Coding Challenge-Insurance

Name:Soundarya V

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

Define `User ` class with the following confidential attributes:

a. userId; b. username; c. password; d. role;

```
mysql> desc user;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| userId   | int          | NO   | PRI | NULL    |       |
| username | varchar(255) | YES  |     | NULL    |       |
| password | varchar(255) | YES  |     | NULL    |       |
| role     | varchar(50)  | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

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

a. clientId; b. clientName; c. contactInfo; d. policy;//Represents the policy associated with the client

```
mysql> desc client;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| clientId    | int          | NO   | PRI | NULL    |       |
| clientName  | varchar(255) | YES  |     | NULL    |       |
| contactInfo | varchar(255) | YES  |     | NULL    |       |
| policy      | varchar(100) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

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

```
mysql> desc claim;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| claimId     | int           | NO   | PRI | NULL    |       |
| claimNumber | varchar(100)  | YES  |     | NULL    |       |
| dateFiled   | date          | YES  |     | NULL    |       |
| claimAmount | decimal(10,2) | YES  |     | NULL    |       |
| status      | varchar(50)   | YES  |     | NULL    |       |
| policy      | varchar(100)  | YES  |     | NULL    |       |
| clientId    | int           | YES  | MUL | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

. Define ` Payment ` class with the following confidential attributes:

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

```
mysql> desc payment;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| paymentId     | int           | NO   | PRI | NULL    |       |
| paymentDate   | date          | YES  |     | NULL    |       |
| paymentAmount | decimal(10,2) | YES  |     | NULL    |       |
| clientId      | int           | YES  | MUL | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

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.

```
class User:
    def __init__(self, userId, username, password, role):
        self.userId = userId
        self.username = username
        self.password = password
        self.role = role

    def getUserId(self):
        return self.userId

    def setUserId(self, userId):
        self.userId = userId

    def getUsername(self):
        return self.username

    def setUsername(self, username):
        self.username = username

    def getPassword(self):
        return self.password

    def setPassword(self, password):
        self.password = password

    def getRole(self):
        return self.role

    def setRole(self, role):
        self.role = role

    def printDetails(self):
        print("User Details:")
        print("User ID:", self.userId)
        print("Username:", self.username)
        print("Password:", self.password)
        print("Role:", self.role)
```

Implementation:

```
from Insurance import User
user1 = User(1, "Rahul", "RL23", "admin")
user1.printDetails()
```

```
User Details:
User ID: 1
Username: Rahul
Password: RL23
Role: admin
```

Client:

```python
class Client:
    def __init__(self, clientId, clientName, contactInfo, policy):
        self.clientId = clientId
        self.clientName = clientName
        self.contactInfo = contactInfo
        self.policy = policy

    def getClientId(self):
        return self.clientId

    def setClientId(self, clientId):
        self.clientId = clientId

    def getClientName(self):
        return self.clientName

    def setClientName(self, clientName):
        self.clientName = clientName

    def getContactInfo(self):
        return self.contactInfo

    def setContactInfo(self, contactInfo):
        self.contactInfo = contactInfo

    def getPolicy(self):
        return self.policy

    def setPolicy(self, policy):
        self.policy = policy

    def printDetails(self):
        print("Client Details:")
        print("Client ID:", self.clientId)
        print("Client Name:", self.clientName)
        print("Contact Info:", self.contactInfo)
        print("Policy:", self.policy)
```

Implemntation:

```
from Insurance import Client
client1 = Client(1, "Som", "9876477289", "Health")
client1.printDetails()
```

```
Client Details:
Client ID: 1
Client Name: Som
Contact Info: 9876477289
Policy: Health
```

