

## Python Coding Challenge

Swathi Baskaran

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

**Query:**

**Creating tables:**

```
CREATE DATABASE Insurance;
```

```
USE Insurance;
```

```
-- Creating table for User
```

```
CREATE TABLE User (  
    userID INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50) NOT NULL,  
    password VARCHAR(50) NOT NULL,  
    role VARCHAR(20) NOT NULL  
);
```

```
-- Creating table for Policy
```

```
CREATE TABLE Policy (  
    PolicyID INT AUTO_INCREMENT PRIMARY KEY,  
    PolicyName VARCHAR(100) NOT NULL,  
    CoverageDetails TEXT  
);
```

```
-- Creating table for Client
```

```
CREATE TABLE Client (  
    ClientID INT AUTO_INCREMENT PRIMARY KEY,
```

```
ClientName VARCHAR(100) NOT NULL,  
ContactInfo VARCHAR(100),  
PolicyID INT,  
FOREIGN KEY (PolicyID) REFERENCES Policy(PolicyID)  
);
```

-- Creating table for Claim

```
CREATE TABLE Claim (  
    ClaimID INT AUTO_INCREMENT PRIMARY KEY,  
    claim_number VARCHAR(50) NOT NULL,  
    date_filed DATE NOT NULL,  
    claim_amount DECIMAL(10,2) NOT NULL,  
    status VARCHAR(20) NOT NULL,  
    PolicyID INT,  
    ClientID INT,  
    FOREIGN KEY (PolicyID) REFERENCES Policy(PolicyID),  
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID)  
);
```

-- Creating table for Payment

```
CREATE TABLE Payment (  
    PaymentID INT AUTO_INCREMENT PRIMARY KEY,  
    PaymentDate DATE NOT NULL,  
    PaymentAmount DECIMAL(10,2) NOT NULL,  
    ClientID INT,  
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID)  
);
```

## Inserting Data:

-- Inserting data for User table

```
INSERT INTO User (username, password, role) VALUES  
( 'admin1', 'admin123', 'admin'),  
( 'agent1', 'agent123', 'agent'),  
( 'agent2', 'agent456', 'agent'),  
( 'client1', 'client123', 'client'),  
( 'client2', 'client456', 'client');
```

-- Inserting data for Policy table

```
INSERT INTO Policy (PolicyName, CoverageDetails) VALUES  
( 'Basic Health', 'Covers hospitalization up to $50,000'),  
( 'Premium Health', 'Full coverage including dental and vision'),  
( 'Auto Basic', 'Covers collision damage up to $25,000'),  
( 'Auto Premium', 'Full coverage including roadside assistance'),  
( 'Homeowners', 'Property damage and liability coverage');
```

-- Inserting data for Client table

```
INSERT INTO Client (ClientName, ContactInfo, PolicyID) VALUES  
( 'John Smith', 'john.smith@email.com', 2),  
( 'Sarah Johnson', 'sarah.j@email.com', 1),  
( 'Michael Brown', 'michael.b@email.com', 3),  
( 'Emily Davis', 'emily.d@email.com', 5),  
( 'David Wilson', 'david.w@email.com', 4);
```

-- Inserting data for Claim table

```
INSERT INTO Claim (claim_number, date_filed, claim_amount, status,  
PolicyID, ClientID) VALUES
```

```

('CL2023001', '2023-01-15', 2500.00, 'Approved', 2, 1),
('CL2023002', '2023-02-20', 12000.00, 'Pending', 1, 2),
('CL2023003', '2023-03-10', 8000.00, 'Denied', 3, 3),
('CL2023004', '2023-04-05', 3500.00, 'Approved', 5, 4),
('CL2023005', '2023-05-12', 15000.00, 'Pending', 4, 5);

```

-- Inserting data for Payment table

```

INSERT INTO Payment (PaymentDate, PaymentAmount, ClientID) VALUES
('2023-01-10', 120.00, 1),
('2023-02-15', 85.50, 2),
('2023-03-20', 210.75, 3),
('2023-04-25', 175.00, 4),
('2023-05-30', 300.00, 5);

