Assignment 2: Software Implementation

Alia Lootah

College of Interdisciplinary Studies, Zayed University

ICS220 - 22873 Programming Fundamentals

Professor Kuhail

April 15, 2023

**UML Class Diagram and Description:**

**Classes and Attributes:**

1. DentalBranch
   * address (private)
   * phoneNumber (private)
   * manager (private)
2. Staff
   * name (private)
   * id (private)
   * title (private)
3. Manager (inherits from Staff)
   * yearsOfExperience (private)
4. Receptionist (inherits from Staff)
   * workingHours (private)
5. Hygienist (inherits from Staff)
   * certification
6. Dentist (inherits from Staff)
   * specialization (private)
7. DentalService
   * serviceName (private)
   * serviceCost (private)
8. Patient
   * name (private)
   * id (private)
   * phoneNumber (private)
9. Appointment
   * appointmentID (private)
   * date (private)
   * time (private)
10. Bill
    * billID (private)
    * totalCost (private)
    * VAT (private)

|  |
| --- |
| DentalBranch |
| -address: str  -phoneNumber: str  -manager: Manager |
| +addStaff(staff: Staff)  +addPatient(patient: Patient)  +addService(service: DentalService) |

**Diagram:**

|  |
| --- |
| DentalService |
| -serviceName: str  \*  -serviceCost: float |

\*

|  |
| --- |
| Patient |
| -name: str  -id: str  -phoneNumber:str |
| +bookAppointment(appointment: Appointment) |

\*



|  |
| --- |
| Manager |
| - yearsOfExperience: int |
| + getYearsOfExperience(): int |

\*

|  |
| --- |
| Staff |
| -name: str  -id: str  -title: str  -salary: float |
| + getName(): str  + getId(): str  + getTitle(): str  + getSalary(): float |

|  |
| --- |
| Hygienist |
| - certification: str |
| + getCertification(): str |

|  |
| --- |
| Dentist |
| - specialization: str |
| + getSpecialization(): str |

1

|  |
| --- |
| Receptionist |
| - workingHours: str |
| + getWorkingHours(): str |

|  |
| --- |
| Appointment |
| -appointmentID:str  -date: Date  -time: Time  -dentist: Dentist  -services: List[DentalService] |
| +generateBill(): Bill |

|  |
| --- |
| Bill |
| -billID: str  1  -totalCost: float  -VAT: float |
| +displayReceipt() |

\*

**Relationships:**

1. DentalBranch has a 1-to-many relationship with Staff, as one branch can have multiple staff members.
2. DentalBranch has a 1-to-many relationship with DentalService, as one branch can offer multiple services.
3. DentalBranch has a 1-to-many relationship with Patient, as one branch can have multiple patients.
4. Staff has a generalization relationship with Manager, Receptionist, Hygienist, and Dentist. These classes inherit from the Staff class.
5. Patient has a 1-to-many relationship with Appointment, as one patient can have multiple appointments.
6. Appointment has a 1-to-1 relationship with Dentist, as one appointment is linked to a single dentist.
7. Appointment has a 1-to-many relationship with DentalService, as multiple services can be provided during one appointment.
8. Bill has a 1-to-1 relationship with Appointment, as one bill is linked to one appointment.

The diagram shows the relationships between these classes, including inheritance, composition, and association relationship

1. **Inheritance**: Manager, Receptionist, Hygienist, and Dentist classes inherit from the Staff class. This relationship is shown by the arrows pointing from the derived classes to the base class.

* The Manager "is-a" Staff and has all the properties and methods of the Staff class.
* The Receptionist "is-a" Staff and has all the properties and methods of the Staff class.
* The Hygienist "is-a" Staff and has all the properties and methods of the Staff class.
* The Dentist "is-a" Staff and has all the properties and methods of the Staff class.

1. **Composition**: DentalBranch class has a composition relationship with the Staff class, as it has an attribute of type Manager, which is a subclass of Staff. This is represented by the filled diamond symbol next to the DentalBranch class and the line connecting it to the Staff class. This relationship indicates that the Manager is an integral part of the DentalBranch, and if the DentalBranch ceases to exist, so does its Manager.
2. **Association**:

