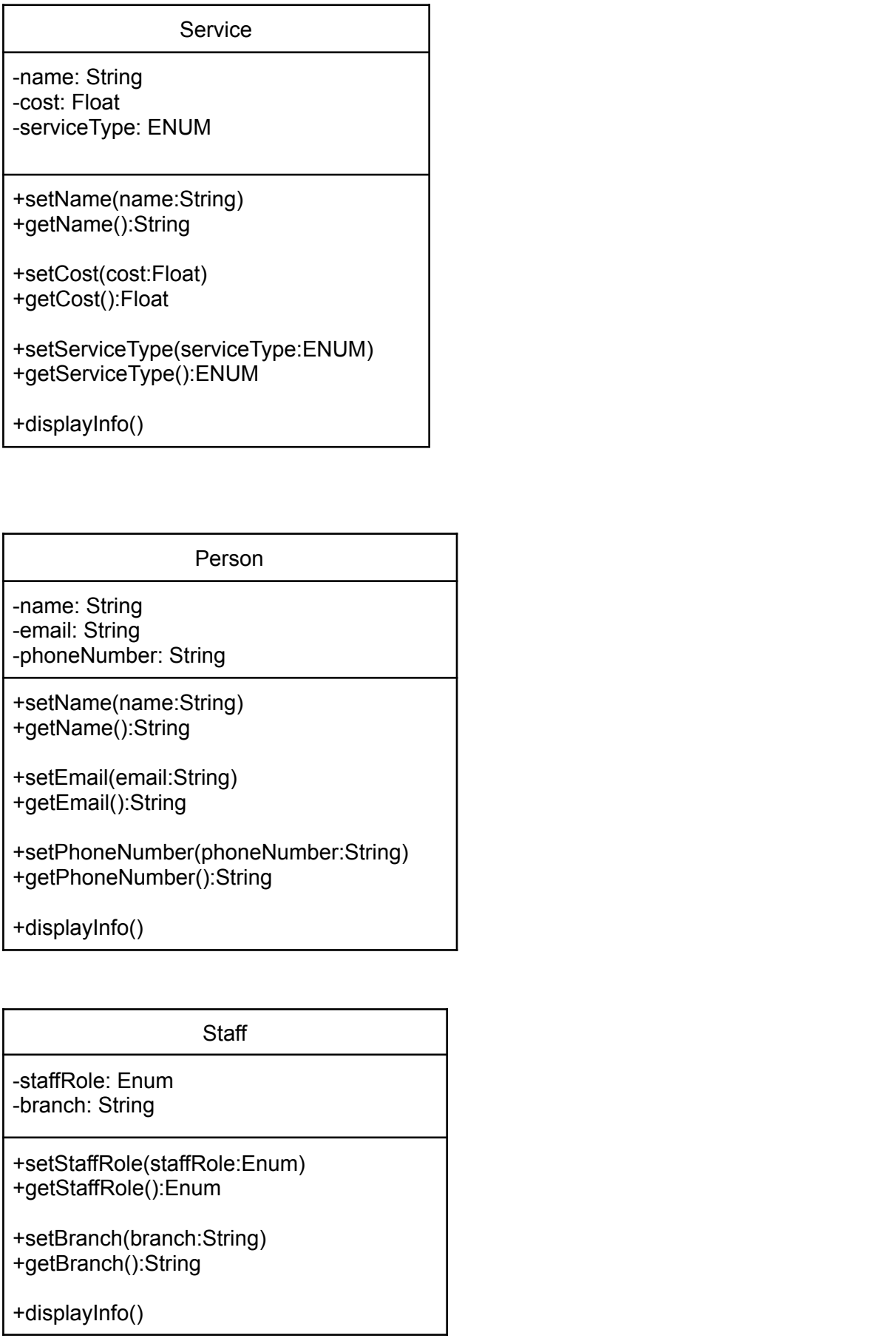
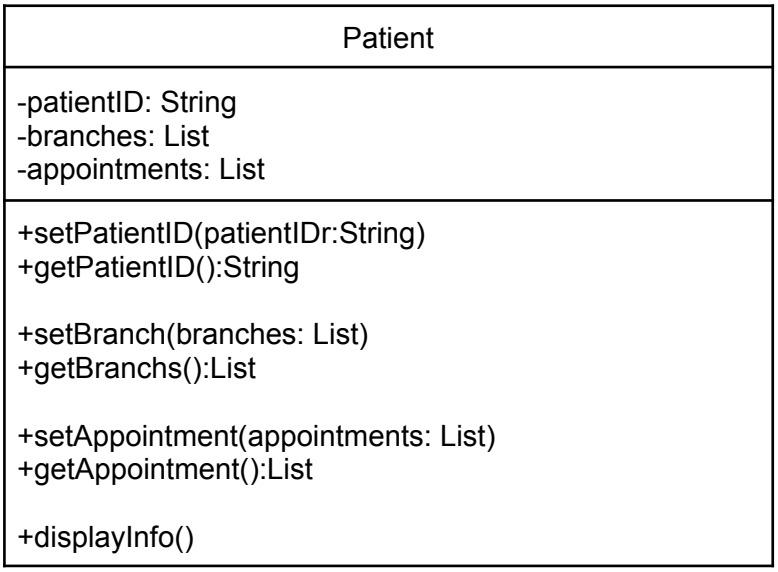
Assignment 2: Software Implementation

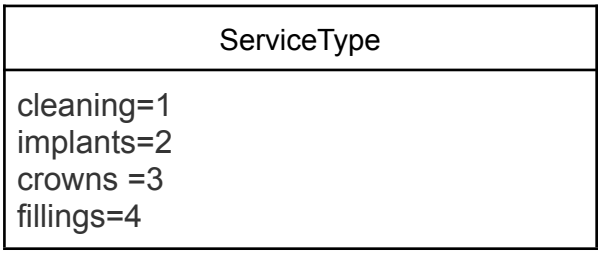
1. **UML Class** Diagram and Description

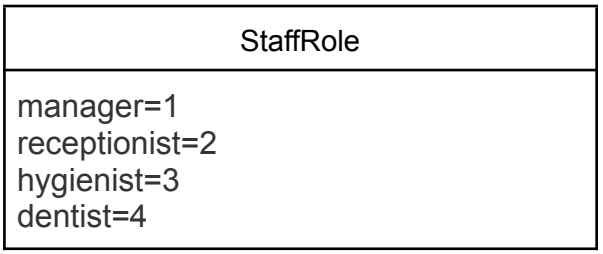
DentalCompany
-companyName: String -branches: List
+setCompanyName(companyName:String) +getCompanyName():String +setBranch(branches:List) +getBranch():List +displayInfo()

