

Hexaware Coding Challenge

Ramireddy Preethi

Topic: Insurance Management System

Problem Statement:

1. Create SQL Schema from the following classes class, use the class attributes for table column names.

```
CREATE DATABASE InsuranceManagementDB;
USE InsuranceManagementDB;

-- Create Users table
CREATE TABLE Users (
    userId INT PRIMARY KEY identity(101,1),
    username NVARCHAR(50) NOT NULL,
    password NVARCHAR(50) NOT NULL,
    role NVARCHAR(50) NOT NULL
);
INSERT INTO Users (username, password, role) VALUES
('preethi', 'password123', 'admin'),
('varun', 'varunpass', 'agent'),
('sreeja', 'sreeja2024', 'customer'),
('pooja', 'pooja@123', 'customer')

-- Create Clients table
CREATE TABLE Clients (
    clientId INT PRIMARY KEY identity(201,1),
    clientName NVARCHAR(100) NOT NULL,
    contactInfo NVARCHAR(100) NOT NULL,
    policy NVARCHAR(100) NOT NULL
);

INSERT INTO Clients(clientname, contactinfo, policy) VALUES
('John Doe', 'john.doe@email.com', 'Life Insurance'),
('Jane Smith', 'jane.smith@email.com', 'Auto Insurance'),
('Alice Johnson', '+1234567890', 'Health Insurance'),
('Robert Brown', 'robert.brown@email.com', 'Home Insurance')

-- Create Policies table
CREATE TABLE Policies (
    policyId INT PRIMARY KEY IDENTITY(1,1),
    policyName NVARCHAR(100) NOT NULL,
    policyDescription NVARCHAR(255) NOT NULL
);
-- Insert into Policies
INSERT INTO Policies (policyName, policyDescription)
VALUES ('Health Insurance', 'Covers medical expenses including hospital stays and treatments'),
```

```

        ('Life Insurance', 'Provides financial security to your family in case of your
death'),
        ('Auto Insurance', 'Covers damages to your car and third-party liability'),
        ('Home Insurance', 'Covers damages to your home and belongings');

```

-- Create Claims table

```

CREATE TABLE Claims (
    claimId INT PRIMARY KEY identity(301,1),
    claimNumber NVARCHAR(100) NOT NULL,
    dateFiled DATE NOT NULL,
    claimAmount DECIMAL(10, 2) NOT NULL,
    status NVARCHAR(50) NOT NULL,
    clientId INT,
    policy NVARCHAR(100),
    FOREIGN KEY (clientId) REFERENCES Clients(clientId)
);
INSERT INTO Claims (claimNumber, dateFiled, claimAmount, status, policy, clientId)
VALUES
('CLM2024001', '2024-10-01', 1500.00, 'Pending', 'Auto Insurance', 201),
('CLM2024002', '2024-10-05', 2500.00, 'Approved', 'Home Insurance', 202),
('CLM2024003', '2024-10-07', 350.00, 'Rejected', 'Health Insurance', 203),
('CLM2024004', '2024-10-12', 4200.00, 'Pending', 'Life Insurance', 204)

```

-- Create Payments table

```

CREATE TABLE Payments (
    paymentId INT PRIMARY KEY identity(401,1),
    paymentDate DATE NOT NULL,
    paymentAmount DECIMAL(10, 2) NOT NULL,
    clientId INT,
    FOREIGN KEY (clientId) REFERENCES Clients(clientId)
);
INSERT INTO Payments (paymentDate, paymentAmount, clientId) VALUES
('2024-10-01', 1500.00, 201), -- Payment for John Doe
('2024-10-02', 2500.00, 202), -- Payment for Jane Smith
('2024-10-03', 350.00, 203), -- Payment for Alice Johnson
('2024-10-04', 4200.00, 204)
SELECT * FROM Clients;
SELECT * FROM Users;
SELECT * FROM Claims;
SELECT * FROM Payments;
SELECT * FROM Policies;

```

1. Create the following model/entity classes within package entity with variables declared private, constructors (default and parametrized, getters, setters and toString())

2. Implement the following for all model classes. Write default constructors and overload the constructor with parameters, getters and setters, method to print all the member variables and values.