* DentalBranch - Staff: A DentalBranch can have multiple Staff members. This association is represented by a line connecting the DentalBranch class to the Staff class, with an asterisk (\*) on the Staff side to denote the "many" part of the relationship.
* DentalBranch - Patient: A DentalBranch can have multiple Patients. This association is represented by a line connecting the DentalBranch class to the Patient class, with an asterisk (\*) on the Patient side to denote the "many" part of the relationship.
* DentalBranch - DentalService: A DentalBranch can offer multiple DentalServices. This association is represented by a line connecting the DentalBranch class to the DentalService class, with an asterisk (\*) on the DentalService side to denote the "many" part of the relationship.
* Patient - Appointment: A Patient can have multiple Appointments. This association is represented by a line connecting the Patient class to the Appointment class, with an asterisk (\*) on the Appointment side to denote the "many" part of the relationship.
* Appointment - Dentist: An Appointment has one Dentist. This association is represented by a line connecting the Appointment class to the Dentist class, with a "1" on the Dentist side to denote the "one" part of the relationship.
* Appointment - DentalService: An Appointment can have multiple DentalServices. This association is represented by a line connecting the Appointment class to the DentalService class, with an asterisk (\*) on the DentalService side to denote the "many" part of the relationship.
* Appointment - Bill: An Appointment has one Bill. This association is represented by a line connecting the Appointment class to the Bill class, with a "1" on the Bill side to denote the "one" part of the relationship.

**Assumptions:**

1. Staff can be uniquely identified by their 'id' attribute.
2. Patients can be uniquely identified by their 'id' attribute.
3. Each dental service has a unique 'serviceName' and a corresponding 'serviceCost'.
4. Appointments can be uniquely identified by the 'appointmentID' attribute.
5. Bills can be uniquely identified by the 'billID' attribute.
6. The 5% VAT is a constant value in the Bill class.

**Python code:**

class DentalBranch:

def \_\_init\_\_(self, address, phoneNumber, manager):

self.address = address # Stores the branch address

self.phoneNumber = phoneNumber # Stores the branch phone number

self.manager = manager # Stores the branch manager

self.staff = [] # Initializes the list of staff members

self.patients = [] # Initializes the list of patients

self.services = [] # Initializes the list of dental services

def addStaff(self, staff):

"""Add a staff member to the branch."""

self.staff.append(staff) # Appends the staff member to the staff list

def addPatient(self, patient):

"""Add a patient to the branch."""

self.patients.append(patient) # Appends the patient to the patients list

def addService(self, service):

"""Add a dental service to the branch."""

self.services.append(service) # Appends the service to the services list

class Staff:

def \_\_init\_\_(self, name, staffId, title, salary):

self.name = name # Initializes the name attribute

self.id = staffId # Initializes the staff Id attribute

self.title = title # Initializes the title attribute

self.\_salary = salary # Initializes the salary attribute

def getName(self):

return self.\_name # Returns the name attribute

def getId(self):

return self.\_id # Returns the staff id attribute

def getTitle(self):

return self.\_title # Returns the title attribute

def getSalary(self):

return self.\_salary # Returns the salary attribute

class Manager(Staff):

def \_\_init\_\_(self, name, staffId, title, salary, yearsOfExperience):

super().\_\_init\_\_(name, staffId, title, salary) # Calls the parent class constructor

self.\_yearsOfExperience = yearsOfExperience # Initializes yearsOfExperience attribute

def getYearsOfExperience(self):

return self.\_yearsOfExperience # Returns the yearsOfExperience attribute

class Receptionist(Staff):

def \_\_init\_\_(self, name, staffId, title, salary, workingHours):

super().\_\_init\_\_(name, staffId, title, salary) # Calls the parent class constructor

self.\_workingHours = workingHours # Initializes workingHours attribute

def getWorkingHours(self):

return self.\_workingHours # Returns the workingHours attribute

class Hygienist(Staff):

def \_\_init\_\_(self, name, staffId, title, salary, certification):

super().\_\_init\_\_(name, staffId, title, salary) # Calls the parent class constructor

self.\_certification = certification # Initializes certification attribute

def getCertification(self):

return self.\_certification # Returns the certification attribute

class Dentist(Staff):

def \_\_init\_\_(self, name, staffId, title, salary, specialization):

super().\_\_init\_\_(name, staffId, title, salary) # Calls the parent class constructor

self.\_specialization = specialization # Initializes specialization attribute

def get\_specialization(self):

return self.\_specialization # Returns the specialization attribute

class DentalService:

def \_\_init\_\_(self, serviceName, serviceCost):

self.serviceName = serviceName # Stores the service name

self.serviceCost = serviceCost # Stores the service cost

class Patient:

def \_\_init\_\_(self, name, patientId, phoneNumber):

self.name = name # Stores the patient's name

self.id = patientId # Stores the patient's unique ID

self.phoneNumber = phoneNumber # Stores the patient's phone number

self.appointments = [] # Initializes the list of appointments

def bookAppointment(self, appointment):

"""Book an appointment for the patient."""

self.appointments.append(appointment) # Appends the appointment to the appointments list

class Appointment:

def \_\_init\_\_(self, appointmentId, date, time, dentist, services):

self.appointmentId = appointmentId # Stores the appointment's unique ID

self.date = date # Stores the appointment date

self.time = time # Stores the appointment time

self.dentist = dentist # Stores the assigned dentist

self.services = services # Stores the list of dental services for the appointment

def generateBill(self):