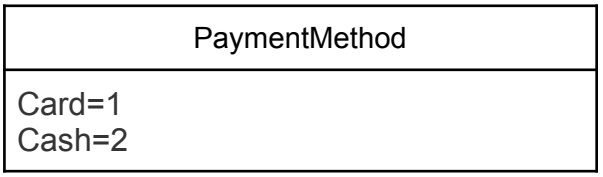
DentalBranch
-address: String -phoneNumber: String -manager: String -services: List -staff: List -patients: List
+setAddress(address:String) +getAddress():String +setPhoneNumber(phoneNumber:String) +getPhoneNumber():String +setManager(manager:String) +getManager():String +setStaff(staff: List) +getStaff():List +setService(Service:List) +getService():List +setPatient(patient:List) +getPatient():List +displayInfo()

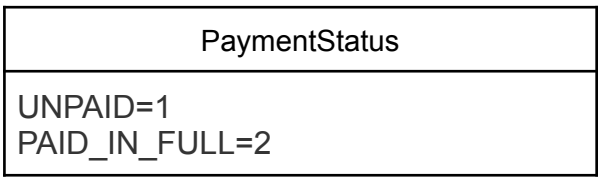

Service
-name: String
-cost: Float
-serviceType: ENUM
+setName(name:String)
+getName():String
+setCost(cost:Float)
+getCost():Float
+setServiceType(serviceType:ENUM)
+getServiceType():ENUM
+displayInfo()
Person
-name: String
-email: String
-phoneNumber: String
+setName(name:String)
+getName():String
+setEmail(email:String)
+getEmail():String
+setPhoneNumber(phoneNumber:String)
+getPhoneNumber():String
+displayInfo()
Staff
-staffRole: Enum
-branch: String
+setStaffRole(staffRole:Enum)
+getStaffRole():Enum
+setBranch(branch:String)
+getBranch():String
+displayInfo()

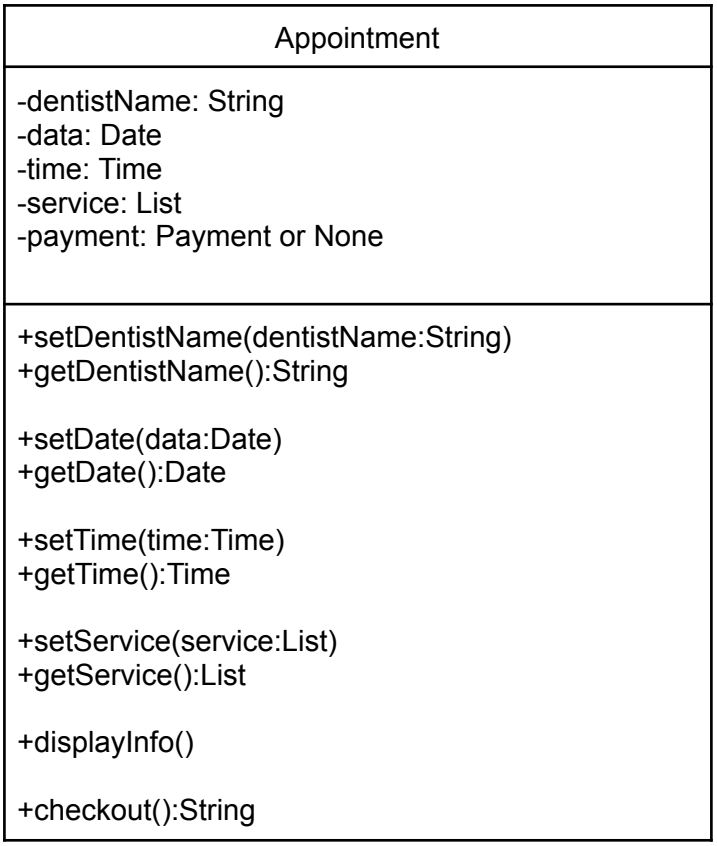

Patient
-patientID: String
-branches: List
-appointments: List
+setPatientID(patientIDr:String)
+getPatientID():String
+setBranch(branches: List)
+getBranchs():List
+setAppointment(appointments: List)
+getAppointment():List
+displayInfo()

