ICS220 22873 Program. Fund.

Assignment 2: Software Implementation

Prof Kuhail

Fri 14 April 2023

Shamsa Rasehd

multiple branches
Each dental branch has an address, phone number, and a manager.

Scenario:

Consider the following problem statement:

A dental company, "Bright Smiles", has multiple branches (i.e., dental clinics) and would like you to create a software application to help manage the business processes and dental services. Each dental branch has an address, phone number, and a manager. A dental branch offers dental services to patients. Examples of services include cleaning, implants, crowns, fillings, and more. Each of the services has a cost. The clinic keeps track of its patients and staff. The staff includes managers, receptionists, hygienists, and dentists. The patient needs to book an appointment before coming to the clinic. Upon checkout, the clinic charges the patient depending on the services she/he has received. Also, a 5% value-added tax (VAT) is added to the final bill.

Link Github
:https://github.com/shamsaalshu/Assignment-2-Software-Implementation-

1. DentalCompany:
    i. Name: string
    ii. branch:List[Branch]
2. Branch:
    - name :string
    - address:string
    - phone number:string
    - manager:Manger
    - dental services:List[Services]
    - patients:List[Patient]
    - staff:List[Staff]

3. Person:
- firstName: string
- lastName: string
- phoneNumber:string

- gender:Enum
- dateOfBirth:string

    a. Staff (inherits from Person):
        - employeeID:string

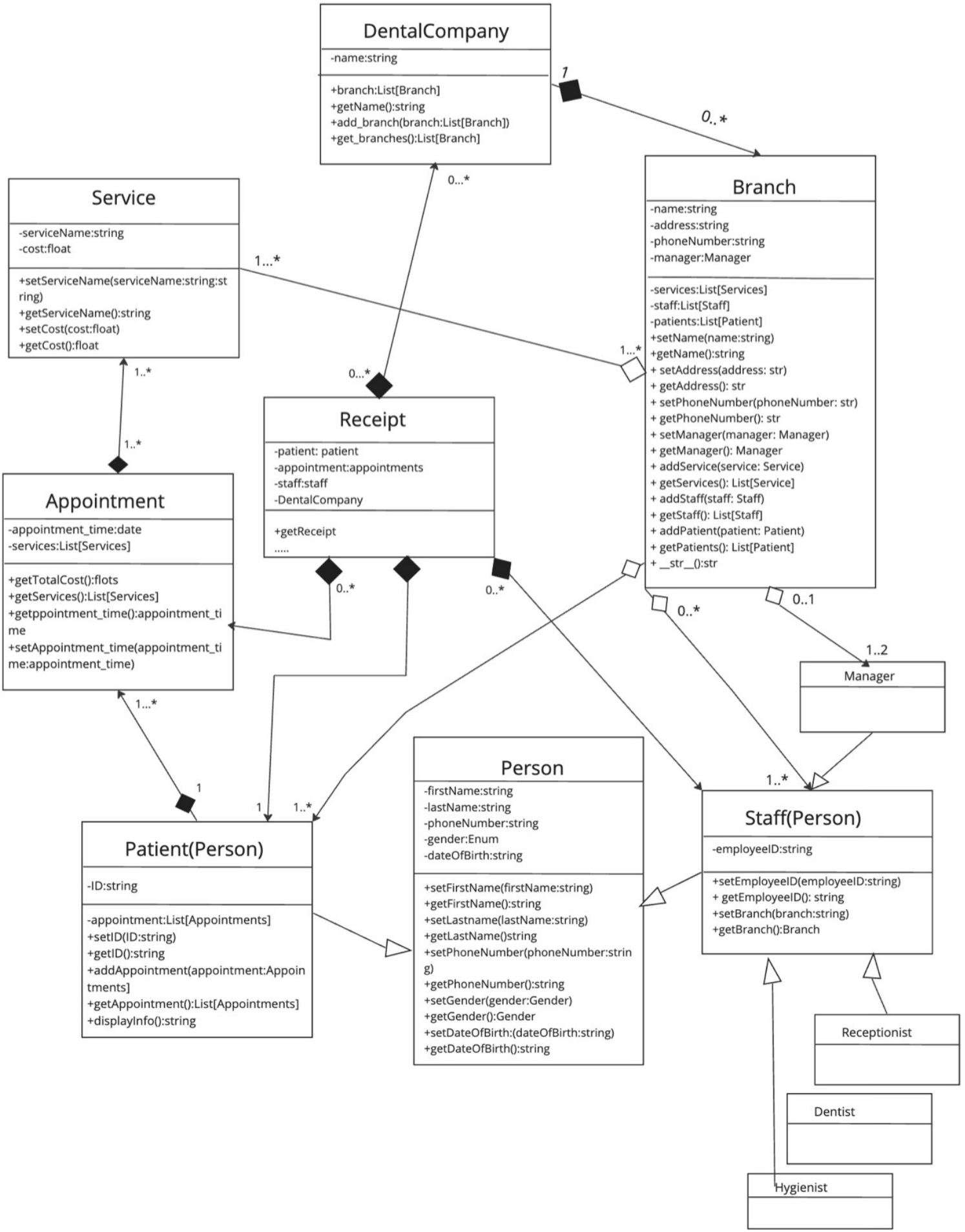
        i. Manager (inherits from StaffMember):
        ii. Receptionist (inherits from StaffMember):
        iii. Hygienist (inherits from StaffMember):
        iv. Dentist (inherits from StaffMember):