1. Define `User` class with the following confidential attributes:

- a. userId;
- b. username;

c. password;

d. role;

CODE:

entity/User.py

```
class User:
    def __init__(self, userId=None, username=None, password=None, role=None):
        self.__userId = userId
        self.__username = username
        self.__password = password
        self.__role = role

    # Getters and Setters
    def get_userId(self):
        return self.__userId

    def set_userId(self, userId):
        self.__userId = userId

    def get_username(self):
        return self.__username

    def set_username(self, username):
        self.__username = username

    def get_password(self):
        return self.__password

    def set_password(self, password):
        self.__password = password

    def get_role(self):
        return self.__role

    def set_role(self, role):
        self.__role = role

    def __str__(self):
        return f"User [userId={self.__userId}, username={self.__username},\nrole={self.__role}]"
```

2. Define `Client` class with the following confidential attributes:

a. clientId;

b. clientName;

c. contactInfo;

d. policy; //Represents the policy associated with the client

CODE:

entity/Client.py

```
class Client:
    def __init__(self, clientId=None, clientName=None, contactInfo=None,
policy=None):
        self.__clientId = clientId
        self.__clientName = clientName
        self.__contactInfo = contactInfo
        self.__policy = policy

    # Getters and Setters
    def get_clientId(self):
        return self.__clientId

    def set_clientId(self, clientId):
        self.__clientId = clientId

    def get_clientName(self):
        return self.__clientName

    def set_clientName(self, clientName):
        self.__clientName = clientName

    def get_contactInfo(self):
        return self.__contactInfo

    def set_contactInfo(self, contactInfo):
        self.__contactInfo = contactInfo

    def get_policy(self):
        return self.__policy

    def set_policy(self, policy):
        self.__policy = policy

    def __str__(self):
        return f"Client [clientId={self.__clientId},
clientName={self.__clientName}, contactInfo={self.__contactInfo},
policy={self.__policy}]"
```

3. Define `Claim` class with the following confidential attributes:

- a. claimId;
- b. claimNumber;
- c. dateFiled;
- d. claimAmount;
- e. status;

code:

entity/Claim.py

```
class Claim:
    def __init__(self, claimId=None, claimNumber=None, dateFiled=None,
claimAmount=None, status=None, policy=None, clientId=None):
        self.__claimId = claimId
        self.__claimNumber = claimNumber
        self.__dateFiled = dateFiled
        self.__claimAmount = claimAmount
        self.__status = status
        self.__policy = policy
        self.__client = clientId

    # Getters and Setters
    def get_claimId(self):
        return self.__claimId

    def set_claimId(self, claimId):
        self.__claimId = claimId

    def get_claimNumber(self):
        return self.__claimNumber

    def set_claimNumber(self, claimNumber):
        self.__claimNumber = claimNumber

    def get_dateFiled(self):
        return self.__dateFiled

    def set_dateFiled(self, dateFiled):
        self.__dateFiled = dateFiled

    def get_claimAmount(self):
```

```

        return self.__claimAmount

    def set_claimAmount(self, claimAmount):
        self.__claimAmount = claimAmount

    def get_status(self):
        return self.__status

    def set_status(self, status):
        self.__status = status

    def get_policy(self):
        return self.__policy

    def set_policy(self, policy):
        self.__policy = policy

    def get_client(self):
        return self.__client

    def set_client(self, clientId):
        self.__client = clientId

    # String representation
    def __str__(self):
        return (f"Claim [claimId={self.__claimId},
claimNumber={self.__claimNumber}, dateFiled={self.__dateFiled}, "
                f"claimAmount={self.__claimAmount}, status={self.__status},
policy={self.__policy}, client={self.__client}]")

```

4. Define `payment` class with the following confidential attributes:

- a. paymentId;
- b. paymentDate;
- c. paymentAmount;
- d. client; // Represents the client associated with the payment

code:
entity/payment.py

```

class Payment:
    def __init__(self, paymentId=None, paymentDate=None, paymentAmount=None,
client=None):
        self.__paymentId = paymentId
        self.__paymentDate = paymentDate

```

```

        self.__paymentAmount = paymentAmount
        self.__client = client

# Getters and Setters
def get_paymentId(self):
    return self.__paymentId

def set_paymentId(self, paymentId):
    self.__paymentId = paymentId

def get_paymentDate(self):
    return self.__paymentDate

def set_paymentDate(self, paymentDate):
    self.__paymentDate = paymentDate

def get_paymentAmount(self):
    return self.__paymentAmount

def set_paymentAmount(self, paymentAmount):
    self.__paymentAmount = paymentAmount

def get_client(self):
    return self.__client

def set_client(self, client):
    self.__client = client

# String representation
def __str__(self):
    return (f"Payment [paymentId={self.__paymentId},
paymentDate={self.__paymentDate}, "
          f"paymentAmount={self.__paymentAmount},
client={self.__client}]")

```

3. Define IPolicyService interface/abstract class with following methods to interact with database

Keep the interfaces and implementation classes in package dao

a. createPolicy()

- I. parameters: Policy Object
- II. return type: Boolean

b. getPolicy()

- I. parameters: policyId
- II. return type: Policy Object

c. getAllPolicies()

- I. parameters: none
- II. return type: Collection of Policy Objects

d. updatePolicy()

- I. parameters: Policy Object
- II. return type: boolean

e. deletePolicy()