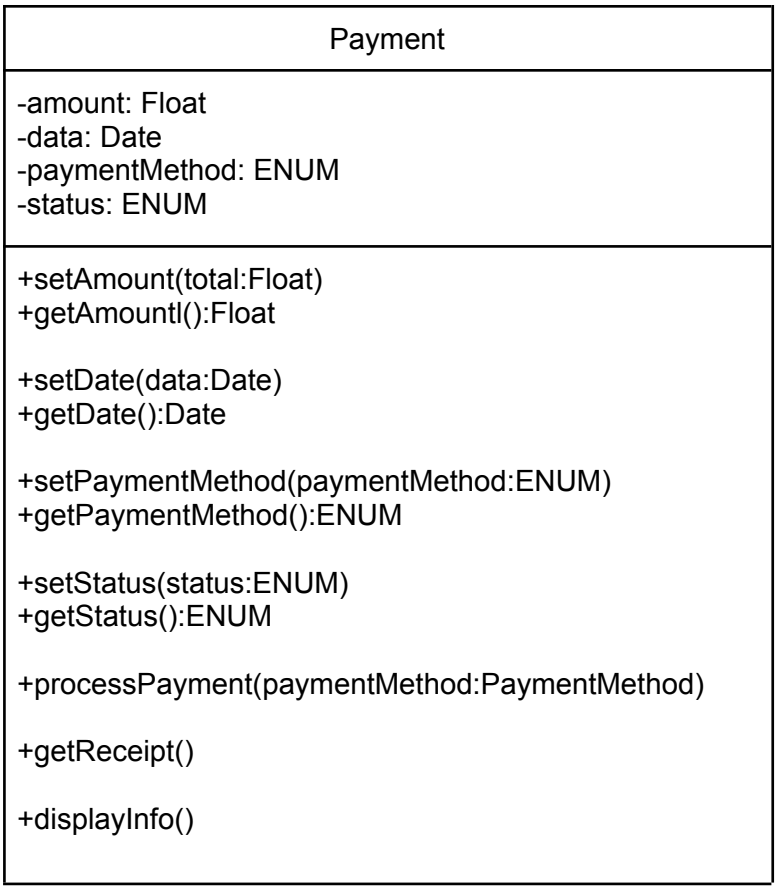

ServiceType
cleaning=1
implants=2
crowns =3
fillings=4

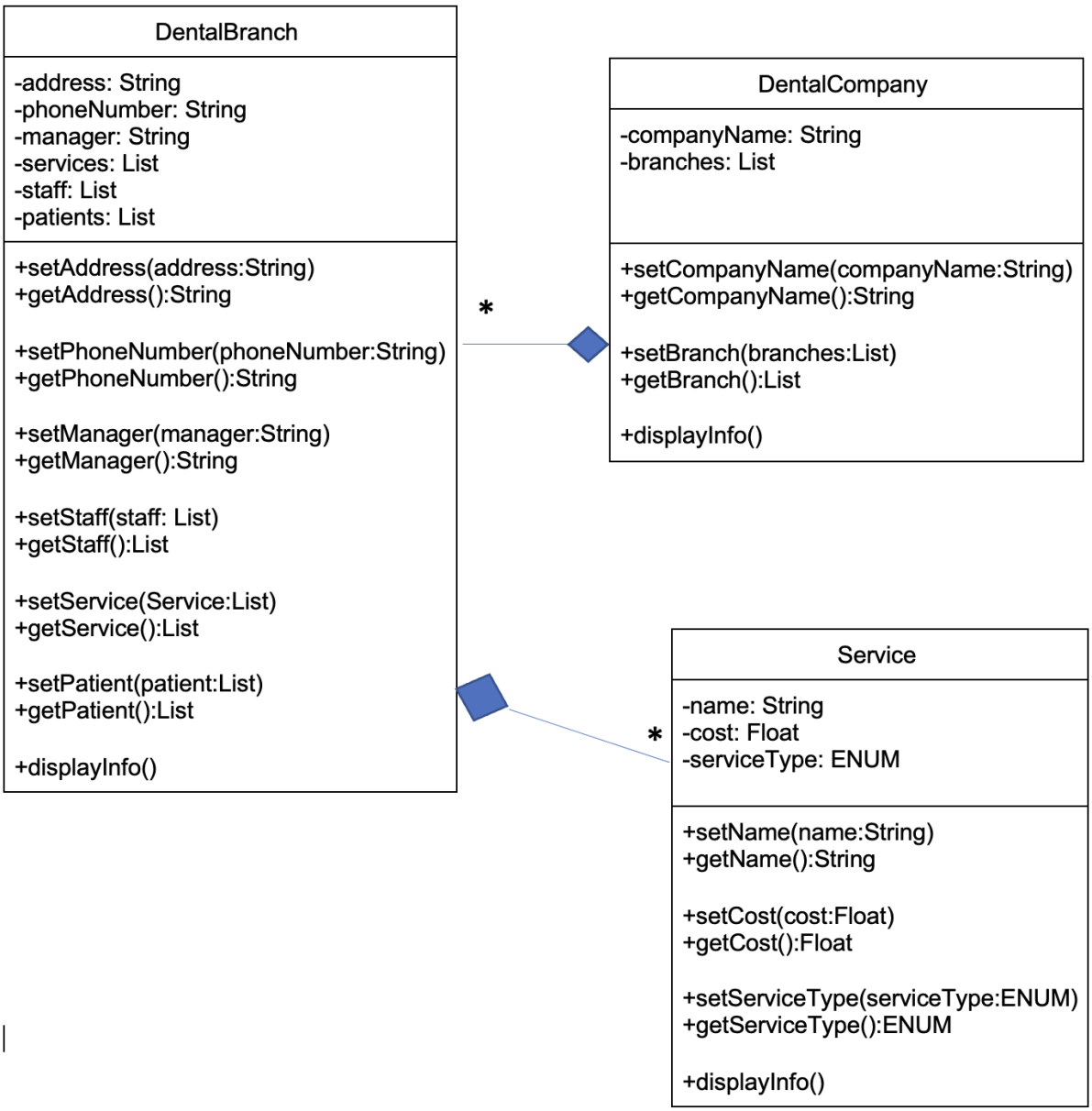

StaffRole
manager=1
receptionist=2
hygienist=3
dentist=4

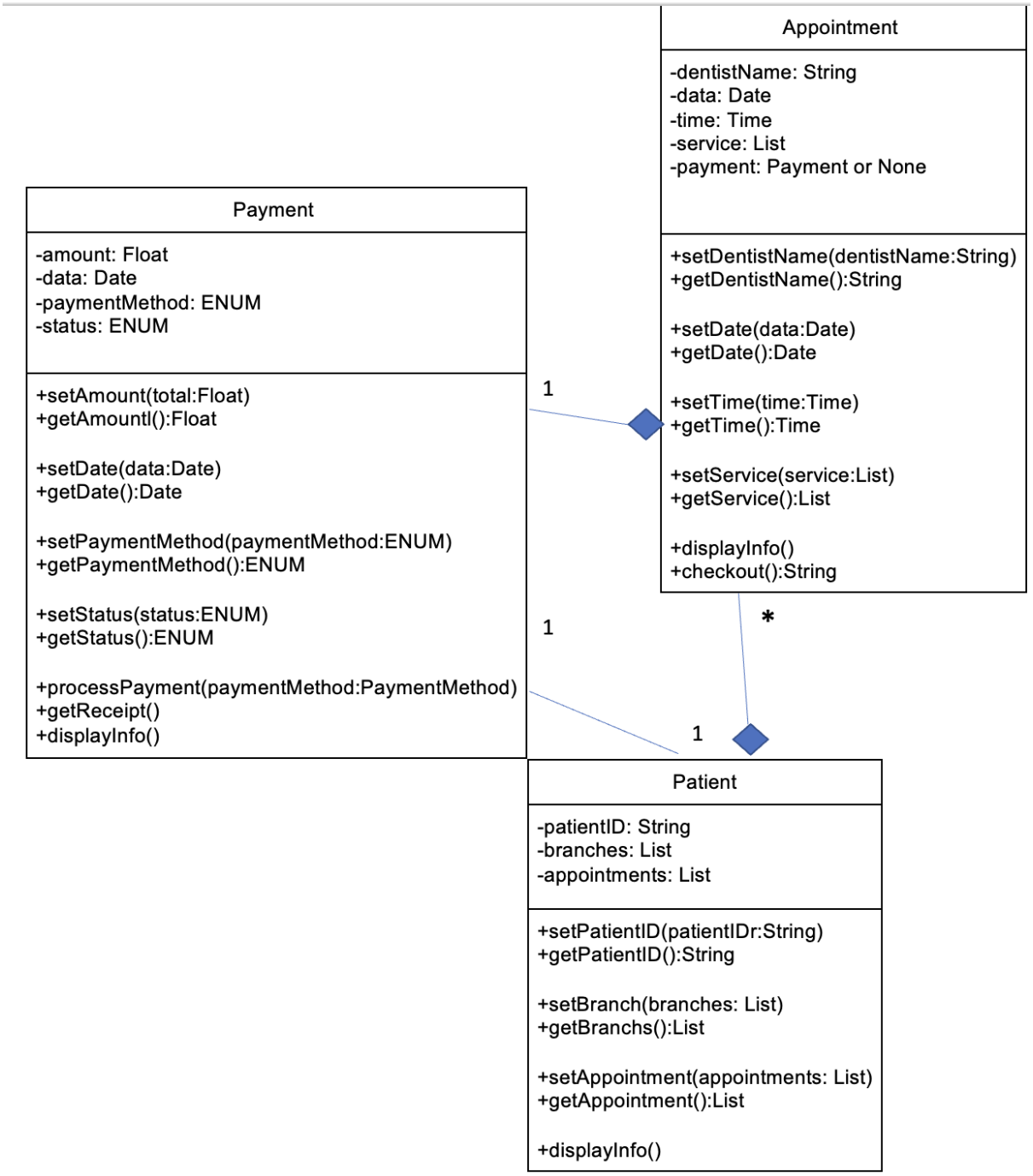

PaymentMethod
Card=1
Cash=2

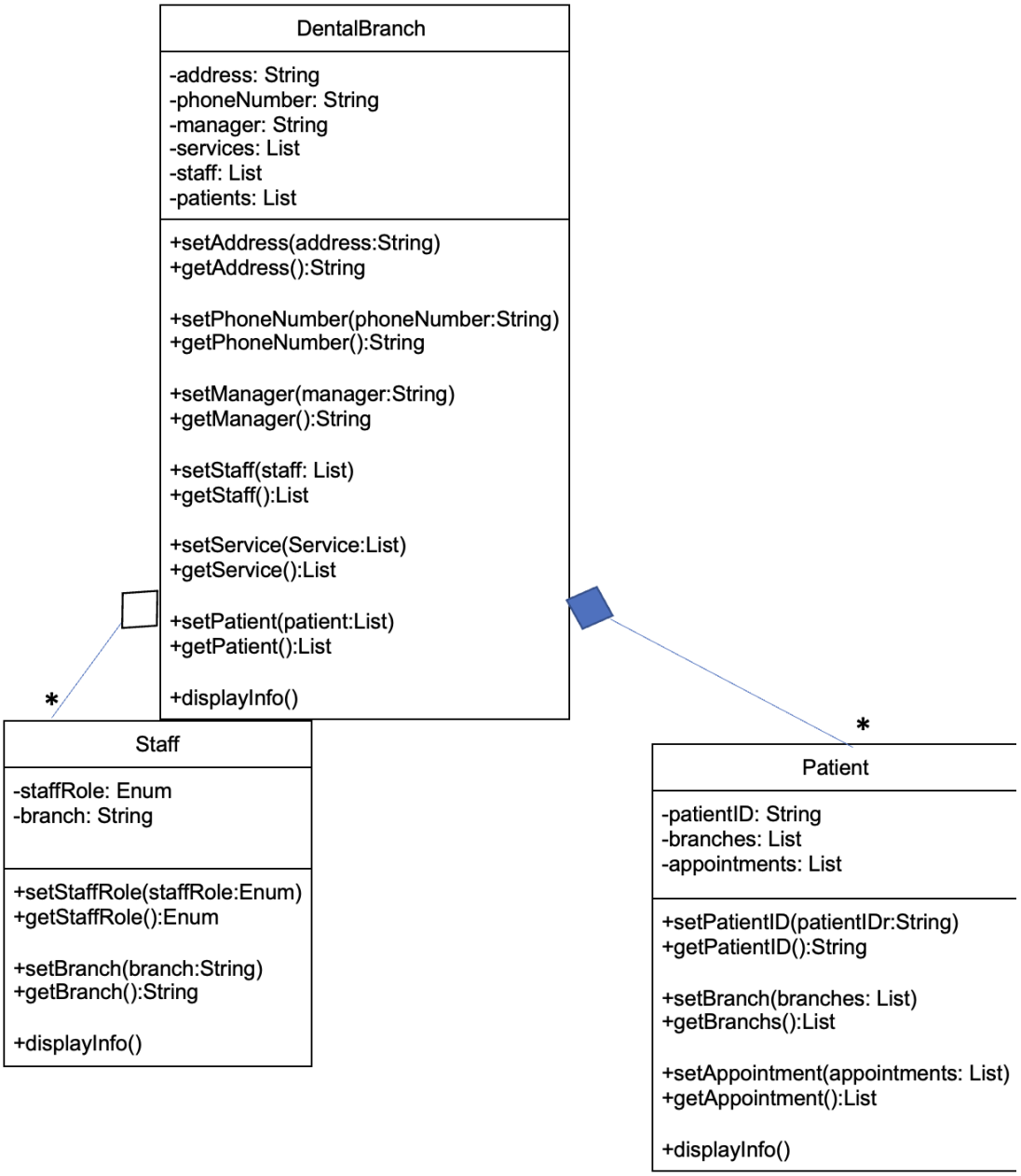

PaymentStatus
UNPAID=1
PAID_IN_FULL=2

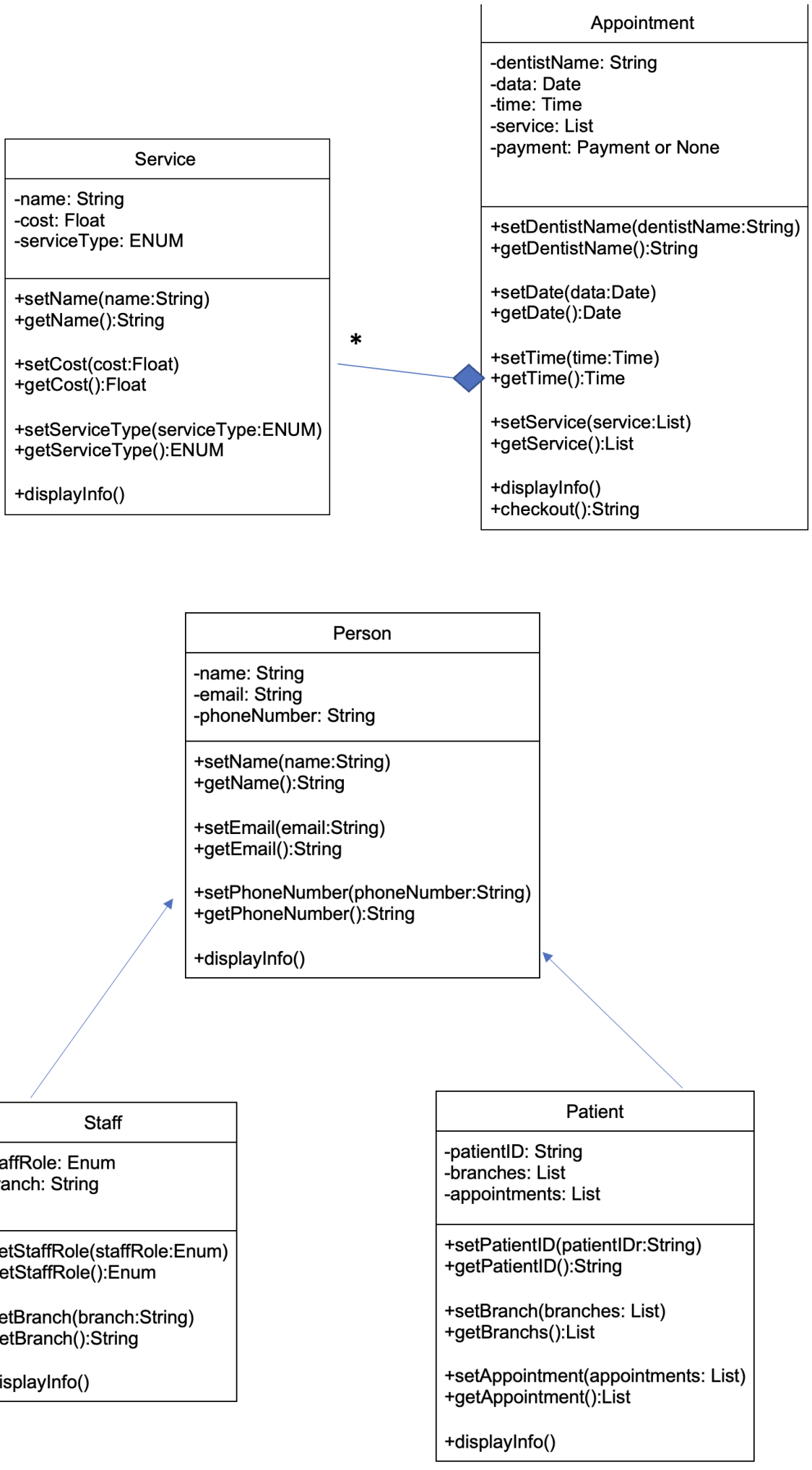

Appointment
-dentistName: String
-data: Date
-time: Time
-service: List
-payment: Payment or None
+setDentistName(dentistName:String)
+getDentistName():String
+setDate(data:Date)
+getDate():Date
+setTime(time:Time)
+getTime():Time
+setService(service:List)
+getService():List
+displayInfo()
+checkout():String


Payment
-amount: Float
-data: Date
-paymentMethod: ENUM
-status: ENUM
+setAmount(total:Float)
+getAmountl():Float
+setDate(data:Date)
+getDate():Date
+setPaymentMethod(paymentMethod:ENUM)
+getPaymentMethod():ENUM
+setStatus(status:ENUM)
+getStatus():ENUM
+processPayment(paymentMethod:PaymentMethod)
+getReceipt()
+displayInfo()


DentalBranch
-address: String
-phoneNumber: String
-manager: String
-services: List
-staff: List
-patients: List
+setAddress(address:String)
+getAddress():String
+setPhoneNumber(phoneNumber:String)
+getPhoneNumber():String
+setManager(manager:String)
+getManager():String
+setStaff(staff: List)
+getStaff():List
+setService(Service:List)
+getService():List
+setPatient(patient:List)
+getPatient():List
+displayInfo()
*
DentalCompany
-companyName: String
-branches: List
+setCompanyName(companyName:String)
+getCompanyName():String
+setBranch(branches:List)
+getBranch():List
+displayInfo()
*
Service
-name: String
-cost: Float
-serviceType: ENUM
+setName(name:String)
+getName():String
+setCost(cost:Float)
+getCost():Float
+setServiceType(serviceType:ENUM)
+getServiceType():ENUM
+displayInfo()


Appointment
-dentistName: String
-data: Date
-time: Time
-service: List
-payment: Payment or None
+setDentistName(dentistName:String)