    b. Patient (inherits from Person):
        - appointments: List[Appointment]
        - ID:string

4. Appointment:
- services:List[Services]
- dateTime:datetime
- Method: totalCost:float

5. Services:
- serviceName:string
- cost:float

6. Receipt:
- patient:patient
- appointments:appointments
- Staff:staff
- DentalCompany:DentalCompany


DentalCompany
-name:string
+branch:List[Branch]
+getName():string
+add_branch(branch:List[Branch])
+get_branches():List[Branch]
1
0..*
Service
-serviceName:string
-cost:float
+setServiceName(serviceName:string:string)
+getServiceName():string
+setCost(cost:float)
+getCost():float
1...*
0...*
Branch
-name:string
-address:string
-phoneNumber:string
-manager:Manager
-services:List[Services]
-staff:List[Staff]
-patients:List[Patient]
+setName(name:string)
+getName():string
+ setAddress(address: str)
+ getAddress(): str
+ setPhoneNumber(phoneNumber: str)
+ getPhoneNumber(): str
+ setManager(manager: Manager)
+ getManager(): Manager
+ addService(service: Service)
+ getServices(): List[Service]
+ addStaff(staff: Staff)
+ getStaff(): List[Staff]
+ addPatient(patient: Patient)
+ getPatients(): List[Patient]
+ __str__():str
1...*
1..*
1..*
0...*
Receipt
-patient: patient
-appointment:appointments
-staff:staff
-DentalCompany
+getReceipt
.....
Appointment
-appointment_time:date
-services:List[Services]
+getTotalCost():flots
+getServices():List[Services]
+getppointment_time():appointment_time
+setAppointment_time(appointment_time:appointment_time)
0..*
0..*
0..*
0..1
1..2
Manager
1...*
1
1
1..*
1..*
Person
-firstName:string
-lastName:string
-phoneNumber:string
-gender:Enum
-dateOfBirth:string
+setFirstName(firstName:string)
+getFirstName():string
+setLastname(lastName:string)
+getLastName()string
+setPhoneNumber(phoneNumber:string)
+getPhoneNumber():string
+setGender(gender:Gender)
+getGender():Gender
+setDateOfBirth:(dateOfBirth:string)
+getDateOfBirth():string
Patient(Person)
-ID:string
-appointment:List[Appointments]
+setID(ID:string)
+getID():string
+addAppointment(appointment:Appointments]
+getAppointment():List[Appointments]
+displayInfo():string
Staff(Person)
-employeeID:string
+setEmployeeID(employeeID:string)
+ getEmployeeID(): string
+setBranch(branch:string)
+getBranch():Branch
Receptionist
Dentist
Hygienist

```
from datetime import datetime
from enum import Enum


class Gender(Enum):
    MALE = "Male"
    FEMALE = "Female"


class DentalCompany:
    def __init__(self,name):
        self.__name = name  # Initialize the name of the company
        self.branches = []  # Create an empty list to hold the branches

    def getName(self):
        return self.__name  # Return the name of the company

    def add_branch(self, branch):
        self.branches.append(branch)  # Add a branch to the list of branches

    def get_branches(self):
        return self.branches  # Return the list of branches



class Branch:
    def __init__(self, name, address, phoneNumber, manager):
        self.__name = name  # Initialize the name of the branch
        self.__address = address  # Initialize the address of the branch
        self.__phoneNumber = phoneNumber  # Initialize the phone number of the
branch
        self.__manager = manager  # Initialize the manager of the branch
        self.__services = []  # Create an empty list to hold the services offered
        self.__patients = []  # Create an empty list to hold the patients registered
```

```
        self.__staff = []  # Create an empty list to hold the staff members

    def setName(self, name):
        self.__name = name  # Set the name of the branch

    def getName(self):
        return self.__name  # Get the name of the branch

    # Address
    def setAddress(self, address):
        self.__address = address  # Set the address of the branch

    def getAddress(self):
        return self.__address  # Get the address of the branch

    # Phone number
    def setPhoneNumber(self, phoneNumber):