- I. parameters: PolicyId
- II. return type: boolean

Code:

dao/IPolicyService.py

```
from abc import ABC, abstractmethod

class IPolicyService(ABC):
    @abstractmethod
    def createPolicy(self, policy):
        pass

    @abstractmethod
    def getPolicy(self, policyId):
        pass

    @abstractmethod
    def getAllPolicies(self):
        pass

    @abstractmethod
    def updatePolicy(self, policy):
        pass
```



```

@abstractmethod
def deletePolicy(self, policyId):
    pass

```

6. Define InsuranceServiceImpl class and implement all the methods InsuranceServiceImpl

Code:

dao/PolicyServiceImpl.py

```

# dao/PolicyServiceImpl.py
import pyodbc
from src.dao.IPolicyService import IPolicyService
from src.entity.Policy import Policy
from src.exception.PolicyNotFoundException import PolicyNotFoundException
from src.util.DBConnUtil import DBConnUtil

class PolicyServiceImpl(IPolicyService):
    def __init__(self):
        self.conn = DBConnUtil.get_connection()

    def createPolicy(self, policy):
        cursor = self.conn.cursor()
        query = "INSERT INTO Policies (policyName, policyDescription) VALUES"
        (?, ?)"
        cursor.execute(query, policy.get_policyName(),
            policy.get_policyDescription())
        self.conn.commit()
        return True

    def getPolicy(self, policyId):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Policies WHERE policyId = ?"
        cursor.execute(query, policyId)
        result = cursor.fetchone()
        if result:
            return Policy(policyId=result.policyId,
                policyName=result.policyName, policyDescription=result.policyDescription)
        else:
            raise PolicyNotFoundException(f"Policy with ID {policyId} not found.")

    def getAllPolicies(self):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Policies"
        cursor.execute(query)

```

```

        policies = []
        for row in cursor.fetchall():
            policy = Policy(policyId=row.policyId, policyName=row.policyName,
policyDescription=row.policyDescription)
            policies.append(policy)
        return policies

    def updatePolicy(self, policy):
        cursor = self.conn.cursor()
        query = "UPDATE Policies SET policyName = ?, policyDescription = ?
WHERE policyId = ?"
        cursor.execute(query, policy.get_policyName(),
policy.get_policyDescription(), policy.get_policyId())
        self.conn.commit()
        return True

    def deletePolicy(self, policyId):
        cursor = self.conn.cursor()
        query = "DELETE FROM Policies WHERE policyId = ?"
        cursor.execute(query, policyId)
        self.conn.commit()
        return True

```

Outputs:

1) CREATE POLICY OUTPUT:

```

Enter your choice: 2
Enter policy name: education insurance
Enter policy description: Get Insurance for poor people
Policy created successfully!

```

BEFORE CREATING POLICY:

Results		Messages	
	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Indian Home Policy	Get Insurance if your house is affected in disaster!

AFTER CREATING POLICY:

Results Messages			
	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Indian Home Policy	Get Insurance if your house is affected in disaster!
6	6	education insurance	Get Insurance for poor people

2) GET POLICY OUTPUT

```
Enter your choice: 3
Enter policy ID: 1
Policy [policyId=1, policyName=Health Insurance, policyDescription=Covers medical expenses including hospital stays and treatments]
```

3) GET ALL POLICIES OUTPUT

```
Enter your choice: 4
Policy [policyId=1, policyName=Health Insurance, policyDescription=Covers medical expenses including hospital stays and treatments]
Policy [policyId=2, policyName=Life Insurance, policyDescription=Provides financial security to your family in case of your death]
Policy [policyId=3, policyName=Auto Insurance, policyDescription=Covers damages to your car and third-party liability]
Policy [policyId=4, policyName=Home Insurance, policyDescription=Covers damages to your home and belongings]
Policy [policyId=5, policyName=Indian Home Policy, policyDescription=Get Insurance if your house is affected in disaster!]
Policy [policyId=6, policyName=education insurance, policyDescription=Get Insurance for poor people]
```

4) UPDATE POLICY OUTPUT

```
Enter your choice: 5
Enter policy ID: 5
Enter new policy name: Travel Insurance
Enter new policy description: covers expenses and damage related
to travelling
Policy updated successfully!
```

BEFORE UPDATING POLICY:

Results Messages			
	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Indian Home Policy	Get Insurance if your house is affected in disaster!
6	6	education insurance	Get Insurance for poor people

AFTER UPDATING POLICY: (updated 'Indian home policy' to 'Travel Insurance')

Results Messages			
	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Travel Insurance	covers expenses and damage related to travelling
6	6	education insurance	Get Insurance for poor people

5) DELETE POLICY:

```
Enter your choice: 6
Enter policy ID: 6
Policy deleted successfully!
```