Claim:

```
class Claim:
    def __init__(self, claimId, claimNumber, dateFiled, claimAmount, status, policy, clientId):
        self.claimId = claimId
        self.claimNumber = claimNumber
        self.dateFiled = dateFiled
        self.claimAmount = claimAmount
        self.status = status
        self.policy = policy
        self.clientId = clientId

    def getClaimId(self):
        return self.claimId

    def setClaimId(self, claimId):
        self.claimId = claimId

    def getClaimNumber(self):
        return self.claimNumber

    def setClaimNumber(self, claimNumber):
        self.claimNumber = claimNumber

    def getDateFiled(self):
        return self.dateFiled

    def setDateFiled(self, dateFiled):
        self.dateFiled = dateFiled

    def getClaimAmount(self):
        return self.claimAmount

    def setClaimAmount(self, claimAmount):
        self.claimAmount = claimAmount

    def getStatus(self):
        return self.status

    def setStatus(self, status):
```

```
        self.status = status

    def getPolicy(self):
        return self.policy

    def setPolicy(self, policy):
        self.policy = policy

    def getClientId(self):
        return self.clientId

    def setClientId(self, clientId):
        self.clientId = clientId

    def printDetails(self):
        print("Claim Details:")
        print("Claim ID:", self.claimId)
        print("Claim Number:", self.claimNumber)
        print("Date Filed:", self.dateFiled)
        print("Claim Amount:", self.claimAmount)
        print("Status:", self.status)
        print("Policy:", self.policy)
        print("Client ID:", self.client.getClientId())
```

Implementation:

```
from Insurance import Claim
claim1=Claim(1, 33, "2024-04-09", 50000, "claimed","Health", 1)
claim1.printDetails()
```

```
Claim Details:
Claim ID: 1
Claim Number: 33
Date Filed: 2024-04-09
Claim Amount: 50000
Status: claimed
Policy: Health
Client ID: 1
```

Payment:

```
class Payment:
    def __init__(self, paymentId, paymentDate, paymentAmount, clientId):
        self.paymentId = paymentId
        self.paymentDate = paymentDate
        self.paymentAmount = paymentAmount
        self.clientId = clientId

    def getPaymentId(self):
        return self.paymentId
```

```
    def setPaymentId(self, paymentId):
        self.paymentId = paymentId

    def getPaymentDate(self):
        return self.paymentDate

    def setPaymentDate(self, paymentDate):
        self.paymentDate = paymentDate

    def getPaymentAmount(self):
        return self.paymentAmount

    def setPaymentAmount(self, paymentAmount):
        self.paymentAmount = paymentAmount

    def getClientId(self):
        return self.clientId

    def setClientId(self, clientId):
        self.clientId = clientId

    def printDetails(self):
        print("Payment ID:", self.paymentId)
        print("Payment Date:", self.paymentDate)
        print("Payment Amount:", self.paymentAmount)
        print("Client ID:", self.client.getClientId())
```

Implementation:

```
from Insurance import Payment
Payment1=Payment(1, "2024-04-09", 50000, 1)
Payment1.printDetails()
```

```
Payment ID: 1
Payment Date: 2024-04-09
Payment Amount: 50000
Client ID: 1
```

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

```
from abc import ABC, abstractmethod
from Insurance.Insurance import Policy
from typing import List


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

```
import mysql.connector
from mysql.connector import Error
from typing import List
from Insurance.dao.insurance_service import IInsuranceService
from Insurance.Insurance import User, Client, Claim, Payment


class InsuranceServiceImpl(IInsuranceService):
   def __init__(self):
      self.connection = mysql.connector.connect(
         host='localhost',
         user='root',
         password='root',
         database='insurance'
      )

   def createUser(self, user: User) -> bool:
      try:
         cursor = self.connection.cursor()
         cursor.execute("INSERT INTO user (userId, username, password, role) VALUES (%s, %s, %s,
%s)",
```

```
                    (user.userId, user.username, user.password, user.role))
            self.connection.commit()
            return True
        except Error as e:
            print("Error creating user:", e)
            return False
        finally:
            cursor.close()

    def getUser(self, userId: int) -> User:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM user WHERE userId = %s", (userId,))
            row = cursor.fetchone()
            if row:
                user = User(*row)
                print("Retrieved User:")
                print("User ID:", user.userId)
                print("Username:", user.username)
                print("Password:", user.password)
                print("Role:", user.role)
                return user

            else:
                return None
        except Error as e:
            print("Error retrieving user:", e)
            return None
        finally:
            cursor.close()

    def getAllUsers(self) -> List[User]:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM user")
            rows = cursor.fetchall()
            users = []
            for row in rows:
                user = User(*row)
                print("User ID:", user.userId)
                print("Username:", user.username)
                print("Password:", user.password)
                print("Role:", user.role)
                users.append(user)
            return users
        except Error as e:
            print("Error retrieving users:", e)
            return []
        finally:
            cursor.close()
```