```

### User Table:


Result Grid				
Filter Rows:				
	userID	username	password	role
▶	1	admin1	admin123	admin
	2	agent1	agent123	agent
	3	agent2	agent456	agent
	4	client1	client123	client
	5	client2	client456	client
✱	NULL	NULL	NULL	NULL

### Policy Table:


Result Grid			
Filter Rows:			
	PolicyID	PolicyName	CoverageDetails
▶	1	Basic Health	Covers hospitalization up to \$50,000
	2	Health Policy	Full coverage of health details
	3	Auto Basic	Covers collision damage up to \$25,000
	4	Auto Premium	Full coverage including roadside assistance
	5	Homeowners	Property damage and liability coverage
	8	Health and Medical Expenses	NULL
✱	NULL	NULL	NULL

## Client Table:

Result Grid



Filter Rows:

Edit: 

	ClientID	ClientName	ContactInfo	PolicyID
▶	1	John Smith	john.smith@email.com	2
	2	Sarah Johnson	sarah.j@email.com	1
	3	Michael Brown	michael.b@email.com	3
	4	Emily Davis	emily.d@email.com	5
	5	David Wilson	david.w@email.com	4
•	NULL	NULL	NULL	NULL

## Claim Table:

Result Grid

Filter Rows:

Edit:

Export/Import:

	ClaimID	claim_number	date_filed	claim_amount	status	PolicyID	ClientID
▶	1	CL2023001	2023-01-15	2500.00	Approved	2	1
	2	CL2023002	2023-02-20	12000.00	Pending	1	2
	3	CL2023003	2023-03-10	8000.00	Denied	3	3
	4	CL2023004	2023-04-05	3500.00	Approved	5	4
	5	CL2023005	2023-05-12	15000.00	Pending	4	5
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL

## Payment Table:

Result Grid

Filter Rows:

Edit:

	PaymentID	PaymentDate	PaymentAmount	ClientID
▶	1	2023-01-10	120.00	1
	2	2023-02-15	85.50	2
	3	2023-03-20	210.75	3
	4	2023-04-25	175.00	4
	5	2023-05-30	300.00	5
✱	NULL	NULL	NULL	NULL

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

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.

Define `User` class with the following confidential attributes:

a. userId;

**b. username;**  
**c. password;**  
**d. role;**

**Query:**

**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
```

```
@property
```

```
def get__userID(self):
```

```
    return self.__userID
```

```
@property
```

```
def get__username(self):
```

```
    return self.__username
```

```
@property
```

```
def get__password(self):
```

```
    return self.__password
```

```
@property
```

```
def get__role(self):
```

```
    return self.__role
```

```

def set_userID(self, value):
    self.__userID = value

def set_name(self, value):
    self.__username = value

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

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

def __str__(self):
    return f"User ID: {self.__userID} \nUsername: {self.__username} \nRole: {self.__role}"

```

**Define `Client` class with the following confidential attributes:**

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

**Query:**

```

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
```

```
@property
def get_clientID(self):
    return self.__clientID
```

```
@property
def get_clientName(self):
    return self.__clientName
```

```
@property
def get_contactInfo(self):
    return self.__contactInfo
```

```
@property
def get_policy(self):
    return self.__policy
```

```
def set_clientID(self, value):
    self.__clientID = value
```

```
def set_clientName(self, value):
    self.__clientName = value
```

```
def set_contactInfo(self, value):
```



```

        self.__contactInfo = value

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

    def __str__(self):
        return f"Client ID: {self.__clientID} \nClient Name: {self.__clientName} \nContact Info: {self.__contactInfo} \nPolicy: {self.__policy}"

```

**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**

**Query:**

```

class Claim:
    def __init__(self, claimID=None, claimNumber=None, dateFiled=None, claimAmount=None, status=None, policy=None, client=None):
        self.__claimID = claimID
        self.__claimNumber = claimNumber
        self.__dateFiled = dateFiled
        self.__claimAmount = claimAmount
        self.__status = status

```