BEFORE DELETING POLICY:

Results Messages			
	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Travel Insurance	covers expenses and damage related to travelling
6	6	education insurance	Get Insurance for poor people

AFTER DELETING POLICY: (Deleted 'education insurance')

Results Messages			
	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Travel Insurance	covers expenses and damage related to travelling

Note: I have implemented user, claim, client, payment also, just for checking purpose but they just asked policy implementation.

dao/UserServiceImpl.py

```
# dao/UserServiceImpl.py
import pyodbc
from src.entity.User import User
from src.util.DBConnUtil import DBConnUtil

class UserServiceImpl:
    def __init__(self):
        self.conn = DBConnUtil.get_connection()

    def createUser(self, user):
        cursor = self.conn.cursor()
        query = "INSERT INTO Users (username, password, role) VALUES (?, ?, ?)"
        cursor.execute(query, user.get_username(), user.get_password(), user.get_role())
        self.conn.commit()
        return True

    def getUser(self, userId):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Users WHERE userId = ?"
        cursor.execute(query, userId)
        result = cursor.fetchone()
        if result:
            return User(userId=result.userId, username=result.username, password=result.password, role=result.role)
        else:
            return None

    def getAllUsers(self):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Users"
        cursor.execute(query)
        users = []
        for row in cursor.fetchall():
            user = User(userId=row.userId, username=row.username, password=row.password, role=row.role)
            users.append(user)
        return users

    def updateUser(self, user):
        cursor = self.conn.cursor()
        query = "UPDATE Users SET username = ?, password = ?, role = ? WHERE userId = ?"
```

```

        cursor.execute(query, user.get_username(), user.get_password(),
user.get_role(), user.get_userId())
        self.conn.commit()
        return True

def deleteUser(self, userId):
    cursor = self.conn.cursor()
    query = "DELETE FROM Users WHERE userId = ?"
    cursor.execute(query, userId)
    self.conn.commit()
    return True

```

1) CREATE USER OUTPUT:

```

Enter your choice: 7
Enter username: priynka
Enter password: priya@123
Enter role (admin/user): user
User created successfully!

```

BEFORE CREATING USER:

	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	sreeja	sreeja2024	customer
4	104	pooja	pooja@123	customer

AFTER CREATING USER: (Created user 'priyanka')

	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	sreeja	sreeja2024	customer
4	104	pooja	pooja@123	customer
5	105	priynka	priya@123	user

2) GET USER OUTPUT:

```
Enter your choice: 8
Enter user ID: 101
User [userId=101, username=preethi, role=admin]
```

3) GET ALL USERS:

```
Enter your choice: 9
User [userId=101, username=preethi, role=admin]
User [userId=102, username=varun, role=agent]
User [userId=103, username=sreeja, role=customer]
User [userId=104, username=pooja, role=customer]
User [userId=105, username=priynka, role=user]
```

4) UPDATE USER:

```
Enter your choice: 10
Enter user ID: 103
Enter new username: naini
Enter new password: naini@12345
Enter new role (admin/user): admin
User updated successfully!
```

BEFORE UPDATING:

Results		Messages		
	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	sreeja	sreeja2024	customer
4	104	pooja	pooja@123	customer
5	105	priynka	priya@123	user

AFTER UPDATING: (Updated 'sreeja' to 'naini')

	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	naini	naini@12345	admin
4	104	pooja	pooja@123	customer
5	105	priynka	priya@123	user

5) DELETE USER:

```
Insurance Management System
1. Create Client
2. Create Policy
3. Get Policy
4. Get All Policies
5. Update Policy
6. Delete Policy
7. Create User
8. Get User
9. Get All Users
10. Update User
11. Delete User
12. Create Claim
13. Get Claim
14. Get All Claims
15. Create Payment
16. Get Payment
17. Exit
Enter your choice: 11
Enter user ID: 105
User deleted successfully!
```

BEFORE DELETING USER:

	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	naini	naini@12345	admin
4	104	pooja	pooja@123	customer
5	105	priynka	priya@123	user

AFTER DELETING USER: (Deleting 'priyanka')