+getDentistName():String
+setDate(data:Date)
+getDate():Date
+setTime(time:Time)
+getTime():Time
+setService(service:List)
+getService():List
+displayInfo()
+checkout():String
Payment
-amount: Float
-data: Date
-paymentMethod: ENUM
-status: ENUM
+setAmount(total:Float)
+getAmountl():Float
+setDate(data:Date)
+getDate():Date
+setPaymentMethod(paymentMethod:ENUM)
+getPaymentMethod():ENUM
+setStatus(status:ENUM)
+getStatus():ENUM
+processPayment(paymentMethod:PaymentMethod)
+getReceipt()
+displayInfo()
1
1
*
1
Patient
-patientID: String
-branches: List
-appointments: List
+setPatientID(patientIDr:String)
+getPatientID():String
+setBranch(branches: List)
+getBranchs():List
+setAppointment(appointments: List)
+getAppointment():List
+displayInfo()


DentalBranch
-address: String
-phoneNumber: String
-manager: String
-services: List
-staff: List
-patients: List
+setAddress(address:String)
+getAddress():String
+setPhoneNumber(phoneNumber:String)
+getPhoneNumber():String
+setManager(manager:String)
+getManager():String
+setStaff(staff: List)
+getStaff():List
+setService(Service:List)
+getService():List
+setPatient(patient:List)
+getPatient():List
+displayInfo()
*
Staff
-staffRole: Enum
-branch: String
+setStaffRole(staffRole:Enum)
+getStaffRole():Enum
+setBranch(branch:String)
+getBranch():String
+displayInfo()
*
Patient
-patientID: String
-branches: List
-appointments: List
+setPatientID(patientIDr:String)
+getPatientID():String
+setBranch(branches: List)
+getBranchs():List
+setAppointment(appointments: List)
+getAppointment():List
+displayInfo()


Appointment
-dentistName: String
-data: Date
-time: Time
-service: List
-payment: Payment or None
+setDentistName(dentistName:String)
+getDentistName():String
+setDate(data:Date)
+getDate():Date
+setTime(time:Time)
+getTime():Time
+setService(service:List)
+getService():List
+displayInfo()
+checkout():String
Service
-name: String
-cost: Float
-serviceType: ENUM
+setName(name:String)
+getName():String
+setCost(cost:Float)
