Hexaware Coding Challenge

Done By: Eswara Venkata Sai Raja

Topic: Insurance Management System

Problem Statement:

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

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

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

a. userId;

b. username;

c. password;

d. role;

CODE:

```
  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},
role={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:

```
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;

f. policy;//Represents the policy associated with the claim

g. client; // Represents the client associated with the claim

CODE:

```
class Claim:

  def __init__(self, claimId=None, claimNumber=None, dateFiled=None, claimAmount=None,
status=None, clientId=None, policy=None):

            self.__claimId = claimId

            self.__claimNumber = claimNumber

            self.__dateFiled = dateFiled

            self.__claimAmount = claimAmount

            self.__status = status

            self.__clientId = clientId

            self.__policy = policy
```

```
        # 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_clientId(self):
    return self.__clientId
```

def set_clientId(self, clientId):

self.__clientId = clientId

def get_policy(self):

return self.__policy

def set_policy(self, policy):

self.__policy = policy

def __str__(self):

return f"Claim [claimId={self.__claimId}, claimNumber={self.__claimNumber}, dateFiled={self.__dateFiled}, claimAmount={self.__claimAmount}, status={self.__status}]"

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:

```
      class Payment:
    def __init__(self, paymentId=None, paymentDate=None, paymentAmount=None, clientId=None):
        self.__paymentId = paymentId
        self.__paymentDate = paymentDate
        self.__paymentAmount = paymentAmount
        self.__clientId = clientId

    # 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_clientId(self):

            return self.__clientId

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

    def __str__(self):
        return f"Payment [paymentId={self.__paymentId},
paymentDate={self.__paymentDate}, paymentAmount={self.__paymentAmount}]"
```

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 I. parameters: Policy Object

II II. return type: boolean

b. getPolicy()

I I. parameters: policyId

II II. return type: Policy Object

c.getAllPolicies()

I I. parameters: none

II II. return type: Collection of Policy Objects

```
def getAllPolicies(self):  1 usage
    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
```

d.updatePolicy()

I I. parameters: Policy Object

II II. return type: boolean

```
def updatePolicy(self, policy):  1 usage
    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
```

. deletePolicy()

I I. parameters: PolicyId

II II. return type: boolean

```
def deletePolicy(self, policyId):  1 usage
    cursor = self.conn.cursor()
    query = "DELETE FROM Policies WHERE policyId = ?"
    cursor.execute(query, policyId)
    self.conn.commit()
    return True
```

6. Define InsuranceServiceImpl class and implement all the methods InsuranceServiceImpl .

CODE:

```
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_clientId(), 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
```

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 fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

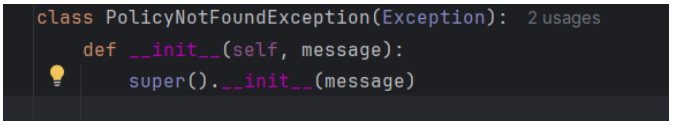
```python
import pyodbc

class DBConnUtil:  6 usages
    @staticmethod  3 usages
    def get_connection():
        # Connect to the new database InsuranceDB1
        connection_string = (
            "DRIVER={ODBC Driver 17 for SQL Server};"
            "SERVER=ESWARAVENKATASA\\SQLEXPRESS;"
            "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



9. Create class named MainModule with main method in package mainmod.

Trigger all the methods in service implementation class.

CODE:

```
from src.dao.PolicyServiceImpl import PolicyServiceImpl
from src.dao.ClientServiceImpl import ClientServiceImpl
from src.dao.ClaimServiceImpl import ClaimServiceImpl
from src.entity.Policy import Policy
from src.entity.Client import Client
from src.entity.Claim import Claim

if __name__ == "__main__":
    policy_service = PolicyServiceImpl()
    client_service = ClientServiceImpl()
    claim_service = ClaimServiceImpl()

    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. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            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':

            policyName = input("Enter policy name: ")

            policyDescription = input("Enter policy description: ")