        self.__phoneNumber = phoneNumber  # Set the phone number of the branch

    def getPhoneNumber(self):
        return self.__phoneNumber  # Get the phone number of the branch

    # Manager
    def setManager(self, manager):
        self.__manager = manager  # Set the manager of the branch

    def getManager(self):
        return self.__manager  # Get the manager of the branch

    # Dental services
    def addService(self, Service):
```

```
        self.__services.append(Service)  # Add a service to the list of services
offered


    def getServices(self):

        return self.__services  # Get the list of services offered


    # Staff member

    def addStaff(self, staff_member):

        self.__staff.append(staff_member)  # Add a staff member to the list of staff
members


    def getStaff(self):

        return self.__staff  # Get the list of staff members


    # Patients

    def addPatient(self, patient):

        self.__patients.append(patient)  # Add a patient to the list of patients


    def getPatients(self):

        return self.__patients  # Get the list of patients


        # String representation of Branch information

    def __str__(self):

        return f"Branch Name: {self.__name}\nAddress: {self.__address}\nPhone
Number: {self.__phoneNumber}\nManager: {self.__manager.getFirstName()}
{self.__manager.getLastName()}"


# Define the Person class with first name, last name, gender and phone number
attributes


class Person:

    def __init__(self, firstName, lastName, phoneNumber, gender, dateOfBirth):

        self.__firstName = firstName  # Initialize the first name of the person

        self.__lastName = lastName  # Initialize the last name of the person
```

```
        self.__phoneNumber = phoneNumber  # Initialize the phone number of the
person

        self.__gender = gender  # Initialize the gender of the person

        self.__dateOfBirth = dateOfBirth



    def setFirstName(self, firstName):

        self.__firstName = firstName  # Set the first name of the person


    def getFirstName(self):

        return self.__firstName  # Get the first name of the person



    # Last name

    def setLastName(self, lastName):

        self.__lastName = lastName  # Set the last name of the person


    def getLastName(self):

        return self.__lastName  # Get the last name of the person


    def setPhoneNumber(self, phoneNumber):

        self.__phoneNumber = phoneNumber

    def getPhoneNumber(self):

        return self.__phoneNumber


# Setter for the 'gender' property.

# This method sets the gender value of the Person object.

# It checks if the provided 'gender' is an instance of the Gender Enum.

# If not, it raises a ValueError with the message "Invalid gender value".


    def setGender(self, gender):

        if isinstance(gender, Gender):
```

```
            self.__gender = gender
        else:
            raise ValueError("Invalid gender value")
    def getGender(self):
        return self.__gender


    def getDateOfBirth(self):
      return self.__dateOfBirth


    def setDateOfBirth(self,dateOfBirth):
      self.__dateOfBirth=dateOfBirth



# Define the Staff class as a subclass of Person with an additional employee ID
attribute

class Staff(Person):

    def __init__(self, firstName, lastName, phoneNumber,
gender,dateOfBirth,employeeID):

        # Call the constructor of the Person class to set the first name, last name,
and phone number attributes

        super().__init__(firstName, lastName, phoneNumber,gender,dateOfBirth)

        # Set the employee ID attribute

        self.__employeeID = employeeID


     # Define getters and setters for the employee ID attribute

    def setEmployeeID(self, employee_id: str):

        self.__employeeID = employee_id

    def getEmployeeID(self):

        return self.__employeeID


    # Set branch for a staff member

    def setBranch(self, branch: Branch):

        self.__branch = branch
```

```