```
self.__policy = policy
```

```
self.__client = client
```

```
def get_claimID(self):
```

```
    return self.__claimID
```

```
def get_claimNumber(self):
```

```
    return self.__claimNumber
```

```
def get_dateFilled(self):
```

```
    return self.__dateFilled
```

```
def get_claimAmount(self):
```

```
    return self.__claimAmount
```

```
def get_status(self):
```

```
    return self.__status
```

```
def get_policy(self):
```

```
    return self.__policy
```

```
def get_client(self):
```

```
    return self.__client
```

```
def set_claimID(self, claimID):
```

```
    self.__claimID = claimID
```

```

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

def set_dateFilled(self, dateFilled):
    self.__dateFilled = dateFilled

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

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

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

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

def __str__(self):
    return f"Claim ID: {self.__claimID}, Claim Number: {self.__claimNumber}, Claim Amount: {self.__claimAmount}, Status: {self.__status})"

```

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

- a. paymentId;**
- b. paymentDate;**
- c. paymentAmount;**

**d. client; // Represents the client associated with the payment**

### **Query:**

```
class Payment:
```

```
    def __init__(self, payment_id=None, payment_date=None,  
payment_amount=None, client=None):
```

```
        self.__payment_id = payment_id
```

```
        self.__payment_date = payment_date
```

```
        self.__payment_amount = payment_amount
```

```
        self.__client = client
```

```
# Getters
```

```
def get_payment_id(self):
```

```
    return self.__payment_id
```

```
def get_payment_date(self):
```

```
    return self.__payment_date
```

```
def get_payment_amount(self):
```

```
    return self.__payment_amount
```

```
def get_client(self):
```

```
    return self.__client
```

```
# Setters
```

```
def set_payment_id(self, payment_id):
```

```
    self.__payment_id = payment_id
```

```

def set_payment_date(self, payment_date):
    self.__payment_date = payment_date

def set_payment_amount(self, payment_amount):
    self.__payment_amount = payment_amount

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

def __str__(self):
    return f"Payment(ID: {self.__payment_id}, Date: {self.__payment_date},
Amount: {self.__payment_amount})"

```

**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**

### **Query:**

#### **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
```

### **IUserService.py:**

```
from abc import ABC, abstractmethod
```

```
class IUserService(ABC):
```

```
    @abstractmethod
    def create_user(self, user):
        pass
```

```
    @abstractmethod
    def get_user(self, username):
        pass
```

```
    @abstractmethod
    def validate_user(self, username, password):
        pass
```

```
    @abstractmethod
    def get_all_users(self):
        pass
```

```
    @abstractmethod
    def update_user(self, user):
        pass
```

```
@abstractmethod
def delete_user(self, user_id):
    pass
```

**Define InsuranceServiceImpl class and implement all the methods  
InsuranceServiceImpl**

**InsuranceServiceImpl.py:**

```
from dao.IPolicyService import IPolicyService
from entity.Policy import Policy
from exception.exceptionHandling import PolicyNotFoundException,
DatabaseError
from util.DB_Connections import DBConnections
import mysql.connector

class InsuranceServiceImpl():
    def __init__(self):
        self.connection = DBConnections.get_connection('db.properties')

    def createPolicy(self,policy):
        try:
            cursor = self.connection.cursor(dictionary = True)
            query = """
            INSERT INTO Policy(PolicyName, CoverageDetails)
            VALUES (%s, %s)
            """
            cursor.execute(query, (policy.get_policyName(),
policy.get_coverageDetails()))
            policyID = cursor.lastrowid
```



```
self.connection.commit()

print(f"Policy created successfully, ID: {policyID}")

return policyID
```

```
except mysql.connector.Error as e:

    raise DatabaseError (f"Database Error: {e.msg}") from e

finally:

    if 'cursor' in locals():

        cursor.close()
```

```
def getPolicy(self, policyID):

    try:

        cursor = self.connection.cursor(dictionary = True)

        query = "SELECT * FROM Policy WHERE PolicyID = %s"

        cursor.execute(query, (policyID,))

        policyData = cursor.fetchone()

        if policyData:

            return Policy(

                policyID = policyData['PolicyID'],

                policyName = policyData['PolicyName'],

                coverageDetails = policyData['CoverageDetails']

            )

        else:

            raise PolicyNotFoundException (f"Policy Not found: {e.msg}") from
```

e

```

except mysql.connector.Error as e:
    raise DatabaseError (f'Database Error: {e.msg}') from e
finally:
    if 'cursor' in locals():
        cursor.close()