```
    def deleteUser(self, userId: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("DELETE FROM user WHERE userId = %s", (userId,))
            self.connection.commit()
            return True
        except Error as e:
            print("Error deleting user:", e)
            return False
        finally:
            cursor.close()

    def updateUser(self, user: User) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "UPDATE user SET"
            values = []

            if user.username is not None:
                query += " username = %s,"
                values.append(user.username)
            if user.password is not None:
                query += " password = %s,"
                values.append(user.password)
            if user.role is not None:
                query += " role = %s,"
                values.append(user.role)

            query = query.rstrip(',')

            query += " WHERE userId = %s"
            values.append(user.userId)

            cursor.execute(query, tuple(values))
            self.connection.commit()
            return True
        except Error as e:
            print("Error updating user:", e)
            return False
        finally:
            cursor.close()

    def createClient(self, client: Client) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("INSERT INTO client (clientId, clientName, contactInfo, policy) VALUES (%s,
%s, %s, %s)",
                           (client.clientId, client.clientName, client.contactInfo, client.policy))
            self.connection.commit()
            return True
        except Error as e:
```

```
            print("Error creating client:", e)
            return False
        finally:
            cursor.close()

    def getClient(self, clientId: int) -> Client:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM client WHERE clientId = %s", (clientId,))
            row = cursor.fetchone()
            if row:
                client = Client(*row)
                print("User:")
                print("Client ID:", client.clientId)
                print("Clientname:", client.clientName)
                print("Contactinfo:", client.contactInfo)
                print("Policy:",client.policy)
                return client
            else:
                return None
        except Error as e:
            print("Error retrieving client:", e)
            return None
        finally:
            cursor.close()

    def getAllClients(self) -> List[Client]:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM client")
            rows = cursor.fetchall()
            clients = []
            for row in rows:
                client = Client(*row)
                print("Client ID:", client.clientId)
                print("Clientname:", client.clientName)
                print("Contactinfo:", client.contactInfo)
                print("Policy:", client.policy)
                clients.append(client)
            return clients
        except Error as e:
            print("Error retrieving clients:", e)
            return []
        finally:
            cursor.close()

    def deleteClient(self, clientId: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("DELETE FROM client WHERE clientId = %s", (clientId,))
            self.connection.commit()
```

```
            return True
        except Error as e:
            print("Error deleting client:", e)
            return False
        finally:
            cursor.close()

    def updateClient(self, client: Client) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "UPDATE clients SET"
            values = []

            if client.clientName is not None:
                query += " clientName = %s,"
                values.append(client.clientName)
            if client.contactInfo is not None:
                query += " contactInfo = %s,"
                values.append(client.contactInfo)
            if client.policyId is not None:
                query += " policyId = %s,"
                values.append(client.policyId)

            query = query.rstrip(',')

            query += " WHERE clientId = %s"
            values.append(client.clientId)

            cursor.execute(query, tuple(values))
            self.connection.commit()
            return True
        except Error as e:
            print("Error updating client:", e)
            return False
        finally:
            cursor.close()

    def createClaim(self, claim: Claim) -> bool:
        try:
            cursor = self.connection.cursor()

            cursor.execute("INSERT INTO claim (claimId, claimNumber, dateFiled, claimAmount, status,
policyId, clientId) VALUES (%s, %s, %s, %s, %s, %s, %s)",
                           (claim.claimId, claim.claimNumber, claim.dateFiled, claim.claimAmount, claim.status,
claim.policyId, claim.clientId))
            self.connection.commit()
            return True
        except Error as e:
            print("Error creating claim:", e)
            return False
        finally:
```