+getCost():Float
+setServiceType(serviceType:ENUM)
+getServiceType():ENUM
+displayInfo()
*
Person
-name: String
-email: String
-phoneNumber: String
+setName(name:String)
+getName():String
+setEmail(email:String)
+getEmail():String
+setPhoneNumber(phoneNumber:String)
+getPhoneNumber():String
+displayInfo()
Staff
-staffRole: Enum
-branch: String
+setStaffRole(staffRole:Enum)
+getStaffRole():Enum
+setBranch(branch:String)
+getBranch():String
+displayInfo()
Patient
-patientID: String
-branches: List
-appointments: List
+setPatientID(patientIDr:String)
+getPatientID():String
+setBranch(branches: List)
+getBranchs():List
+setAppointment(appointments: List)
+getAppointment():List
+displayInfo()

Relationships:
dentalCompany has a **dentalBranch.** One dentalCompany can have many branches, so It is a composition as if the dentalCompany was removed, the branch cannot still exist independently since it is related to and managed through the company. Also, the lifetime of a branch depends on the company itself, and the addition of it is controlled by the class dentalCompany.

—----------

An aggregation relationship exists between **Staff** and **dentalBranch** because a staff member can belong to only one branch at a time, but a branch can have many staff members. Staff members have an instance variable branch, which represents the branch they belong to. Staff members are added to the Branch class using the add_staff method. The Staff object is part of the Branch object, but it does not own it.

—----------

Dentalbranch has a list of **Service** objects, and the relationship between Dentalbranch and Service classes can be characterized as a composition relationship. In other words, a Dentalbranch object is made up of many Service objects. As the class dentalBranch has a list of services, it keeps track of them. Here in our case, each branch can have the services provided such as cleaning or filling, which are not specific to one of the branches.