            # Create a new Policy instance using the parameters
```

```
        policy = Policy(policyName=policyName, policyDescription=policyDescription)

        policy_service.createPolicy(policy)

        print("Policy created successfully!")
    elif choice == '3':
        policyId = int(input("Enter policy ID: "))
        try:
            policy = policy_service.getPolicy(policyId)
            print(policy)
        except Exception as e:
            print(e)

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

    elif choice == '5':
        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':
        policyId = int(input("Enter policy ID: "))
        policy_service.deletePolicy(policyId)
        print("Policy deleted successfully!")

    elif choice == '7':
        print("Exiting...")
        break
    else:
        print("Invalid choice! Please try again.")
```

OUTPUTS

```
CREATE DATABASE InsuranceManagementDB;

USE InsuranceManagementDB;


-- Create Users table

CREATE TABLE Users (

    userId INT PRIMARY KEY IDENTITY(1,1),

    username NVARCHAR(50) NOT NULL,

    password NVARCHAR(50) NOT NULL,

    role NVARCHAR(50) NOT NULL

);
```

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

-- Create Policies table
CREATE TABLE Policies (
    policyId INT PRIMARY KEY IDENTITY(1,1),
    policyName NVARCHAR(100) NOT NULL,
    policyDescription NVARCHAR(255) NOT NULL
);

-- Create Claims table
CREATE TABLE Claims (
    claimId INT PRIMARY KEY IDENTITY(1,1),
    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)
);

-- Create Payments table
CREATE TABLE Payments (
    paymentId INT PRIMARY KEY IDENTITY(1,1),
    paymentDate DATE NOT NULL,
    paymentAmount DECIMAL(10, 2) NOT NULL,
    clientId INT,
    FOREIGN KEY (clientId) REFERENCES Clients(clientId)
);
```

-- Insert sample policies 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');

SAMPLE TABLES

	clientId	clientName	contactInfo	policy
1	1	Eswara	6301497898	Health Insurance

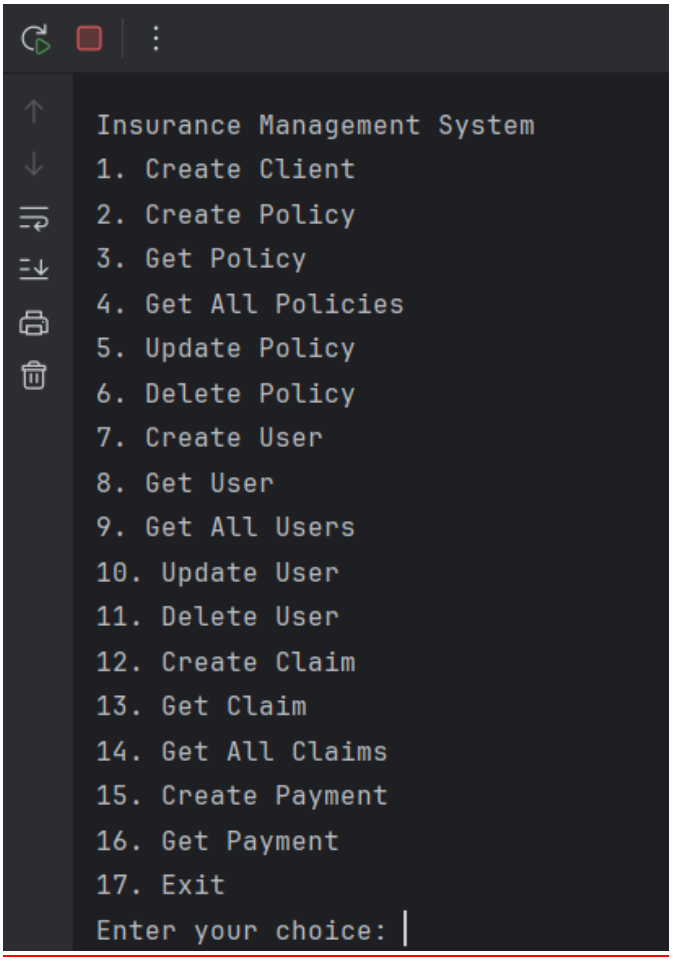
userId	username	password	role

claimId	claimNumber	dateFiled	claimAmount	status	clientId	policy

paymentId	paymentDate	paymentAmount	clientId

	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	term insurance	this is description of term insurance