```
        cursor.close()

    def getClaim(self, claimId: int) -> Claim:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM claim WHERE claimId = %s", (claimId,))
            row = cursor.fetchone()
            if row:
                claim = Claim(*row)
                print("Claim ID:", claim.claimId)
                print("Claim Number:", claim.claimNumber)
                print("Date Filed:", claim.dateFiled)
                print("Claim Amount:", claim.claimAmount)
                print("Status:", claim.status)
                print("Policy ID:", claim.policy)
                print("Client ID:", claim.clientId)
                return claim
            else:
                return None
        except Error as e:
            print("Error retrieving claim:", e)
            return None
        finally:
            cursor.close()

    def getAllClaim(self) -> List[Claim]:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM claim")
            rows = cursor.fetchall()
            claims = []
            for row in rows:
                claim = Claim(*row)
                print("Claim ID:", claim.claimId)
                print("Claim Number:", claim.claimNumber)
                print("Date Filed:", claim.dateFiled)
                print("Claim Amount:", claim.claimAmount)
                print("Status:", claim.status)
                print("Policy ID:", claim.policy)
                print("Client ID:", claim.clientId)
                claims.append(claim)
            return claims
        except Error as e:
            print("Error retrieving claims:", e)
            return []
        finally:
            cursor.close()

    def deleteClaim(self, claimId: int) -> bool:
        try:
            cursor = self.connection.cursor()
```

```
            cursor.execute("DELETE FROM claim WHERE claimId = %s", (claimId,))
            self.connection.commit()
            return True
        except Error as e:
            print("Error deleting claim:", e)
            return False
        finally:
            cursor.close()

    def updateClaim(self, claim: Claim) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "UPDATE claims SET"
            values = []

            if claim.claimNumber is not None:
                query += " claimNumber = %s,"
                values.append(claim.claimNumber)
            if claim.dateFiled is not None:
                query += " dateFiled = %s,"
                values.append(claim.dateFiled)
            if claim.claimAmount is not None:
                query += " claimAmount = %s,"
                values.append(claim.claimAmount)
            if claim.status is not None:
                query += " status = %s,"
                values.append(claim.status)
            if claim.policyId is not None:
                query += " policyId = %s,"
                values.append(claim.policyId)
            if claim.clientId is not None:
                query += " clientId = %s,"
                values.append(claim.clientId)

            query = query.rstrip(',')

            query += " WHERE claimId = %s"
            values.append(claim.claimId)

            cursor.execute(query, tuple(values))
            self.connection.commit()
            return True
        except Error as e:
            print("Error updating claim:", e)
            return False
        finally:
            cursor.close()

    def createPayment(self, payment: Payment) -> bool:
        try:
            cursor = self.connection.cursor()
```

```
            cursor.execute("INSERT INTO payment (paymentId, paymentDate, paymentAmount, clientId)
VALUES (%s, %s, %s, %s)",
                           (payment.paymentId, payment.paymentDate, payment.paymentAmount,
payment.clientId))
            self.connection.commit()
            return True
        except Error as e:
            print("Error creating payment:", e)
            return False
        finally:
            cursor.close()

    def getPayment(self, paymentId: int) -> Payment:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM payment WHERE paymentId = %s", (paymentId,))
            row = cursor.fetchone()
            if row:
                payment = Payment(*row)
                return payment
            else:
                return None
        except Error as e:
            print("Error retrieving payment:", e)
            return None
        finally:
            cursor.close()

    def getAllPayments(self) -> List[Payment]:
        try:
            cursor = self.connection.cursor()
            cursor.execute("SELECT * FROM payment")
            rows = cursor.fetchall()
            payments = []
            for row in rows:
                payment = Payment(*row)
                payments.append(payment)
            return payments
        except Error as e:
            print("Error retrieving payments:", e)
            return []
        finally:
            cursor.close()

    def deletePayment(self, paymentId: int) -> bool:
        try:
            cursor = self.connection.cursor()
            cursor.execute("DELETE FROM payment WHERE paymentId = %s", (paymentId,))
            self.connection.commit()
            return True
        except Error as e:
```