```

```

def getAllPolicies(self):

```

```

    try:
        cursor = self.connection.cursor(dictionary = True)
        query = "SELECT * FROM Policy ORDER BY PolicyID"

```

```

        cursor.execute(query)
        policyData = cursor.fetchall()

```

```

        policies = []

```

```

        if policyData:

```

```

            for policy in policyData:

```

```

                print("-----")

```

```

                print(f'PolicyID      : {policy['PolicyID']}')

```

```

                print(f'PolicyName   : {policy['PolicyName']}')

```

```

                print(f'CoverageDetails : {policy['CoverageDetails']}')

```

```

            return

```

```

        else:

```

```

            raise PolicyNotFoundException (f'Policy Not found: {e.msg}') from

```

e

```

except mysql.connector.Error as e:
    raise DatabaseError (f'Database Error: {e.msg}') from e
finally:
    if 'cursor' in locals():
        cursor.close()

def updatePolicy(self,policy):
    try:
        cursor = self.connection.cursor(dictionary = True)
        query = """
        UPDATE Policy SET PolicyName = %s, CoverageDetails = %s
        WHERE PolicyID = %s
        """
        cursor.execute(query, (policy.get_policyName(),
policy.get_coverageDetails(), policy.get_policyID()))
        self.connection.commit()
        return cursor.rowcount > 0

    except mysql.connector.Error as e:
        raise DatabaseError (f'Database Error: {e.msg}') from e
    finally:
        if 'cursor' in locals():
            cursor.close()

def deletePolicy(self,policyID):
    try:
        cursor = self.connection.cursor(dictionary = True)

```

```

        query = "DELETE FROM Policy WHERE PolicyID = %s"
        cursor.execute(query, (policyID,))
        self.connection.commit()
        return cursor.rowcount > 0

    except mysql.connector.Error as e:
        raise DatabaseError (f"Database Error: {e.msg}") from e
    finally:
        if 'cursor' in locals():
            cursor.close()

```

### **UserServiceImpl.py:**

```

from dao.IUserService import IUserService
from entity.User import User
from util.DB_Connections import DBConnections
import mysql.connector

class UserServiceImpl(IUserService):
    def __init__(self):
        self.connection = DBConnections.get_connection('db.properties')

    def create_user(self, user):
        try:
            cursor = self.connection.cursor()

            query = "INSERT INTO User (username, password, role) VALUES (%s, %s, %s)"

            cursor.execute(query, (user.get_username(), user.get_password(),
user.get_role()))

```

```

        self.connection.commit()

    return True

except mysql.connector.Error as e:
    print(f'Error creating user: {e}')
    return False


def get_user(self, username):
    try:
        cursor = self.connection.cursor()
        query = "SELECT * FROM User WHERE username = %s"
        cursor.execute(query, (username,))
        user_data = cursor.fetchone()

        if user_data:
            return User(user_data[0], user_data[1], user_data[2], user_data[3])
        return None

    except mysql.connector.Error as e:
        print(f'Error fetching user: {e}')
        return None


def validate_user(self, username, password):
    user = self.get_user(username)
    if user and user.get_password == password:
        return user
    return None


def get_all_users(self):

```

```

try:
    cursor = self.connection.cursor()
    cursor.execute("SELECT * FROM User")
    return [User(row[0], row[1], row[2], row[3]) for row in cursor.fetchall()]
except mysql.connector.Error as e:
    print(f"Error fetching users: {e}")
    return []

```

```

def update_user(self, user):

```

```

    try:
        cursor = self.connection.cursor()
        query = """UPDATE User
                    SET username = %s, password = %s, role = %s
                    WHERE userID = %s"""
        cursor.execute(query, (user.get_username(), user.get_password(),
                               user.get_role(), user.get_userID()))
        self.connection.commit()
        return cursor.rowcount > 0
    except mysql.connector.Error as e:
        print(f"Error updating user: {e}")
        return False

```

```

def delete_user(self, userID):