Results		Messages		
	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	naini	naini@12345	admin
4	104	pooja	pooja@123	customer

dao/ClaimServiceImpl.py

```
# dao/ClaimServiceImpl.py
import pyodbc
from src.entity.Claim import Claim
from src.util.DBConnUtil import DBConnUtil

class ClaimServiceImpl:
    def __init__(self):
        self.conn = DBConnUtil.get_connection()

    def createClaim(self, claim):
        cursor = self.conn.cursor()
        query = "INSERT INTO Claims (claimNumber, dateFiled, claimAmount, status, clientId, policy) VALUES (?, ?, ?, ?, ?, ?)"
        cursor.execute(query, claim.get_claimNumber(), claim.get_dateFiled(), claim.get_claimAmount(), claim.get_status(), claim.get_client(), claim.get_policy())
        self.conn.commit()
        return True

    def getClaim(self, claimId):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Claims WHERE claimId = ?"
        cursor.execute(query, claimId)
        result = cursor.fetchone()
        if result:
            return Claim(claimId=result.claimId, claimNumber=result.claimNumber, dateFiled=result.dateFiled, claimAmount=result.claimAmount, status=result.status, clientId=result.clientId, policy=result.policy)
        else:
            return None

    def getAllClaims(self):
        cursor = self.conn.cursor()
```

```

query = "SELECT * FROM Claims"
cursor.execute(query)
claims = []
for row in cursor.fetchall():
    claim = Claim(claimId=row.claimId, claimNumber=row.claimNumber,
dateFiled=row.dateFiled, claimAmount=row.claimAmount, status=row.status,
clientId=row.clientId, policy=row.policy)
    claims.append(claim)
return claims

def updateClaim(self, claim):
    cursor = self.conn.cursor()
    query = "UPDATE Claims SET claimNumber = ?, dateFiled = ?, claimAmount
= ?, status = ?, clientId = ?, policy = ? WHERE claimId = ?"
    cursor.execute(query, claim.get_claimNumber(), claim.get_dateFiled(),
claim.get_claimAmount(), claim.get_status(), claim.get_clientId(),
claim.get_policy(), claim.get_claimId())
    self.conn.commit()
    return True

def deleteClaim(self, claimId):
    cursor = self.conn.cursor()
    query = "DELETE FROM Claims WHERE claimId = ?"
    cursor.execute(query, claimId)
    self.conn.commit()
    return True

```

Outputs: (In coding challenge they only asked to execute policy creation, get policy, get all policies, update policy, delete policy but also I executed some claim options)

1) CREATE CLAIM:

```

Enter your choice: 12
Enter claim number: 2024005
Enter date filed (YYYY-MM-DD): 2024-10-10
Enter claim amount: 2000
Enter status: approved
Enter associated policy: Health Insurance
Enter associated client: 205
Claim created successfully!

```

BEFORE CREATING CLAIM:

Results		Messages					
	claimId	claimNumber	dateFiled	claimAmount	status	clientId	policy
1	301	CLM2024001	2024-10-01	1500.00	Pending	201	Auto Insurance
2	302	CLM2024002	2024-10-05	2500.00	Approved	202	Home Insurance
3	303	CLM2024003	2024-10-07	350.00	Rejected	203	Health Insurance
4	304	CLM2024004	2024-10-12	4200.00	Pending	204	Life Insurance

AFTER CREATING CLAIM:

Results		Messages					
	claimId	claimNumber	dateFiled	claimAmount	status	clientId	policy
1	301	CLM2024001	2024-10-01	1500.00	Pending	201	Auto Insurance
2	302	CLM2024002	2024-10-05	2500.00	Approved	202	Home Insurance
3	303	CLM2024003	2024-10-07	350.00	Rejected	203	Health Insurance
4	304	CLM2024004	2024-10-12	4200.00	Pending	204	Life Insurance
5	305	2024005	2024-10-10	2000.00	approved	205	Health Insurance

2) GET CLAIM:

```
Enter your choice: 13
Enter claim ID: 301
Claim [claimId=301, claimNumber=CLM2024001, dateFiled=2024-10-01, claimAmount=1500.00, status=Pending, policy=Auto Insurance, client=201]
```

3) GET ALL CLAIMS:

```
enter your choice: 14
Claim [claimId=301, claimNumber=CLM2024001, dateFiled=2024-10-01, claimAmount=1500.00, status=Pending, policy=Auto Insurance, client=201]
Claim [claimId=302, claimNumber=CLM2024002, dateFiled=2024-10-05, claimAmount=2500.00, status=Approved, policy=Home Insurance, client=202]
Claim [claimId=303, claimNumber=CLM2024003, dateFiled=2024-10-07, claimAmount=350.00, status=Rejected, policy=Health Insurance, client=203]
Claim [claimId=304, claimNumber=CLM2024004, dateFiled=2024-10-12, claimAmount=4200.00, status=Pending, policy=Life Insurance, client=204]
Claim [claimId=305, claimNumber=2024005, dateFiled=2024-10-10, claimAmount=2000.00, status=approved, policy=Health Insurance, client=205]
```