"""Generate a bill for the appointment."""

totalCost = sum(service.serviceCost for service in self.services) # Calculates the total cost of services

vat = totalCost \* 0.05 # Calculates the VAT

return Bill(self.appointmentId, totalCost, vat) # Returns a new Bill instance

class Bill:

def \_\_init\_\_(self, billId, totalCost, vat):

self.billId = billId # Stores the bill's unique ID

self.totalCost = totalCost # Stores the total cost of services

self.vat = vat # Stores the VAT

self.finalAmount = totalCost + vat # Stores the final amount after adding VAT

def displayReceipt(self):

"""Display the payment receipt."""

print(f"Bill ID: {self.billId}") # Prints the bill ID

print(f"Total Cost: {self.totalCost}") # Prints the total cost

print(f"VAT: {self.vat}") # Prints the VAT

print(f"Final Amount: {self.finalAmount}") # Prints the final amount

# TEST CASES

# Test case a: Addition of branches to the dental company

manager = Manager("Mary", "M1", "Manager", 5000, 5)

branch1 = DentalBranch("Dubai", "(04) 123 4567", manager)

branch2 = DentalBranch("Sharjah", "(06) 765 4321", manager)

# Test case b: Addition of dental services, staff, and patients to a branch

cleaning = DentalService("Cleaning", 250)

implants = DentalService("Implants", 1500)

branch1.addService(cleaning)

branch1.addService(implants)

receptionist = Receptionist("Kim", "R1", "Receptionist", 4000, "9 AM - 5 PM")

dentist = Dentist("Dr. Brown", "D1", "Dentist", 9000, "Orthodontics")

branch1.addStaff(receptionist)

branch1.addStaff(dentist)

patient1 = Patient("Aisha", "P1", "050 987 6543")

patient2 = Patient("Fatma", "P2", "055 123 4567")

branch1.addPatient(patient1)

branch1.addPatient(patient2)

# Test case c: Addition of patients booking appointments

appointment1 = Appointment("Reciept1", "2023-04-20", "11:00 AM", dentist, [cleaning])

appointment2 = Appointment("Reciept2", "2023-06-13", "14:00 PM", dentist, [implants, cleaning])

patient1.bookAppointment(appointment1)

patient2.bookAppointment(appointment2)

# Test case d: Display of payment receipts for patient services (one or more) upon checking out

bill1 = appointment1.generateBill()

bill2 = appointment2.generateBill()

print("Receipt for Patient 1:", patient1.name)

print(appointment1.date, appointment1.time)

print("Branch:", branch1.address)

bill1.displayReceipt()

print("\nReceipt for Patient 2:", patient2.name)

print(appointment2.date, appointment2.time)

print("Branch:", branch2.address)

bill2.displayReceipt()

**Output**:

Receipt for Patient 1: Aisha

2023-04-20 11:00 AM

Branch: Dubai

Bill ID: Reciept1

Total Cost: 250

VAT: 12.5

Final Amount: 262.5

Receipt for Patient 2: Fatma

2023-06-13 14:00 PM

Branch: Sharjah

Bill ID: Reciept2

Total Cost: 1750

VAT: 87.5

Final Amount: 1837.5

**Summary**:

The task of creating a billing system for a dental service allowed me to apply my knowledge of object-oriented programming that I have learned in class to a real-world scenario. Throughout this exercise, I have learned and expanded my knowledge and understanding of various aspects related to designing and implementing a software application for managing a dental clinic.

I learned about the purpose and usage of UML class diagrams to visually represent the structure of a system, including its classes, attributes, operations, and relationships among them.

I gained an understanding of different types of class relationships, such as association, composition, and inheritance, and how they are represented in UML class diagrams.

I learned to identify and explain associations between classes, such as the relationships between DentalBranch and Staff, DentalBranch and Patient, DentalBranch and DentalService, Patient and Appointment, Appointment and Dentist, Appointment and DentalService, and Appointment and Bill.

I learned to identify and explain the composition relationship between classes, such as the relationship between DentalBranch and Manager, which represents a strong form of aggregation where the whole (DentalBranch) has ownership of its parts (Manager).

I learned to identify and explain the inheritance relationships between classes, such as the relationships between Staff and its subclasses (Manager, Receptionist, Hygienist, and Dentist), which represent "is-a” relationships and promote code reusability and modularity.

I gained the experience in writing Python code to implement the UML class diagram, including defining classes, attributes, methods, and test cases to showcase the program features.

By learning and applying these concepts, I have developed the skills to effectively design and implement software applications to solve real-world problems, such as the dental clinic management system in this scenario.

**GitHub**: <https://github.com/alyalootah/Assignment-2-Software-Implementation>