```python
#import ENUM
from enum import Enum
# define an enum for the names of different services
class ServiceName(Enum):
    CLEANING = 1
    IMPLANTS = 2
    CROWNS = 3
    FILLINGS = 4
    MORE = 5

# define an enum for the job titles
class job_title(Enum):
    Manager = 1
    Receptionist = 2
    Hygienist = 3
    Dentist = 4

#Aggregation relationship between the dental company and its branches
class DentalCompany:
    # Constructor method to initialize attributes
    def __init__(self, companyName, owner, email, website):
        self.companyName = companyName
        self.owner = owner
        self.email = email
        self.website = website
        self.branches = []  # Empty list to store branches

    # Method to add a new branch to the company's list
    def add_branch(self, branch):
        self.branches.append(branch)

class Branch:
    # Constructor to initialize attributes
    def __init__(self, address, phoneNumber, manager):
        self.address = address
        self.phoneNumber = phoneNumber
        self.manager = manager
        # initialize an empty list for the branch's services
        self.services = []  # list of Service objects
        self.appointments = [] # list of appointment objects
        self.staff = []  # list of Staff objects
        self.patients = [] # list of Patient objects


    # method to add a new service to the branch's list
    def add_service(self, service):
        self.services.append(service)

    # method to add a new appointment to the branch's list
    def add_appointment(self, appointment):
        self.appointments.append(appointment)

    # method to add a new staff member to the branch's list
    def add_staff(self, staff):
        self.staff.append(staff)

    # method to add a new patient to the branch's list
```

```
    def add_patient(self, patient):
        self.patients.append(patient)


class Service:
    # Constructor to initialize attributes
    def __init__(self, name, cost):
        self.name = name
        self.cost = cost

# define a class for a patient
class Patient:
    # Initialize common attributes
    def __init__(self, first_name, last_name, age, address, phone_number):
        # Initialize common attributes for all people
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.address = address
        self.phone_number = phone_number

# inheritance
class Person:
    # Constructor method to initialize attributes
    def __init__(self, first_name, last_name, age, address, phone_number):
        # Initialize common attributes for all people
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.address = address
        self.phone_number = phone_number

# inherit from Person class
class Staff(Person):
    # Constructor method to initialize attributes
    def __init__(self, first_name, last_name, age, address, phone_number,
job_title, staff_id):
        # Call the constructor of the parent class (Person) to initialize
common attributes
        # using super function
        super().__init__(first_name, last_name, age, address, phone_number)
        # Initialize attributes specific to staff members
        self.job_title = job_title
        self.staff_id = staff_id

# define a class for a dental appointment
class Appointment:
    # Constructor to initialize attributes
    def __init__(self, patient_name, date, time,patient, staff):
        self.patient_name = patient_name
        self.date = date
        self.time = time
        # The patient object associated with the appointment
        self.patient = patient
        # The staff member associated with the appointment
        self.staff = staff
```

```
class PaymentReceipt:
    # Initialize instance variables for the given arguments
    def __init__(self, patient, service):
        self.patient = patient
        self.service = service

    def get_total_cost(self):
        total_cost = 0
        # Loop through the list of services
        for sr in self.service:
            # Calculate the total cost by adding the cost of each service
            total_cost += sr.cost
        return total_cost

    def get_vat(self):
        # Calculate the VAT (Value Added Tax) by multiplying the total cost
by 0.05
        return self.get_total_cost() * 0.05

    def get_total_cost_with_vat(self):
        # Calculate the total cost including VAT
        return self.get_total_cost() + self.get_vat()

    def display_receipt(self):
        # Print the receipt
        print("Payment Receipt:")
        print("Patient Name: " + self.patient.first_name + " " +
self.patient.last_name)
        print("Service(s): ")
        for sr in self.service:
            print(sr.service.name + ": AED" + str(sr.service.cost))
        print("Total Cost: AED" + str(self.get_total_cost()))
        print("VAT (5%): AED" + str(self.get_vat()))
        print("Total Cost (with VAT): AED" +
str(self.get_total_cost_with_vat()))
.get_total_cost() + self.get_vat()
```

Test cases:

First test case:
a. the addition of branches to the dental company.

The code is provided here to test the code. It creates a dental company object, and two branch objects add the branch objects to the dental company and print out the branch information.