    # Get branch information for a staff member
    def getBranch(self):
        return self.__branch




# Define the Manager, Receptionist, Hygienist, and Dentist classes as subclasses of
Staff with no additional attributes
class Manager(Staff):
    pass


class Receptionist(Staff):
    pass


class Hygienist(Staff):
    pass


class Dentist(Staff):
    pass


# Define the Patient class as a subclass of Person with an additional ID attribute
and a list of appointments
class Patient(Person):


    def __init__(self, firstName, lastName, phoneNumber, gender, dateOfBirth, ID):
        # Call the constructor of the Person class to set the first name, last name,
and phone number attributes
        super().__init__(firstName, lastName, phoneNumber, gender, dateOfBirth)
        # Set the ID attribute and initialize the list of appointments
        self.__ID = ID
        self.__appointments = []
```

```
    # Define getters and setters for the ID attribute
    def getID(self):
        return self.__ID

    def setID(self, ID):
        self.__ID = ID
    # Define getters for the appointments list
    def getAppointments(self):
        return self.__appointments
    # Define a method to add an appointment to the list
    def bookAppointment(self, appointment):
        self.__appointments.append(appointment)
    def displayInfo(self):
      return
f'Name:{self.getFirstName()}{self.getLastName()}\nGender:{self.getGender().name}\nP
hone number{self.getPhoneNumber()}'



# Define the Appointment class with service and appointment time attributes
class Appointment:
    def __init__(self, services, appointment_time):
        # Set the list of services and appointment time attributes
        self.__services = services
        self.__appointment_time = appointment_time

     # Define a method to calculate the total cost of the services
    def getTotalCost(self):
        return sum(service.getCost() for service in self.__services)
    # Define getters and setters for the services and appointment time attributes
    def getServices(self):
        return self.__services

    def getAppointmentTime(self):
```

```
        return self.__appointment_time


    def setAppointmentTime(self, appointment_time):
        self.__appointment_time = appointment_time


# Define the Service class with name and cost attributes
class Service:
    def __init__(self, serviceName, cost ):
        # Set the name and cost attributes
        self.__serviceName = serviceName
        self.__cost = cost


    # Define getters and setters for the name and cost attributes
    def setServiceName(self, serviceName):
        self.__serviceName = serviceName
    def getServiceName(self):
        return self.__serviceName
    # Cost
    def setCost(self, cost):
        self.__cost = cost
    def getCost(self):
        return self.__cost



# Define the Receipt class with patient, appointments, and dental company
attributes


class Receipt:
    def __init__(self, patient, appointments,staff,DentalCompany):
        # Set the patient, appointments, and dental company attributes
        self.__patient = patient
        self.__appointments = appointments
```

```
        self.__staff=staff

        # Calculate the total cost of the appointments using the getTotalCost method
of the Appointment class

        self.__totalCost = sum(appointment.getTotalCost() for appointment in
appointments)

        self.__DentalCompany=DentalCompany



    # Define a method to print the receipt

    def getReceipt(self):

        print("Dental Company:", self.__DentalCompany.getName())

        print(self.__staff.getBranch())

        print("Patient info:") #  print branch information

        print(self.__patient.displayInfo())







        total_cost = 0

         # Iterate through the appointments and print the appointment date and
services

        for appointment in self.__appointments:

            appointment_date = appointment.getAppointmentTime().strftime("%Y-%m-%d")