dao/ClientServiceImpl.py

```
# dao/ClientServiceImpl.py
import pyodbc
from src.entity.Client import Client
from src.util.DBConnUtil import DBConnUtil

class ClientServiceImpl:
    def __init__(self):
        self.conn = DBConnUtil.get_connection()

    def createClient(self, client):
        cursor = self.conn.cursor()
        query = "INSERT INTO Clients (clientName, contactInfo, policy) VALUES"
        (? , ? , ?)"
        cursor.execute(query, client.get_clientName(),
client.get_contactInfo(), client.get_policy())
        self.conn.commit()
        return True

    def getClient(self, clientId):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Clients WHERE clientId = ?"
        cursor.execute(query, clientId)
        result = cursor.fetchone()
        if result:
            return Client(clientId=result.clientId,
clientName=result.clientName, contactInfo=result.contactInfo,
policy=result.policy)
        else:
            return None

    def getAllClients(self):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Clients"
        cursor.execute(query)
        clients = []
        for row in cursor.fetchall():
            client = Client(clientId=row.clientId, clientName=row.clientName,
contactInfo=row.contactInfo, policy=row.policy)
            clients.append(client)
        return clients

    def updateClient(self, client):
        cursor = self.conn.cursor()
        query = "UPDATE Clients SET clientName = ?, contactInfo = ?, policy =
? WHERE clientId = ?"
        cursor.execute(query, client.get_clientName(),
client.get_contactInfo(), client.get_policy(), client.get_clientId())
        self.conn.commit()
```

```

        return True

def deleteClient(self, clientId):
    cursor = self.conn.cursor()
    query = "DELETE FROM Clients WHERE clientId = ?"
    cursor.execute(query, clientId)
    self.conn.commit()
    return True

```

Output: (In coding challenge they only asked to execute policy creation, get policy, get all policies, update policy, delete policy but also I executed some client options)

Create client:

```

Enter your choice: 1
Enter client name: priya
Enter contact info: 1234567890
Enter policy: health Insurance
client created successfully!

```

Before creating client:

	clientId	clientName	contactInfo	policy
1	201	John Doe	john.doe@email.com	Life Insurance
2	202	Jane Smith	jane.smith@email.com	Auto Insurance
3	203	Alice Johnson	+1234567890	Health Insurance
4	204	Robert Brown	robert.brown@email.com	Home Insurance

After creating client:

	clientId	clientName	contactInfo	policy
1	201	John Doe	john.doe@email.com	Life Insurance
2	202	Jane Smith	jane.smith@email.com	Auto Insurance
3	203	Alice Johnson	+1234567890	Health Insurance
4	204	Robert Brown	robert.brown@email.com	Home Insurance
5	205	priya	1234567890	health Insurance

dao/PaymentServiceImpl.py

```
# dao/PaymentServiceImpl.py
import pyodbc
from src.entity.Payment import Payment
from src.util.DBConnUtil import DBConnUtil

class PaymentServiceImpl:
    def __init__(self):
        self.conn = DBConnUtil.get_connection()

    def createPayment(self, payment):
        cursor = self.conn.cursor()
        query = "INSERT INTO Payments (paymentDate, paymentAmount, clientId)
VALUES (?, ?, ?)"
        cursor.execute(query, payment.get_paymentDate(),
payment.get_paymentAmount(), payment.get_client())
        self.conn.commit()
        return True

    def getPayment(self, paymentId):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Payments WHERE paymentId = ?"
        cursor.execute(query, paymentId)
        result = cursor.fetchone()
        if result:
            return Payment(paymentId=result.paymentId,
paymentDate=result.paymentDate, paymentAmount=result.paymentAmount,
clientId=result.clientId)
        else:
            return None

    def getAllPayments(self):
        cursor = self.conn.cursor()
        query = "SELECT * FROM Payments"
        cursor.execute(query)
        payments = []
        for row in cursor.fetchall():
            payment = Payment(paymentId=row.paymentId,
paymentDate=row.paymentDate, paymentAmount=row.paymentAmount,
clientId=row.clientId)
            payments.append(payment)
        return payments

    def updatePayment(self, payment):
        cursor = self.conn.cursor()
        query = "UPDATE Payments SET paymentDate = ?, paymentAmount = ?,
clientId = ? WHERE paymentId = ?"
        cursor.execute(query, payment.get_paymentDate(),
payment.get_paymentAmount(), payment.get_client(), payment.get_paymentId())
```

```

        self.conn.commit()
        return True

    def deletePayment(self, paymentId):
        cursor = self.conn.cursor()
        query = "DELETE FROM Payments WHERE paymentId = ?"
        cursor.execute(query, paymentId)
        self.conn.commit()
        return True

```

Output: (In coding challenge they only asked to execute policy creation, get policy, get all policies, update policy, delete policy but also I executed some payment options)

```

Enter your choice: 15
Enter payment date (YYYY-MM-DD): 2024-10-13
Enter payment amount: 2400
Enter associated client: 205
Payment created successfully!