I used the assert function here to check that the condition is TRUE.

```
# Create a dental company object
dental_company = DentalCompany("My Dental", "Hessa", "MyDental797@gmail.com",
"www.mydental.com")
```

```
# Create two branch objects
branch1 = Branch("Ajman-12", "050789876", "Alice")
branch2 = Branch("Dubai-11", "050789777", "Bob")

# Add the branches to the dental company
dental_company.add_branch(branch1)
dental_company.add_branch(branch2)

print("There are,",(len(dental_company.branches)),"dental company branches")
# Assert that the dental company has the correct number of branches
assert len(dental_company.branches) == 2

print("First branch info")
print(dental_company.branches[0].address)
print(dental_company.branches[0].phoneNumber)
print(dental_company.branches[0].manager)

print("Second branch info")
print(dental_company.branches[1].address)
print(dental_company.branches[1].phoneNumber)
print(dental_company.branches[1].manager)
# Assert that the branch information was added correctly
assert dental_company.branches[0].address == "Ajman-12"
assert dental_company.branches[0].phoneNumber == "050789876"
assert dental_company.branches[0].manager == "Alice"

assert dental_company.branches[1].address == "Dubai-11"
assert dental_company.branches[1].phoneNumber == "050789777"
assert dental_company.branches[1].manager == "Bob"
```

Second test case:

b. the addition of dental services, staff, and patients to a branch.

```
# Testing adding services, staff, and patients to a branch

branch1 = Branch("Ajman-12", "050789876", "Alice")

service1 = Service(ServiceName.CLEANING, 150)
staff1 = Staff("Ali", "Rashed", 29, "Ajman123", "050763452",
job_title.Dentist, "12345A")
patient1 = Patient("John", "Smith", 22, "Dubai98-", "0501234567")
branch1.add_service(service1)
branch1.add_staff(staff1)
branch1.add_patient(patient1)

assert len(branch1.services) == 1
assert len(branch1.staff) == 1
assert len(branch1.patients) == 1

print("Adding services, staff, and patients to a branch test passed.")
```

Third test case:

c. the addition of patients booking appointments.

```
staff1 = Staff("Ali", "Rashed", 29, "Ajman123", "050763452",
job_title.Dentist, "12345A")
branch = Branch("123 Main St", "555-1234", staff1) # create a new branch

# create a new patient and add them to the branch's list of patients
patient = Patient("Maha", "Saeed", 33, "789AlJurf", "05067888")
branch.add_patient(patient)

# create a new appointment with the patient and a staff member and add it to
the branch's list of appointments
staff2 = Staff("Lisa", "Smith", 35, "321aj", "05035678", job_title.Hygienist,
"456b")
appointment = Appointment("Hessa", "2023-05-01", "10:00 AM", patient, staff2)
branch.add_appointment(appointment)
assert len(branch.patients) == 1
assert len(branch.appointments) == 1
assert branch.appointments[0].staff == staff2
assert branch.appointments[0].patient == patient
```

Fourth test case:

Printing the total cost with vat and without vat.

```
# create a new patient
patient1 = Patient("Maha", "Saeed", 33, "789AlJurf", "05067888")

# create some services for the branch
cleaning = Service(ServiceName.CLEANING, 100)
fillings = Service(ServiceName.FILLINGS, 150)
implants = Service(ServiceName.IMPLANTS, 1500)

# create a branch
branch1 = Branch("1234aj", "055555555", Staff("Maitha", "Ahmed", 25, "123sh",
"052345678", job_title.Dentist, "D123"))

# add services to the branch
branch1.add_service(cleaning)
branch1.add_service(fillings)
branch1.add_service(implants)

# add the patient to the branch
branch1.add_patient(patient1)


# create a payment receipt for the patient's services
receipt1 = PaymentReceipt(patient1, [cleaning, fillings, implants])

# print the total cost and total cost with VAT for the patient's services
print("Total cost:", receipt1.get_total_cost(), "AED")
print("Total cost with VAT:", receipt1.get_total_cost_with_vat(), "AED")
```

Or

Testing the code using the assert function:

```
# create a payment receipt for the patient's services
receipt1 = PaymentReceipt(patient1, [cleaning, fillings, implants])

# print the total cost and total cost with VAT for the patient's services
print("Total cost: $", receipt1.get_total_cost())
print("Total cost with VAT: $", receipt1.get_total_cost_with_vat())


# Create a Patient object
patient = Patient("Maha", "Saeed", 33, "789AlJurf", "05067888")

# Create a Service object
service = Service(ServiceName.IMPLANTS, 1500)

# Create a PaymentReceipt object
receipt = PaymentReceipt(patient, [service])

# Test the get_total_cost() method
assert receipt.get_total_cost() == 1500

# Test the get_vat() method
assert receipt.get_vat() == 75

# Test the get_total_cost_with_vat() method
assert receipt.get_total_cost_with_vat() == 1575
```