            print("Appointment Date:", appointment_date)

            print()



            print("Services:")

            print("Staff Name:",self.__staff.getFirstName(),
self.__staff.getLastName())

            # Iterate through the services and print the name and cost

            for service in appointment.getServices():

                service_cost = service.getCost()

                total_cost += service_cost

                print(f"{service.getServiceName()} .....................
{service_cost:.2f} AED" )
```

```
        # Calculate the VAT and grand total
        vat = total_cost * 0.05
        grand_total = total_cost + vat

        # Print the subtotal, VAT, and total cost
        print("Subtotal : ",(total_cost),"AED")
        print("VAT (5%): ", (vat),"AED")
        print("Total cost: ",(grand_total),"AED")

from datetime import datetime

def main():
   # Create dental company
   dental_company = DentalCompany('Bright Smiles')

   # Create a manager
   manager = Manager("Mohammed", "Al Shamsi", "056321234", Gender.MALE, "1 April
1964", "M01")
   manager2= Manager("Maryam","Al Mansoori","0553793878",Gender.FEMALE,"1 April
1964", "M02")
    # Create a branch and add it to the dental company
   branch1 = Branch("Dubai , Al Diyafa ", "2nd December St, Villa 123",
"043444197", manager)
   branch2 = Branch("Dubai , Deira"," Khalid Bin Al Waleed Rd ,Villa
34","0423435425",manager2)
   dental_company.add_branch(branch1)
   dental_company.add_branch(branch2)

   # Create services
```

```
    cleaning = Service("Cleaning", 150)

    implants = Service("Implants", 5000)

    crowns = Service("Crowns", 2000)

    fillings = Service("Fillings", 600)


    # Add services to the branch

    branch1.addService(cleaning)

    branch1.addService(implants)

    branch1.addService(crowns)

    branch1.addService(fillings)


    branch2.addService(cleaning)

    branch2.addService(implants)

    branch2.addService(crowns)

    branch2.addService(fillings)


    # Create staff members

    receptionist = Receptionist("Fatima", "Al Ali", "056294739", Gender.FEMALE, "20
May 1999", "R01")

    hygienist = Hygienist("Ahmed", "Al Mansoori", "0504535603", Gender.MALE, "20 Aug
1999", "H01")

    dentist = Dentist("Noor", "Al Khaja", "056826495073", Gender.FEMALE, "20 April
1999", "D01")

    dentist1 = Dentist("Ahmed", "Al Rais", "057291739389", Gender.MALE,
"2021-06-01", "D02")

    dentist2 = Dentist("Fatima", "Al Hashimi" , "05521312412", Gender.FEMALE,
"2021-06-01", "D03")




    # Associate staff members with the branch

    receptionist.setBranch(branch1)

    hygienist.setBranch(branch1)

    dentist.setBranch(branch1)

    dentist1.setBranch(branch2)
```

```
    dentist2.setBranch(branch2)



    # Add staff members to the branch

    branch1.addStaff(receptionist)

    branch1.addStaff(hygienist)

    branch1.addStaff(dentist)

    branch2.addStaff(dentist1)

    branch2.addStaff(dentist2)




    # create patients for different branches

    #branch 1

    patient1 = Patient("Khalid", "Al Suwaidi", "055445829", Gender.MALE, "20 May
1999", "P01")

    #branch2

    patient2 = Patient("Mariam", "Al Hashimi", "059299991", Gender.FEMALE, "3 May
1790", "P02")




    # Add patients to the branch

    branch1.addPatient(patient1)

    branch2.addPatient(patient2)


    # Create appointments

    appointment1 = Appointment([cleaning, fillings], datetime(2023, 4, 15, 10, 0))

    appointment2 = Appointment([crowns], datetime(2023, 4, 15, 11, 0))


    # Add appointments to patients

    patient1.bookAppointment(appointment1)

    patient2.bookAppointment(appointment2)


    # Create a receipt
```

```
    receipt = Receipt(patient1, [appointment1], dentist, dental_company)
    receipt2 = Receipt(patient2, [appointment2], dentist2, dental_company)


    # Print the receipt
    receipt.getReceipt()

    print()
    print()
    print()
    receipt2.getReceipt()

if __name__ == "__main__":
    main()
```