```

Before creating payment:

	paymentId	paymentDate	paymentAmount	clientId
1	401	2024-10-01	1500.00	201
2	402	2024-10-02	2500.00	202
3	403	2024-10-03	350.00	203
4	404	2024-10-04	4200.00	204

After creating payment:

	paymentId	paymentDate	paymentAmount	clientId
1	401	2024-10-01	1500.00	201
2	402	2024-10-02	2500.00	202
3	403	2024-10-03	350.00	203
4	404	2024-10-04	4200.00	204
5	405	2024-10-13	2400.00	205

Get payment:

```
Enter your choice: 16
Enter payment ID: 403
Payment [paymentId=403, paymentDate=2024-10-03, paymentAmount=350.00, client=203]
```

Exit:

```
Enter your choice: 17
Exiting...
```

7. Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection. Connection properties supplied in the connection string should be read from a property file. Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

src/util/DBConnUtil.py (both classes or implemented in one file)

```
# src/util/DBConnUtil.py
import pyodbc

class DBConnUtil:
    @staticmethod
    def get_connection():
        # Connect to the new database InsuranceDB1
        connection_string = (
            "DRIVER={ODBC Driver 17 for SQL Server};"
            "SERVER=LAPTOP-2JD3H0E6;"
            "DATABASE=InsuranceManagementDB;"
            "Trusted_Connection=yes;"
        )
        return pyodbc.connect(connection_string)
```

8. Create the exceptions in package myexceptions Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

1. PolicyNotFoundException :throw this exception when user enters an invalid patient number which doesn't exist in db

Code:

exception/PolicyNotFoundException.py

```
# exception/PolicyNotFoundException.py
class PolicyNotFoundException(Exception):
    def __init__(self, message):
        super().__init__(message)
```

9. Create class named MainModule with main method in package mainmod. Trigger all the methods in service implementation class.

app/ MainModule.py

```
from src.dao.PolicyServiceImpl import PolicyServiceImpl
from src.dao.ClientServiceImpl import ClientServiceImpl
from src.dao.ClaimServiceImpl import ClaimServiceImpl
from src.dao.UserServiceImpl import UserServiceImpl
from src.dao.PaymentServiceImpl import PaymentServiceImpl
from src.entity.Policy import Policy
from src.entity.Client import Client
from src.entity.Claim import Claim
from src.entity.User import User
from src.entity.Payment import Payment
from src.exception.PolicyNotFoundException import PolicyNotFoundException

if __name__ == "__main__":
    # Create instances of service classes
    policy_service = PolicyServiceImpl()
    client_service = ClientServiceImpl()
    claim_service = ClaimServiceImpl()
    user_service = UserServiceImpl()
    payment_service = PaymentServiceImpl()

    while True:
        print("\nInsurance Management System")
        print("1. Create Client")
        print("2. Create Policy")
        print("3. Get Policy")
        print("4. Get All Policies")
        print("5. Update Policy")
        print("6. Delete Policy")
        print("7. Create User")
        print("8. Get User")
        print("9. Get All Users")
```

```

print("10. Update User")
print("11. Delete User")
print("12. Create Claim")
print("13. Get Claim")
print("14. Get All Claims")
print("15. Create Payment")
print("16. Get Payment")
print("17. Exit")

choice = input("Enter your choice: ")

if choice == '1': # Create Client

    clientName = input("Enter client name: ")
    contactInfo = input("Enter contact info: ")
    policy = input("Enter policy: ")
    client = Client(clientName=clientName, contactInfo=contactInfo,
policy=policy)
    client_service.createClient(client)
    print("Client created successfully!")

elif choice == '2': # Create Policy
    policyName = input("Enter policy name: ")
    policyDescription = input("Enter policy description: ")
    policy = Policy(policyName=policyName,
policyDescription=policyDescription)
    policy_service.createPolicy(policy)
    print("Policy created successfully!")

elif choice == '3': # Get Policy
    policyId = int(input("Enter policy ID: "))
    try:
        policy = policy_service.getPolicy(policyId)
        print(policy)
    except PolicyNotFoundException as e:
        print(e)

elif choice == '4': # Get All Policies
    policies = policy_service.getAllPolicies()
    for policy in policies:
        print(policy)

elif choice == '5': # Update Policy
    policyId = int(input("Enter policy ID: "))
    policyName = input("Enter new policy name: ")
    policyDescription = input("Enter new policy description: ")
    policy = Policy(policyId=policyId, policyName=policyName,
policyDescription=policyDescription)