Start Output:



CREATE USER:

```
Enter your choice: 7
Enter username: GaniUser101
Enter password: Hexa101
Enter role (admin/user): admin
User created successfully!
```

CREATE POLICY

```
Enter your choice: 2
Enter policy name: Indian Home Policy
Enter policy description: Get Insurance if your house is affected in disaster!
Policy created successfully!
```

CREATE CLAIM

```
Enter your choice: 12
Enter claim number: 101
Enter date filed (YYYY-MM-DD): 2024-10-09
Enter claim amount: 2000
Enter status: Active
Enter associated policy: LIC
Enter associated client: 1
Claim created successfully!
```

CREATE PAYMENT

```
Enter your choice: 15
Enter payment date (YYYY-MM-DD): 2024-10-09
Enter payment amount: 2000
Enter associated client: 3
Payment created successfully!
```

CREATE CLIENT

```
Enter your choice: 1
Enter client name: Gani
Enter contact info: 9999999999
Enter policy: NPS policy
Client created successfully!
```

GET ALL POLICIES

```
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=term insurance, policyDescription=this is description of term insurance]
Policy [policyId=7, policyName=Indian Home Policy, policyDescription=Get Insurance if your house is affected in disaster!]
```

GET ALL USERS

```
Enter your choice: 9
User [userId=1, username=eswara101, role=user]
User [userId=2, username=GaniUser101, role=admin]
```

GET ALL CLAIMS

```
Enter your choice: 14
Claim [claimId=1, claimNumber=101, dateFiled=2024-10-09, claimAmount=2000.00, status=Active, policy=LIC, client=1]
```

INSERTED DATA

Results | Messages

	clientId	clientName	contactInfo	policy
1	1	Eswara	6301497898	Health Insurance
2	2	venkata	8309525344	LIC
3	3	Gani	9999999999	NPS policy

	userId	username	password	role
1	1	eswara101	eswara101	user
2	2	GaniUser101	Hexa101	admin

	claimId	claimNumber	dateFiled	claimAmount	status	clientId	policy
1	1	101	2024-10-09	2000.00	Active	1	LIC

	paymentId	paymentDate	paymentAmount	clientId
1	1	2024-10-09	2000.00	3

	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	term insurance	this is description of term insurance
6	7	Indian Home Policy	Get Insurance if your house is affected in disaster!

GET POLICY

```
Enter your choice: 3
Enter policy ID: 7
Policy [policyId=7, policyName=Indian Home Policy, policyDescription=Get Insurance if your house is affecte
```

UPDATE POLICY

```
Enter your choice: 5
Enter policy ID: 3
Enter new policy name: Car Insurance
Enter new policy description: get car insurance of disaster chelamities!
Policy updated successfully!
```

DELETE POLICY

```
Enter your choice: 6
Enter policy ID: 5
Policy deleted successfully!
```

GET USERS

```
Enter your choice: 8
Enter user ID: 2
User [userId=2, username=GaniUser101, role=admin]
```

UPDATE USER

```
Enter your choice: 10
Enter user ID: 2
Enter new username: Gani202
Enter new password: Gani202PWD
Enter new role (admin/user): user
User updated successfully!
```

DELETE USER

```
Enter your choice: 11
Enter user ID: 1
User deleted successfully!
```

GET CLAIM

```
Enter your choice: 13
Enter claim ID: 1
Claim [claimId=1, claimNumber=101, dateFiled=2024-10-09, claimAmount=2000.00, status=Active, policy=LIC, client=1]
```

GET PAYMENT

```
Enter your choice: 16
Enter payment ID: 1
Payment [paymentId=1, paymentDate=2024-10-09, paymentAmount=2000.00, client=3]
```

EXITING

```
Enter your choice: 17
Exiting... Insurance Management System
```