The output:

```
Dental Company: Bright Smiles
Branch Name: Dubai , Al Diyafa
Address: 2nd December St, Villa 123
Phone Number: 043444197
Manager: Mohammed Al Shamsi
Patient info:
Name:KhalidAl Suwaidi
Gender:MALE
Phone number055445829
Appointment Date: 2023-04-15

Services:
Staff Name: Noor Al Khaja
```

Cleaning ..... 150.00 AED

Fillings ..... 600.00 AED

Subtotal : 750 AED

VAT (5%): 37.5 AED

Total cost: 787.5 AED

Dental Company: Bright Smiles

Branch Name: Dubai , Deira

Address: Khalid Bin Al Waleed Rd ,Villa 34

Phone Number: 0423435425

Manager: Maryam Al Mansoori

Patient info:

Name:MariamAl Hashimi

Gender:FEMALE

Phone number059299991

Appointment Date: 2023-04-15

Services:

Staff Name: Fatima Al Hashimi

Crowns ..... 2000.00 AED

Subtotal : 2000 AED

VAT (5%): 100.0 AED

Total cost: 2100.0 AED

I have created a more organized, modular, and efficient codebase by making use of relationships such as association, composition, and inheritance. Let me elaborate on these connections and describe how they improve the code's readability and efficiency:

1. The relationship between Branch and DentalCompany is a composition relationship. The Branch class is a part of the DentalCompany class, and cannot exist without it. The DentalCompany class contains a list of Branch objects, which are created and added to the list using the add_branch method of the DentalCompany class. This means that when the DentalCompany object is deleted, all its associated Branch objects will also be deleted.

2. the relationship between Appointment and Service is a composition relationship. The Appointment class has an attribute called service, which is an instance of the Service class. This means that an Appointment object cannot exist without a Service object, and the Service object is considered to be a part of the Appointment object.The composition relationship is represented in the code through the use of the service attribute in the Appointment class, which is set when an Appointment object is created and cannot be changed afterwards. The service attribute is an instance of the Service class and is initialized with a default value of None, but can be set to a specific Service object using the set_service method.

3. The Branch and Manager classes have an aggregation relationship, which is a "has-a" relationship where one class contains objects of another class as a part of its state, but the contained object can exist without the container object. In the given code, the Branch class contains a list of Manager objects, which can exist without the Branch object.This is shown in the code by the add_manager method of the Branch class, which adds a manager to the list of managers: self.__managers.append(manager). The get_managers method of the Branch class returns the list of managers: return self.__managers. Therefore, we can say that the Branch "has-a" list of Managers as its state, and the Manager objects can exist without the Branch object.

4. The relationship between Branch and Staff is an aggregation relationship. The Branch class has a list of Staff objects, which are not considered as part of the Branch object itself, but rather as separate objects that are associated with the Branch object. This means that a Staff object can exist without a Branch object, and can be associated with different Branch objects at different times.The aggregation relationship is represented in the code

through the use of a list attribute in the Branch class, which can be modified using methods such as add_staff and remove_staff to add or remove Staff objects from the list.

5. The relationship between Branch and Patient is an aggregation relationship. A Branch object contains a collection of Patient objects, but the Patient objects can exist independently of the Branch object. Patients can be associated with multiple branches, and the deletion of a Branch object does not necessarily imply the deletion of its associated Patient objects.

6. Staff and Patient classes inherit from the Person class. This means that they have access to all the attributes and methods defined in the Person class. Additionally, the Manager, Receptionist, Dentist, and Hygienist classes inherit from the Staff class, which in turn inherits from the Person class. This allows these specialized subclasses to have all the attributes and methods of their parent classes, as well as any additional attributes and methods defined in their own class.

7. The Receipt class has a patient ,appointments,staff ,DentalCompany attribute that holds a reference to a Patient , appointments ,staff and DentalCompany objects. This relationship can be described as a "has-a" "composition" relationship .This means that a Receipt object is composed of or has references to objects from these classes to represent the details of a specific receipt, such as the patient who received the services, the appointments when the services were provided, the staff member who provided the services, and the dental company where the services were provided.