—----------

Patient and **Dentalbranch** have a composition relationship. This is the case because a Patient object is made up of one or more Dentalbranch objects, and if one is deleted, all related Patient objects will likewise be deleted. The Dentalbranch object can exist independently of the Patient objects linked with it, but a Patient object cannot exist in this scenario without being associated with a Dentalbranch object. As the class dentalBranch has a list of patients, it keeps track of them. It is a good approach as we don't limit the patient and let it be associated with one branch only, they have the freedom to choose.

—----------

The relationship between an **appointment** and a **service** since an appointment may be for multiple services and a service may be provided to multiple appointments. Each appointment object is made up of one or more instances of the Service class thanks to the composition relationship that exists between the Appointment and Service classes. This implies a connection between the Service object lifespan and the Appointment object lifecycle. All of the Service objects connected to an Appointment object are likewise destroyed when it is destroyed.

In my case, we are assessing the addition of patients booking appointments, so without services, we won't have appointments, so I will consider it as composition.

—----------

Patient has a one-to-many association with the class **Appointment** and vice versa because one patient can make several appointments yet only one patient can book an appointment. It is a binary association as it connects two classes. The patient class keeps track of the appointments the patient books before coming to the branch which allows easy access to all appointments for a particular patient. It is simpler and more flexible to get the list of appointments through each patient instead of storing it on the dentalBranch which will complicate things and let us search through all the branches to find the relevant appointments.

It is considered composition as if we deleted the patient, we won't be able to keep track of the appointments.
—----------
There is a one-to-one association between **Payment** and **Appointment** and vice versa as a Payment can be made for a single Appointment and an Appointment can only have one Payment. Assuming that just a single payment can be made only. It is considered composition as if we deleted the appointment, we won't need to have a payment then.
—----------
The class **Person** is the parent class and Both **Staff** and **Patient** inherit from it. It is a hierarchical inheritance, where more than one child(Staff and Patient) class is derived from a single-parent class(Person).
—----------
The relationship between the classes **Patient** and **Payment** is managed through the appointment class as discussed above and in this case, it is indirect. In general, each **patient** has one **payment**(one-to-one binary association).

Assumptions:

- Each service has a unique name and can be related to different branches.
- Each company can have many branches, but the names are not fixed, so using string will ensure flexibility.
- Each company has a unique address and phone number
- Each branch can have multiple services, staff, and patients.
- Each staff member has a unique name.
- Each person has a unique name including staff and patients.
- Each patient has a unique name and phone number.
- Each appointment has a unique date, time, patient, and staff. It can include different services as well.
- Each payment has a unique date and patient.
- Each payment can be done one single time.
- Each staff member can have only one staff role at the branch, and work only within that branch.
- Each patient can be related to different branches and have different appointments.

Further explanation:
We maintain a record of the appointment's payment in the Payment class. The payment instance variable is initialized to None when an appointment is established, denoting that the appointment has not yet been paid for. The overall cost of the appointment is determined and a Payment object is created with the final amount when the checkout method is called on the appointment object. The appointment object's payment instance variable is subsequently allocated to the Payment object. Through its instance variables and methods, the Payment object keeps track of the payment amount, date, payment method, and status (paid or unpaid). To receive a formatted receipt with the payment information, use the getReceipt function of the Payment class.