```
            print("Error deleting payment:", e)
            return False
        finally:
            cursor.close()

    def updatePayment(self, payment: Payment) -> bool:
        try:
            cursor = self.connection.cursor()
            query = "UPDATE payment SET"
            values = []

            if payment.paymentDate is not None:
                query += " paymentDate = %s,"
                values.append(payment.paymentDate)
            if payment.paymentAmount is not None:
                query += " paymentAmount = %s,"
                values.append(payment.paymentAmount)
            if payment.clientId is not None:
                query += " clientId = %s,"
                values.append(payment.clientId)

            query = query.rstrip(',')
            query += " WHERE paymentId = %s"
            values.append(payment.paymentId)

            cursor.execute(query, tuple(values))
            self.connection.commit()
            return True
        except Error as e:
            print("Error updating payment:", e)
            return False
        finally:
            cursor.close()
```

Implementation:
Example for Claim:

```
from Insurance import User, Client, Claim, Payment
from dao.insurance_impl import InsuranceServiceImpl
insurance_service=InsuranceServiceImpl()
claim1 = Claim(claimId=1, claimNumber="CLM001", dateFiled="2024-05-01",
claimAmount=5000.00, status="Pending", policy="Life", clientId=1)
claim2 = Claim(claimId=2, claimNumber="CLM002", dateFiled="2024-04-11",
claimAmount=50000.00, status="Claimed", policy="Health", clientId=2)
insurance_service.createClaim(claim1)
insurance_service.createClaim(claim2)
updated_claim = Claim(claimId=2, claimNumber="CLM002", dateFiled="2024-03-
11", claimAmount=7500.00, status="Claimed", policy="Health", clientId=1)
insurance_service.updateClaim(updated_claim)
claim6= Claim(claimId=6, claimNumber="CLM006", dateFiled="2024-04-11",
claimAmount=250000.00, status="Claimed", policy="Fire", clientId=1)
```

```
insurance_service.createClaim(claim6)
retrieved_claim = insurance_service.getClaim(6)

retrieved_claim = insurance_service.getClaim(6)
claim.printdetails(retrived_claim)
```

```
Claim ID: 6
Claim Number: CLM006
Date Filed: 2024-04-11
Claim Amount: 250000.00
Status: Claimed
```

4.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.

```
import mysql.connector
from configparser import ConfigParser


class DBConnection:
    connection = None

    @staticmethod
    def getConnection():
        if DBConnection.connection is None:
            # Read connection properties from property file
            connection_string = PropertyUtil.getPropertyString()

            try:
                # Establish database connection
                DBConnection.connection = mysql.connector.connect(**connection_string)
            except mysql.connector.Error as e:
                print("Error connecting to MySQL:", e)

        return DBConnection.connection


class PropertyUtil:
    @staticmethod
    def getPropertyString():
        config = ConfigParser()
        config.read('connection.properties')  # Assuming the property file is named connection.properties
```

```
    # Read connection details from the property file
    hostname = config.get('Database', 'localhost')
    dbname = config.get('Database', 'insurance')
    username = config.get('Database', 'root')
    password = config.get('Database', 'root')
    port = config.get('Database', 3306)

    # Construct connection dictionary
    connection_string = {
        'host': hostname,
        'user': username,
        'password': password,
        'database': dbname,
        'port': int(port),
        'charset': 'utf8mb4',
        'cursorclass': mysql.connector.cursor.MySQLCursorDict
    }

    return connection_string
```

5.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

```
class PolicyNotFoundException(Exception):

    def __init__(self, policy_id):
        super().__init__(f"Policy with ID {policy_id} not found in the
database.")
```

Implementation:

```
def retrieve_policy(policy_id):
    try:
        policy = insurance_service.getPolicy(policy_id)
        if policy:
            print("Retrieved Policy:")
            print("Policy ID:", policy.policyId)
            print("Policy Name:", policy.policyName)
            print("Policy Description:", policy.policyDescription)
        else:
            raise PolicyNotFoundException(policy_id)
```