I find that composition relationships better represent real-world relationships and provide stronger "has-a" connections.Using Composition aids me in ensuring that my code is consistent and that my data is correct. Composition relationships also help me model "part-whole" relationships in the real world, which makes my code more accurate to life and enhances its quality. Inheritance offers additional benefits, such as code reusability and accurate modeling of real-world relationships. By allowing a subclass to inherit properties and methods from a superclass, I can encourage code reuse and reduce the need for duplication. Subclasses can inherit properties and

behaviors from their superclasses, just as real-world objects inherit properties and behaviors from their parents. This facilitates the creation of a more realistic and effective code structure.

About the code structure :

The code structure for the dental clinic management system is carefully designed with multiple classes to create a comprehensive and organized system using object-oriented programming. The code is based on a dental company and its various components, such as branches, staff members, patients, services, and appointments. The main classes include DentalCompany, Branch, Person, and their subclasses like Staff, Manager, Receptionist, Hygienist, Dentist, Patient, Appointment, Service, and Receipt. Composition and inheritance relationships between these classes ensure the code's modularity and reusability.

The Person class serves as a base class for Staff and Patient classes, streamlining the reuse of common attributes and methods. The Staff class is further extended by the Manager, Receptionist, Hygienist, and Dentist classes. The DentalCompany class maintains a list of branches, and the Branch class manages lists of services, patients, and staff members. This organization enables efficient access to relevant data and enhances system management and maintenance capabilities.

The Appointment and Service classes help track services provided in each appointment, while the Receipt class generates receipts for patients based on their appointments. The main function demonstrates how these classes work together to create a dental company with multiple branches, staff members, services, and patients. It also shows how appointments are created, scheduled, and processed to generate receipts for patients.

In the Branch class, a string function is added to return a string representation of the branch information, such as branch name, address, phone number, and manager. A displayInfo function is also added to the Patient class to return a string representation of the patient's basic information, such as name, gender, and phone number. These functions offer several advantages, such as providing clear, human-readable representations, ensuring consistency, offering flexibility, maintaining the principle of encapsulation, and reducing code duplication.

The relationships between the classes make the code more efficient and professional. To add a new patient, the code follows a sequential method, creating a patient, adding the patient to the desired branch, creating an appointment, adding appointments to patients, and finally creating a receipt object. This approach makes additions to appointments or patients easy.

The code utilizes encapsulation by making attributes private and accessed only through getter and setter methods, ensuring data integrity and hiding implementation details. Several test cases demonstrate the program's functionality, and the ability to print different receipts for different patients showcases its effectiveness in generating required information. This comprehensive dental clinic management system caters to various aspects of managing a dental practice, ensuring smooth and efficient operations.

During the process of making a management system for a dental clinic using object-oriented programming, I learned a number of important ideas and methods, including:

- I learned to design classes and objects that represent key components of the system, such as DentalCompany, Branch, Person, Staff, Manager, Receptionist, Hygienist, Dentist, Patient, Appointment, Service, and Receipt.

- I learned to implement inheritance by creating subclasses (e.g., Staff, Manager, Receptionist, Hygienist, Dentist) that inherit properties and methods from a parent class (e.g., Person).

- I learned to employ encapsulation, concealing a class's internal implementation details through private attributes (e.g., __name, __address, __phoneNumber) and offering public methods (getters and setters) to access and modify these attributes.

- I learned to establish composition and association relationships between classes, such as adding services, staff members, and patients to a Branch or linking appointments with patients.

- I learned to utilize enumeration (Enum) to define a collection of named values, like Gender in this case, which aids in assigning valid values to attributes.

- I learned to work with the datetime module for handling dates and times, such as creating and formatting appointment times.

- I learned to conduct calculations and manipulations with objects, like determining the total cost of appointments and producing receipts.

- I learned to create a main function (main()) to execute the code, showcasing the system's functionality by generating instances of the classes, setting their attributes, and invoking their methods. This function also tests the implemented features and verifies their proper operation.

- In conclusion, the experience of constructing a dental clinic management system using object-oriented programming principles has deepened my comprehension of these concepts and enhanced my capacity to apply them in practical situations.