self.numpeople_entry.delete(0, tk.END)

class MenuItemWindow:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Menu Item Window")

        self.menu_item = MenuItem()

        # Entry widgets for setting menu item information
        self.category_entry = ttk.Entry(self.root, width=30)
        self.name_entry = ttk.Entry(self.root, width=30)
        self.description_entry = ttk.Entry(self.root, width=30)
        self.price_entry = ttk.Entry(self.root, width=30)

        # Labels for entry widgets
        ttk.Label(self.root, text="Category:").grid(row=0, column=0,
padx=5, pady=5)
        self.category_entry.grid(row=0, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Name:").grid(row=1, column=0, padx=5,
pady=5)
        self.name_entry.grid(row=1, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Description:").grid(row=2, column=0,
padx=5, pady=5)
        self.description_entry.grid(row=2, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Price:").grid(row=3, column=0, padx=5,
pady=5)
        self.price_entry.grid(row=3, column=1, padx=5, pady=5)

        # Buttons
        self.set_info_button = ttk.Button(self.root, text="Set Menu Item
Info", command=self.set_menu_item_info)
        self.set_info_button.grid(row=4, column=0, columnspan=2, pady=10)
```

```
        self.clear_button = ttk.Button(self.root, text="Clear",
command=self.clear_entries)
        self.clear_button.grid(row=5, column=0, pady=10)

        self.delete_button = ttk.Button(self.root, text="Delete",
command=self.delete_item)
        self.delete_button.grid(row=5, column=1, pady=10)

        # Treeview to display menu item information
        self.tree = ttk.Treeview(self.root, columns=("Category", "Name",
"Description", "Price"), show="headings")
        self.tree.heading("Category", text="Category")
        self.tree.heading("Name", text="Name")
        self.tree.heading("Description", text="Description")
        self.tree.heading("Price", text="Price")
        self.tree.grid(row=6, column=0, columnspan=2, pady=10)

        self.root.mainloop()

    def set_menu_item_info(self):
        category = self.category_entry.get()
        name = self.name_entry.get()
        description = self.description_entry.get()
        price = self.price_entry.get()

        self.menu_item.set_menu_item(category, name, description, price)
        self.display_menu_item_info()

    def display_menu_item_info(self):
        info = self.menu_item.get_item_info()
        self.tree.insert("", "end", values=info)

    def clear_entries(self):
        # Clear entry widgets
        self.category_entry.delete(0, tk.END)
        self.name_entry.delete(0, tk.END)
        self.description_entry.delete(0, tk.END)
        self.price_entry.delete(0, tk.END)
```

```
    def delete_item(self):
        # Get the selected item and delete it from the Treeview
        selected_item = self.tree.selection()
        if selected_item:
            self.tree.delete(selected_item)

class OrderWindow:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Order Window")

        self.order = Order()

        # Entry widgets for setting order information
        self.waiter_entry = ttk.Entry(self.root, width=30)
        self.table_entry = ttk.Entry(self.root, width=30)
        self.menu_item_entry = ttk.Entry(self.root, width=30)
        self.num_order_entry = ttk.Entry(self.root, width=30)
        self.price_entry = ttk.Entry(self.root, width=30)

        # Labels for entry widgets
        ttk.Label(self.root, text="Waiter:").grid(row=0, column=0, padx=5,
pady=5)
        self.waiter_entry.grid(row=0, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Table:").grid(row=1, column=0, padx=5,
pady=5)
        self.table_entry.grid(row=1, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Menu Item:").grid(row=2, column=0,
padx=5, pady=5)
        self.menu_item_entry.grid(row=2, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Number of Orders:").grid(row=3,
column=0, padx=5, pady=5)
        self.num_order_entry.grid(row=3, column=1, padx=5, pady=5)

        ttk.Label(self.root, text="Price:").grid(row=4, column=0, padx=5,
pady=5)
        self.price_entry.grid(row=4, column=1, padx=5, pady=5)
```

```
        # Buttons
        self.add_item_button = ttk.Button(self.root, text="Add to Order",
command=self.add_to_order)
        self.add_item_button.grid(row=5, column=0, columnspan=2, pady=10)

        self.display_total_button = ttk.Button(self.root, text="Display
Total", command=self.display_total_price)
        self.display_total_button.grid(row=6, column=0, columnspan=2,
pady=10)

        # Treeview widget to display order information
        self.tree = ttk.Treeview(self.root, columns=("Waiter", "Table",
"Menu Item", "Number of Orders", "Price"))
        self.tree.heading("Waiter", text="Waiter")
        self.tree.heading("Table", text="Table")
        self.tree.heading("Menu Item", text="Menu Item")
        self.tree.heading("Number of Orders", text="Number of Orders")
        self.tree.heading("Price", text="Price")
        self.tree.grid(row=7, column=0, columnspan=2, pady=10)

        self.root.mainloop()

    def add_to_order(self):
        waiter = self.waiter_entry.get()
        table = self.table_entry.get()
        menu_item = self.menu_item_entry.get()
        num_order = self.num_order_entry.get()
        price = self.price_entry.get()


        self.tree.insert("", tk.END, values=(waiter, table,
menu_item,num_order, price))
        return self.order.add_items( waiter, table, menu_item, num_order,
price)



    def display_total_price(self):
        waiter = self.waiter_entry.get()
```

```
        table = self.table_entry.get()
        menu_item = self.menu_item_entry.get()
        num_order = self.num_order_entry.get()
        price = self.price_entry.get()
        file={}
        file[(waiter, table)]=[menu_item, num_order, price]

        file=self.add_to_order
```

```
Manager Information:
Full Name: John Manager, Emirates ID: 123456789, Phone: 555-1234, Date Of Birth: 1990-01-01, Department: Manager, Room: Main Office

Chef Information:
Full Name: Chef Gordon, Emirates ID: 987654321, Phone: 555-5678, Date Of Birth: 1980-02-02, Department: Chef, Room: Kitchen

Waiter Information:
Full Name: Alice Waiter, Emirates ID: 111223344, Phone: 555-4321, Date Of Birth: 1995-05-05, Department: Waiter, Room: Dining Area

Receptionist Information:
Full Name: Rebecca Receptionist, Emirates ID: 444556677, Phone: 555-8765, Date Of Birth: 1992-03-03, Department: Receptionist, Room: Front Desk

Customer Information:
Full Name: Bob Customer, Emirates ID: 998877665, Phone: 555-9999, Date Of Birth: 1988-04-04

Reservation Information:
Reservation Information:
Table: Table 1
Time: 18:00
Date: 2023-01-01
Customer Name: Full Name: Bob Customer, Emirates ID: 998877665, Phone: 555-9999, Date Of Birth: 1988-04-04
Number of People: 4
Receptionist: Full Name: Rebecca Receptionist, Emirates ID: 444556677, Phone: 555-8765, Date Of Birth: 1992-03-03, Department: Receptionist, Room: Front Desk

Menu Information:
Menu:
Category     Name              Price
-----------  ----------------  -------
Main Course  Grilled Chicken    12.99
```