```
Policy with ID 123 not found in the database.
```

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

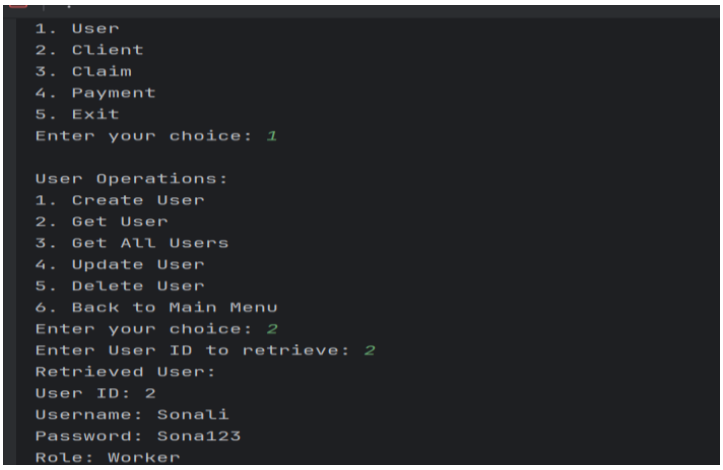
```
from Insurance.Insurance import User, Client, Claim, Payment
from Insurance.dao.insurance_impl import InsuranceServiceImpl
insurance_service=InsuranceServiceImpl()

def user(insurance_service):
    while True:
        print("\nUser Operations:")
        print("1. Create User")
        print("2. Get User")
        print("3. Get All Users")
        print("4. Update User")
        print("5. Delete User")
        print("6. Back to Main Menu")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            user = User(userId=int(input("Enter User ID: ")),
                        username=input("Enter Username: "),
                        password=input("Enter Password: "),
                        role=input("Enter Role: "))
            result = insurance_service.createUser(user)
            print("User created:", result)
        elif choice == 2:
            user_id = int(input("Enter User ID to retrieve: "))
            user = insurance_service.getUser(user_id)
            print("Retrieved User:", user)
        elif choice == 3:
            users = insurance_service.getAllUsers()
            print("All Users:", users)
        elif choice == 4:
            user = User(userId=int(input("Enter User ID to update: ")),
                        username=input("Enter New Username: "),
                        password=input("Enter New Password: "),
                        role=input("Enter New Role: "))
            result = insurance_service.updateUser(user)
            print("User updated:", result)
        elif choice == 5:
            user_id = int(input("Enter User ID to delete: "))
            result = insurance_service.deleteUser(user_id)
            print("User deleted:", result)
        elif choice == 6:
            break
        else:
            print("Invalid choice!")

def client(insurance_service):
    while True:
        print("\nClient Operations:")
        print("1. Create Client")
```

```
        print("2. Get Client")
        print("3. Get All Clients")
        print("4. Update Client")
        print("5. Delete Client")
        print("6. Back to Main Menu")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            client = Client(clientId=int(input("Enter Client ID: ")),
                            clientName=input("Enter Client Name: "),
                            contactInfo=input("Enter Contact Info: "),
                            policy=int(input("Enter Policy ID: ")))
            result = insurance_service.createClient(client)
            print("Client created:", result)
        elif choice == 2:
            client_id = int(input("Enter Client ID to retrieve: "))
            client = insurance_service.getClient(client_id)
            print("Retrieved Client:", client)
        elif choice == 3:
            clients = insurance_service.getAllClients()
            print("All Clients:", clients)
        elif choice == 4:
            client = Client(clientId=int(input("Enter Client ID to update:
")),
                            clientName=input("Enter New Client Name: "),
                            contactInfo=input("Enter New Contact Info: "),
                            policy=int(input("Enter New Policy ID: ")))
            result = insurance_service.updateClient(client)
            print("Client updated:", result)
        elif choice == 5:
            client_id = int(input("Enter Client ID to delete: "))
            result = insurance_service.deleteClient(client_id)
            print("Client deleted:", result)
        elif choice == 6:
            break
        else:
            print("Invalid choice!")

