Управление рестораном

Описание задачи:
Создайте программу для управления рестораном. В ресторане есть столики, официанты и система заказов.

Требования:

1. Создайте классы для столиков, официантов и ресторана.
2. Реализуйте возможность бронирования столиков и назначения официантов к столикам.
3. Реализуйте систему принятия заказов от клиентов.
4. Храните информацию о заказах и обрабатывайте их.

Решение:

```
class Table:
    def __init__(self, table_number, seats):
        self.table_number = table_number
        self.seats = seats
        self.is_reserved = False
        self.orders = []

    def reserve(self):
        if not self.is_reserved:
            self.is_reserved = True
            print(f"Table {self.table_number} reserved.")
        else:
            print(f"Table {self.table_number} is already reserved.")

class Waiter:
    def __init__(self, name):
        self.name = name
        self.tables = []

    def assign_table(self, table):
        self.tables.append(table)
        print(f"{self.name} is assigned to table {table.table_number}.")

class Order:
    def __init__(self, table, waiter, items):
        self.table = table
        self.waiter = waiter
        self.items = items
        self.is_completed = False

    def complete_order(self):
        self.is_completed = True
        print(f"Order for table {self.table.table_number} completed.")

class Restaurant:
    def __init__(self):
        self.tables = []
        self.waiters = []

    def add_table(self, table):
        self.tables.append(table)

    def hire_waiter(self, waiter):
        self.waiters.append(waiter)

    def reserve_table(self, table_number):
        table = next((t for t in self.tables if t.table_number == table_number), None)
        if table:
```

```
            table.reserve()
        else:
            print(f"Table {table_number} not found.")

    def assign_waiter_to_table(self, waiter_name, table_number):
        waiter = next((w for w in self.waiters if w.name == waiter_name), None)
        table = next((t for t in self.tables if t.table_number == table_number), None)

        if waiter and table:
            waiter.assign_table(table)
        else:
            print("Waiter or Table not found.")

    def take_order(self, table_number, items):
        table = next((t for t in self.tables if t.table_number == table_number), None)
        if table and table.is_reserved:
            waiter = next((w for w in self.waiters if table in w.tables), None)
            if waiter:
                order = Order(table, waiter, items)
                table.orders.append(order)
                print(f"Order taken for table {table_number}.")
            else:
                print(f"No waiter assigned to table {table_number}.")
        else:
            print(f"Table {table_number} is not reserved.")

# Пример использования
restaurant = Restaurant()

table1 = Table(1, 4)
table2 = Table(2, 2)
waiter1 = Waiter("John")
waiter2 = Waiter("Alice")

restaurant.add_table(table1)
restaurant.add_table(table2)
restaurant.hire_waiter(waiter1)
restaurant.hire_waiter(waiter2)

restaurant.reserve_table(1)
restaurant.assign_waiter_to_table("John", 1)
restaurant.take_order(1, ["Pizza", "Salad"])
```

Описание решения:

1. **Класс** `Table`: Представляет столик в ресторане, содержит номер столика, количество мест, статус бронирования и список заказов.
2. **Класс** `Waiter`: Представляет официанта, содержит имя и список столиков, к которым он назначен.
3. **Класс** `Order`: Представляет заказ, содержит ссылку на столик, официанта и список заказанных блюд.
4. **Класс** `Restaurant`: Управляет списками столиков и официантов, реализует методы для бронирования столиков, назначения официантов и принятия заказов