2- Python code with test cases.

```
#Define a class called "DentalCompany" with its instance variables
```

```
class DentalCompany:
    def __init__(self, companyName): # Define the constructor method for
the class
        # Set the instance variables for each object of the class
        self.companyName = companyName # Set the companyname attribute
        self.branches = []# initialize an empty list for branches

    def setBranch(self, branch): # method to add a new branch to the
company
        self.branches.append(branch)
    def getBranches(self): # method to get all  branches of the company
        return self.branches
    def setCompanyName(self, companyName): # method to set the company
name
        self.companyName = companyName
    def getCompanyName(self): # method to get the company name
        return self.companyName
    def displayInfo(self): #function to display all info
        print(f"Dental Company: {self.companyName}")
        print("Branches:")
        for branch in self.branches: # Iterate over each branch in the
list of branches and display information about each branch
            branch.displayInfo()
# Define a class called "Dentalbranch" with its instance variables
class Dentalbranch:
    def __init__(self, address, phoneNumber, manager, services=None,
staff=None, patients=None): # Define the constructor method for the
class
        # Set the instance variables for each object of the class
        self.address = address # Set the address attribute
        self.phoneNumber = phoneNumber # Set the phoneNumber attribute
        self.manager = manager# Set the manager attribute
        self.services = services if services is not None else [] # Set
the services to an empty list if no value is passed, otherwise set it
to the value passed
        self.staff = staff if staff is not None else [] # Set the staff
to an empty list if no value is passed, otherwise set it to the value
passed
        self.patients = patients if patients is not None else [] # Set
the patients to an empty list if no value is passed, otherwise set it
to the value passed
```

```
    def setService(self, service): # method to add a new service to the
branch
        self.services.append(service)
    def getServices(self): # method to get all services of the branch
        return self.services
    def setStaff(self, staff): # method to add a new staff member to the
branch
        self.staff.append(staff)
    def getStaff(self): # method to get all staff members of the branch
        return self.staff
    def setPatient(self, patient):# method to add a new patient to the
branch
        if patient not in self.patients: #Adds patient to the list of
patients for the current branch if not already present.
            self.patients.append(patient)
            patient.setBranch(self)
    def getPatients(self): # method to get all patients of the branch
        return self.patients
    def setAddress(self, address):  # method to set the branch address
        self.address = address
    def getAddress(self): # method to get the branch address
        return self.address
    def setPhoneNumber(self, phoneNumber): # method to set the branch
phone number
        self.phoneNumber = phoneNumber
    def getPhoneNumber(self): # method to get the branch phone number
        return self.phoneNumber
    def setManager(self, manager): # method to set the branch manager
        self.manager = manager
    def getManager(self): # method to get the branch manager
        return self.manager
    def displayInfo(self): #function to display all info
        print(f"\nBranch Info:\nAddress: {self.address}\nPhone number:
{self.phoneNumber}\nManager: {self.manager}")
        print("Services:")
        for service in self.services: #Display information for all
services
            service.displayInfo()
        print("Staff members:")
        for staff_member in self.staff: #Display information for all
staff
            staff_member.displayInfo()
        print("Patients:")
```

```
        for patient in self.patients: #Display information for all
patients
            patient.displayInfo()

# Importing the Enum class from the enum module
from enum import Enum
class ServiceType(Enum): # Define an enum class called "ServiceType"
for dental services
    cleaning=1
    implants=2
    crowns= 3
    fillings=4

class Service: # Define a class called "Service" with its instance
variables
    def __init__(self, name, cost, serviceType): # Define the
constructor method for the class
        # Set the instance variables for each object of the class
        self.name = name # Set the name attribute
        self.cost = cost # Set the cost attribute
        self.serviceType = serviceType # Set the serviceType attribute

    def setName(self, name): # method to set the service name
        self.name = name
    def getName(self): # method to get the service name
        return self.name
    def setCost(self, cost): # method to set the service cost
        self.cost = cost
    def getCost(self): # method to get the service cost
        return self.cost
    def setServiceType(self, serviceType):# method to set the service
type
        self.serviceType= serviceType
    def getServiceType(self): # method to get the service type
        return self.serviceType
    def displayInfo(self): #function to display all info
        print('Name =',self.name,' ,Cost=',self.cost,'
,ServiceType=',self.serviceType)

class Person: # Define a Python class called "Person" with its instance
variables
    def __init__(self, name, email, phoneNumber): # Define the
constructor method for the class
```

```
        # Set the instance variables for each object of the class
        self.name=name # Set the name attribute
        self.email=email # Set the email attribute
        self.phoneNumber= phoneNumber # Set the phoneNumber attribute

    def setName(self, name): # Define a method to set the name instance
variable
        self.name = name
    def getName(self): # Define a method to get the name instance
variable
        return self.name
    def setEmail(self, email): # Define a method to set the email
instance variable
        self.email = email
    def getEmail(self): # Define a method to get the email instance
variable
        return self.email
    def setPhoneNumber(self, phoneNumber): # Define a method to set the
phone number instance variable
        self.phoneNumber = phoneNumber
    def getPhoneNumber(self): # Define a method to get the phone number
instance variable
        return self.phoneNumber
    def displayInfo(self): #function to display all info
        print(f"\nName: {self.name}\nEmail: {self.email}\nPhoneNumbe:
{self.phoneNumber}")
# Importing the Enum class from the enum module
from enum import Enum
class StaffRole(Enum): # Defining an enumeration called StaffRole
    manager = 1
    receptionist = 2
    hygienist = 3
    dentist = 4
class Staff(Person): # Defining a Staff class that inherits from the
Person class
    def __init__(self, name, email, phoneNumber, staffRole, branch): #
Define the constructor method for the class
        # Calling the constructor of the parent class (Person) and
passing name, email, and phone number
        super().__init__(name, email, phoneNumber)
        # Set the instance variables for each object of the class
        self.staffRole = staffRole # Set the staffRole attribute
        self.branch = branch # Set the branch attribute
```

```
    def setStaffRole(self, staffRole): # Method to set the staff role of
the Staff class
        self.staffRole = staffRole
    def getStaffRole(self):# Method to get the staff role of the Staff
class
        return self.staffRole
    def setBranch(self, branch):  # Method to set the branch of the
Staff class
        self.branch = branch
    def getBranch(self): # Method to get the branch of the Staff class
        return self.branch
    def displayInfo(self): #function to display all info
        Person.displayInfo(self) #refer to the parent function
        print('StaffRole=',self.staffRole, 'Branch=',self.branch)

class Patient(Person): # Defining a Patient class that inherits from
the Person class
    def __init__(self, name, email, phoneNumber, patientID,
appointment=None): # Define the constructor method for the class
        # Calling the constructor of the parent class (Person) and
passing name, email, and phone number
        super().__init__(name, email, phoneNumber)
        # Set the instance variables for each object of the class
        self.patientID = patientID # Set the patientID attribute
        self.branches = []# initialize an empty list for branches
        self.appointment = appointment if appointment is not None else
[] # Set the appointments to an empty list if no value is passed,
otherwise set it to the value passed
     def setPatientID(self, patientID): # Method to set the patient ID
of the Patient class
        self.patientID = patientID
    def getPatientID(self): # Method to get the patient ID of the
Patient class
        return self.patientID
    def setBranch(self, branch): # Method to set the branch of the
Patient class
        self.branches.append(branch)
        branch.setPatient(self)
    def getBranches(self): # Method to get the Branches list of the
Patient class
        return self.branches
    def setAppointment(self, appointment): # Method to add an
appointment to the appointments list of the Patient class
```

```
        self.appointment.append(appointment)  # add appointment to
patient's list
    def getAppointment(self): # Method to get the appointments list of
the Patient class
        return self.appointment
    def displayInfo(self): #function to display all info
        print("Patient Info:")
        Person.displayInfo(self) #refer to the parent function
        print(f"\nID: {self.patientID}")
        print("Appointments:")
        for appointment in self.appointment: #Display appointment for
all patients
            appointment.displayInfo()
class Appointment: # Define a class Appointment with its instance
variables
    def __init__(self, dentistName, date, time, service=None): # Define
the constructor method for the class
        # Set the instance variables for each object of the class
        self.dentistName = dentistName # Set the dentist name attribute
        self.date = date # Set the date attribute
        self.time = time # Set the time attribute
        self.service = service if service is not None else [] # Set the
service to an empty list if no value is passed, otherwise set it to the
value passed
        self.payment = None # Initialize the payment attribute to None

    def setDentistName(self, dentistName): # Method to set the dentist
name
        self.dentistName = dentistName
    def getDentistName(self):# Method to get the dentist name
        return self.dentistName
    def setDate(self, date):# Method to set the date
        self.date = date
    def getDate(self): # Method to get the date
        return self.date
    def setTime(self, time): # Method to set the time
        self.time = time
    def getTime(self): # Method to get the time
        return self.time
    def setService(self, service): # Method to set the service
        self.service = service
    def getService(self): # Method to get the service
        return self.service
```

```
    def displayInfo(self): #function to display all info
        print(f"\nAppointment Info:\nDentistName:
{self.dentistName}\nDate: {self.date}\nTime: {self.time}")
        print("Services:")
        for service in self.service: #Display service for all patients
            service.displayInfo()
    def checkout(self):# Method to calculate the total cost of the
appointment and create a payment object
        total_cost = 0
        for s in self.service:
          total_cost += s.cost
        # calculate total cost based on additional services or
procedures
        vat = 0.05 * total_cost
        final_cost = total_cost + vat
        self.payment = Payment(final_cost, self.date,
PaymentMethod.Card)
        return self.payment.getReceipt()

from enum import Enum
# Define an enumeration class for the payment method
class PaymentMethod(Enum):
    Card = 1
    Cash = 2
# Define an enumeration class for the payment status
class PaymentStatus(Enum):
    UNPAID = 1
    PAID_IN_FULL = 2
# Define a class Payment with its instance variables
class Payment:
    def __init__(self, amount, date, paymentMethod,
status=PaymentStatus.UNPAID): # Define the constructor method for the
class
        # Set the instance variables for each object of the class
        self.amount = amount # Set the amount attribute
        self.date = date # Set the date attribute
        self.paymentMethod = PaymentMethod(paymentMethod) # Set the
payment method attribute
        self.status = PaymentStatus(status) # Set the status attribute
with an optional default value of "unpaid"

    # Method to process the payment
    def processPayment(self, paymentMethod):
```

```
        self.paymentMethod = paymentMethod
        self.status = PaymentStatus.PAID_IN_FULL

    # Method to get the payment receipt
    def getReceipt(self):
        receipt = "Payment receipt:\n"
        receipt += f"Date: {self.date}\n"
        receipt += f"Amount: {self.amount}\n"
        receipt += f"Payment method: {self.paymentMethod}\n"
        receipt += f"Payment status: {self.status}\n"
        return receipt
    def getAmount(self): # Getter method for amount
        return self.amount
    def setAmount(self, amount): # Setter method for amount
        self.amount = amount
    def getDate(self): # Getter method for date
        return self.date
    def setDate(self, date): # Setter method for date
        self.date = date
    def getPaymentMethod(self): # Getter method for paymentMethod
        return self.paymentMethod
    def setPaymentMethod(self, paymentMethod): # Setter method for
paymentMethod
        self.paymentMethod = paymentMethod
    def getStatus(self): # Getter method for status
        return self.status
    def setStatus(self, status): # Setter method for status
        self.status = status
    def displayInfo(self): #function to display all info
        print('Amount =',self.amount,' ,Date=',self.date, '
,paymentMethod=',self.paymentMethod,' ,Status=',self.status)
```