def claim(insurance_service):
    while True:
        print("\nClaim Operations:")
        print("1. Create Claim")
        print("2. Get Claim")
        print("3. Get All Claims")
        print("4. Update Claim")
        print("5. Delete Claim")
        print("6. Back to Main Menu")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            claim = Claim(claimId=int(input("Enter Claim ID: ")),
                          claimNumber=input("Enter Claim Number: "),
                          dateFiled=input("Enter Date Filed: "),
                          claimAmount=float(input("Enter Claim Amount: ")),
                          status=input("Enter Status: "),
```

```
                        policy=int(input("Enter Policy ID: ")),
                        clientId=int(input("Enter Client ID: ")))
            result = insurance_service.createClaim(claim)
            print("Claim created:", result)
        elif choice == 2:
            claim_id = int(input("Enter Claim ID to retrieve: "))
            claim = insurance_service.getClaim(claim_id)
            print("Retrieved Claim:", claim)
        elif choice == 3:
            claims = insurance_service.getAllClaims()
            print("All Claims:", claims)
        elif choice == 4:
            claim = Claim(claimId=int(input("Enter Claim ID to update: ")),
                        claimNumber=input("Enter New Claim Number: "),
                        dateFiled=input("Enter New Date Filed: "),
                        claimAmount=float(input("Enter New Claim Amount:
")),
                        status=input("Enter New Status: "),
                        policy=int(input("Enter New Policy ID: ")),
                        clientId=int(input("Enter New Client ID: ")))
            result = insurance_service.updateClaim(claim)
            print("Claim updated:", result)
            # Delete Claim
            claim_id = int(input("Enter Claim ID to delete: "))
            result = insurance_service.deleteClaim(claim_id)
            print("Claim deleted:", result)
        elif choice == 6:
            break
        else:
            print("Invalid choice!")

def payment(insurance_service):
    while True:
        print("\nPayment Operations:")
        print("1. Create Payment")
        print("2. Get Payment")
        print("3. Get All Payments")
        print("4. Update Payment")
        print("5. Delete Payment")
        print("6. Back to Main Menu")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            payment = Payment(paymentId=int(input("Enter Payment ID: ")),
                              paymentDate=input("Enter Payment Date: "),
                              paymentAmount=float(input("Enter Payment
Amount: ")),
                              clientId=int(input("Enter Client ID: ")))
            result = insurance_service.createPayment(payment)
            print("Payment created:", result)
        elif choice == 2:
            payment_id = int(input("Enter Payment ID to retrieve: "))
            payment = insurance_service.getPayment(payment_id)
            print("Retrieved Payment:", payment)
        elif choice == 3:
            payments = insurance_service.getAllPayments()
```

```
            print("All Payments:", payments)
        elif choice == 4:
            payment = Payment(paymentId=int(input("Enter Payment ID to
update: ")),
                              paymentDate=input("Enter New Payment Date: "),
                              paymentAmount=float(input("Enter New Payment
Amount: ")),
                              clientId=int(input("Enter New Client ID: ")))
            result = insurance_service.updatePayment(payment)
            print("Payment updated:", result)
        elif choice == 5:
            # Delete Payment
            payment_id = int(input("Enter Payment ID to delete: "))
            result = insurance_service.deletePayment(payment_id)
            print("Payment deleted:", result)
        elif choice == 6:
            # Back to Main Menu
            break
        else:
            print("Invalid choice")


def main():
    insurance_service = InsuranceServiceImpl()

    while True:
        print("\nMain Menu:")
        print("1. User")
        print("2. Client")
        print("3. Claim")
        print("4. Payment")
        print("5. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            user(insurance_service)
        elif choice == 2:
            client(insurance_service)
        elif choice==3:
            claim(insurance_service)
        elif choice==4:
            payment(insurance_service)
        elif choice == 5:
            print("Exiting...")
            break
        else:
            print("Invalid choice")



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

Implementation:


1. User
2. Client
3. Claim
4. Payment
5. Exit
Enter your choice: 1
User Operations:
1. Create User
2. Get User
3. Get All Users
4. Update User
5. Delete User
6. Back to Main Menu
Enter your choice: 2
Enter User ID to retrieve: 2
Retrieved User:
User ID: 2
Username: Sonali
Password: Sona123
Role: Worker