```

```

        policy_service.updatePolicy(policy)
        print("Policy updated successfully!")

    elif choice == '6': # Delete Policy
        policyId = int(input("Enter policy ID: "))
        policy_service.deletePolicy(policyId)
        print("Policy deleted successfully!")

    elif choice == '7': # Create User
        username = input("Enter username: ")
        password = input("Enter password: ")
        role = input("Enter role (admin/user): ")
        user = User(username=username, password=password, role=role)
        user_service.createUser(user)
        print("User created successfully!")

    elif choice == '8': # Get User
        userId = int(input("Enter user ID: "))
        user = user_service.getUser(userId)
        if user:
            print(user)
        else:
            print("User not found.")

    elif choice == '9': # Get All Users
        users = user_service.getAllUsers()
        for user in users:
            print(user)

    elif choice == '10': # Update User
        userId = int(input("Enter user ID: "))
        username = input("Enter new username: ")
        password = input("Enter new password: ")
        role = input("Enter new role (admin/user): ")
        user = User(userId=userId, username=username, password=password,
role=role)
        user_service.updateUser(user)
        print("User updated successfully!")

    elif choice == '11': # Delete User
        userId = int(input("Enter user ID: "))
        user_service.deleteUser(userId)
        print("User deleted successfully!")

    elif choice == '12': # Create Claim
        claimNumber = input("Enter claim number: ")
        dateFiled = input("Enter date filed (YYYY-MM-DD): ")
        claimAmount = float(input("Enter claim amount: "))

```

```

        status = input("Enter status: ")
        policy = input("Enter associated policy: ")
        clientId = input("Enter associated client: ")
        claim = Claim(claimNumber=claimNumber, dateFiled=dateFiled,
claimAmount=claimAmount, status=status,
                        policy=policy, clientId=clientId)
        claim_service.createClaim(claim)
        print("Claim created successfully!")

    elif choice == '13': # Get Claim
        claimId = int(input("Enter claim ID: "))
        claim = claim_service.getClaim(claimId)
        if claim:
            print(claim)
        else:
            print("Claim not found.")

    elif choice == '14': # Get All Claims
        claims = claim_service.getAllClaims()
        for claim in claims:
            print(claim)

    elif choice == '15': # Create Payment
        paymentDate = input("Enter payment date (YYYY-MM-DD): ")
        paymentAmount = float(input("Enter payment amount: "))
        client = input("Enter associated client: ")
        payment = Payment(paymentDate=paymentDate,
paymentAmount=paymentAmount, client=client)
        payment_service.createPayment(payment)
        print("Payment created successfully!")

    elif choice == '16': # Get Payment
        paymentId = int(input("Enter payment ID: "))
        payment = payment_service.getPayment(paymentId)
        if payment:
            print(payment)
        else:
            print("Payment not found.")

    elif choice == '17': # Exit
        print("Exiting...")
        break
    else:
        print("Invalid choice! Please try again.")

```

Output:

```
Insurance Management System
1. Create Client
2. Create Policy
3. Get Policy
4. Get All Policies
5. Update Policy
6. Delete Policy
7. Create User
8. Get User
9. Get All Users
10. Update User
11. Delete User
12. Create Claim
13. Get Claim
14. Get All Claims
15. Create Payment
16. Get Payment
17. Exit
Enter your choice: |
```

Entire Database after execution:

Results

Messages

	clientId	clientName	contactInfo	policy
1	201	John Doe	john.doe@email.com	Life Insurance
2	202	Jane Smith	jane.smith@email.com	Auto Insurance
3	203	Alice Johnson	+1234567890	Health Insurance
4	204	Robert Brown	robert.brown@email.com	Home Insurance
5	205	priya	1234567890	health Insurance

	userId	username	password	role
1	101	preethi	password123	admin
2	102	varun	varunpass	agent
3	103	naini	naini@12345	admin
4	104	pooja	pooja@123	customer

	claimId	claimNumber	dateFiled	claimAmount	status	clientId	policy
1	301	CLM2024001	2024-10-01	1500.00	Pending	201	Auto Insurance
2	302	CLM2024002	2024-10-05	2500.00	Approved	202	Home Insurance
3	303	CLM2024003	2024-10-07	350.00	Rejected	203	Health Insuran...
4	304	CLM2024004	2024-10-12	4200.00	Pending	204	Life Insurance
5	305	2024005	2024-10-10	2000.00	approved	205	Health Insuran...

	paymentId	paymentDate	paymentAmount	clientId
1	401	2024-10-01	1500.00	201
2	402	2024-10-02	2500.00	202
3	403	2024-10-03	350.00	203
4	404	2024-10-04	4200.00	204
5	405	2024-10-13	2400.00	205

	policyId	policyName	policyDescription
1	1	Health Insurance	Covers medical expenses including hospital stays...
2	2	Life Insurance	Provides financial security to your family in case of...
3	3	Auto Insurance	Covers damages to your car and third-party liability
4	4	Home Insurance	Covers damages to your home and belongings
5	5	Travel Insurance	covers expenses and damage related to travelling