a. the addition of branches to the dental company.

```
# Create a DentalCompany object
company = DentalCompany("Bright Smiles")
company.displayInfo()
#Create staff members
dentist1 = Staff("John Smith", "john@example.com", "0507478344",
StaffRole.dentist.name, branch="Al Rashidiya 1")
hygienist1 = Staff("Jane Doe", "jane@example.com", "0506467433"
,StaffRole.hygienist.name, branch="Al Rashidiya 1")
manager1 = Staff("Farah Mohammed", "Farah@example.com", "0507878344",
StaffRole.manager.name, branch="Al Rashidiya 1")
```

```
receptionist1= Staff("Zahra Mohammed", "Zahra@example.com",
"0503459100", StaffRole.receptionist.name, branch="Al Rashidiya 1")
dentist2 = Staff("Ameen Salem", "Ameen@example.com", "0507478344",
StaffRole.dentist.name, "Al Hamidiya 1")
hygienist2 = Staff("Fatima Khaled", "Fatima@example.com", "0506467433"
,StaffRole.hygienist.name, "Al Hamidiya 1")
manager2 = Staff("Mohammed Khalied", "Mohammed@example.com",
"0563421788", StaffRole.manager.name, "Al Hamidiya 1")
receptionist2= Staff("Taif Obaid", "Taif@example.com", "0521237866",
StaffRole.receptionist.name, "Al Hamidiya 1")
# Add some services to the branch
service1 = Service("Cleaning", 100, ServiceType.cleaning.name)
service2 = Service("Implants", 1000, ServiceType.implants.name)
service3=Service("crowns", 2000, ServiceType.crowns.name)
service4=Service("fillings", 500, ServiceType.fillings.name)
#patients
patient1 = Patient("Alice Smith", "alice@example.com", "0506781233",
"200346F55")
patient2 = Patient("Bob Jones", "bob@example.com", "0556463877",
"200156F33")
patient3 = Patient("Reem Salem", "Reem@example.com", "0527463877",
"200843R11")
# Create two Dentalbranch objects
branch1 = Dentalbranch("Al Rashidiya 1", "0507478344", "Dr.
Smith",services=[service1, service2, service3,
service4],staff=[dentist1,hygienist1, manager1, receptionist1]
,patients=[patient1,patient2])
branch2 = Dentalbranch("Al Hamidiya 1", "0506777342", "Dr.
Brown",services=[service1, service2, service3,
service4],staff=[dentist2,hygienist2, manager2, receptionist2]
,patients=[patient1,patient3])
# Add the two branches to the company
company.setBranch(branch1)
company.setBranch(branch2)
# Get all the branches of the company
branches = company.getBranches()
# Print the details of each branch
for branch in branches:
 branch.displayInfo()
 print("-----------")
```

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

```
# Create a Dentalbranch object

branch = Dentalbranch("Al Rashidiya 1", "0507478344", "Dr. Smith")

# Create staff members

dentist1 = Staff("John Smith", "john@example.com", "0507478344",
StaffRole.dentist.name, branch="Al Rashidiya 1")

hygienist1 = Staff("Jane Doe", "jane@example.com", "0506467433"
,StaffRole.hygienist.name, branch="Al Rashidiya 1")

manager1 = Staff("Farah Mohammed", "Farah@example.com", "0507878344",
StaffRole.manager.name, branch="Al Rashidiya 1")

receptionist1= Staff("Zahra Mohammed", "Zahra@example.com",
"0503459100", StaffRole.receptionist.name, branch="Al Rashidiya 1")

# Add staff members to the branch

branch.setStaff(dentist1)

branch.setStaff(hygienist1)

branch.setStaff(manager1)

branch.setStaff(receptionist1)

# Create some Patient objects

patient1 = Patient("Alice Smith", "alice@example.com", "0506781233",
"200346F55")

patient2 = Patient("Bob Jones", "bob@example.com", "0556463877",
"200156F33")

# Add the patients to the branch

branch.setPatient(patient1)

branch.setPatient(patient2)

# Add some services to the branch

service1 = Service("Cleaning", 100, ServiceType.cleaning.name)

service2 = Service("Implants", 1000, ServiceType.implants.name)
```

```
service3=Service("crowns", 2000, ServiceType.crowns.name)

service4=Service("fillings", 500, ServiceType.fillings.name)

# Add the services to the branch

branch.setService(service1)

branch.setService(service2)

branch.setService(service3)

branch.setService(service3)

# Get all the staff members of the branch

#staff = branch.getStaff()

branch.displayInfo()
```

c. the addition of patients booking appointments

```
# create some services

service1 = Service("Cleaning", 100, ServiceType.cleaning.name)

service2 = Service("Implants", 1000, ServiceType.implants.name)

# create a new patient object

new_patient = Patient("Alice Smith", "alice@example.com", "0506781233",
"200346F55")

# create a new appointment object

new_appointment = Appointment("Khaled", date='2023-05-01', time='10:00
AM', service=[service1, service2])

# add the new appointment to the patient

new_patient.setAppointment(new_appointment)

new_patient.displayInfo()
```

d. the display of payment receipts for patient services (one or more) upon checking out. The final bill should be presented to the patient on completion of service.

```
from datetime import time,date
# Create an instance of the Appointment class
appointment = Appointment("Dr. John Doe", date(2023, 4, 16),
time(10,0))

# Add a service to the appointment
service1 = Service("Cleaning", 100, ServiceType.cleaning.name)
appointment.setService([service1])
# Display the appointment information
appointment.displayInfo()
# Checkout the appointment and process the payment
receipt = appointment.checkout()
print(receipt)
# Process the payment after completion of service
payment = appointment.payment
payment.processPayment(PaymentMethod.Cash)
print(payment.getReceipt())
```

3- Github repository link