```

```

    try:
        cursor = self.connection.cursor()
        cursor.execute("DELETE FROM User WHERE userID = %s",
            (userID,))
        self.connection.commit()

```

```

        return cursor.rowcount > 0
    except mysql.connector.Error as e:
        print(f'Error deleting user: {e}')
    return False

```

**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.**

**Query:**

**DB\_Properties\_Util.py:**

```

from exception.exceptionHandling import DatabaseError
import configparser
import os

```

```

class DBPropertiesUtil():

```

```

    @staticmethod

```

```

    def get_connection_string(property_file_name):

```

```

        try:

```

```

            if not os.path.exists(property_file_name):

```

```

                raise DatabaseError(f'Properties file not found:
{property_file_name}')

```

```
config = configparser.ConfigParser()
```

```
config.read(property_file_name)
```

```
if 'database' not in config:
```

```
    raise DatabaseError (f'Database not found: {e.msg}') from e
```

```
return {
```

```
    'host' : config.get('database','host'),
```

```
    'database' : config.get('database','database'),
```

```
    'user' : config.get('database','user'),
```

```
    'password' : config.get('database','password'),
```

```
    'port' : config.get('database','port')
```

```
}
```

```
    # return f'Host: {host} dbName = {database} User: {user} Password: {password} Port: {port}'
```

```
except Exception as e:
```

```
    raise DatabaseError (f'Database Error: {e}')
```

### **DB\_Connections.py:**

```
from util.DB_Properties_Util import DBPropertiesUtil
```

```
from exception.exceptionHandling import DatabaseError
```

```
import mysql.connector
```

```
class DBConnections():
```

```
    @staticmethod
```



```

def get_connection(property_file_name):
    try:
        conn_params =
DBPropertiesUtil.get_connection_string(property_file_name)
        if conn_params:

            connection = mysql.connector.connect(
                host = conn_params['host'],
                database = conn_params['database'],
                user = conn_params['user'],
                password = conn_params['password'],
                port = int(conn_params.get('port', 3306))
            )

            return connection

    except mysql.connector.Error as e:
        raise DatabaseError (f'Database Error: {e.msg}') from e

```

### **db.properties:**

```

[database]
host = localhost
database = Insurance
user = root
password = mysql
port = 3306

```

**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**

**Query:**

```
class PolicyNotFoundException(Exception):  
    def __init__(self, message = "Policy not found in the database"):  
        self.message = message  
        super().__init__(self.message)
```

```
class DatabaseError(Exception):  
    def __init__(self, message = "Database Error"):  
        self.message = message  
        super().__init__(self.message)
```

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

**Main.py:**

```
from dao.IPolicyService import IPolicyService  
from dao.InsuranceServiceImpl import InsuranceServiceImpl  
from dao.UserServiceImpl import UserServiceImpl  
from entity.Policy import Policy  
from entity.User import User  
from exception.exceptionHandling import PolicyNotFoundException,  
DatabaseError
```

```
def login(user_service):
```

```
user_service = UserServiceImpl()
print("\nLogin")
username = input("Username: ")
password = input("Password: ")
return user_service.validate_user(username, password)
```

```
def display_menu(current_user):
    while True:
        print("Here are the things you can do: ")
        print("1. Create a policy")
        print("2. View a specific policy")
        print("3. View all policies")
        print("4. Update a policy")
        print("5. Delete a policy")

        if current_user.get_role.lower() == "admin":
            print("6. User Management")
            print("7. Exit")
        else:
            print("6. Exit")

    try:
        choice = int(input("\nEnter your choice(1-6): "))

        if choice == 1:
            try:
```

```

        print("Creating a Policy: ")
        policyName = input("Enter the name of your policy: ")
        coverageDetails = input("Enter the coverage details of your policy: ")

    ")

    policy = Policy(policyName, coverageDetails)
    insurance_policy = InsuranceServiceImpl()
    if insurance_policy.createPolicy(policy):
        return
    else:
        print("Policy creation failed. Please try again later.")
except ValueError as e:
    print(f"Enter valid input. Error: [{e}]")
    continue
except Exception as e:
    print(f"Unexpected Error. Error: [{e}]")
    continue

elif choice == 2:
    try:
        print("Viewing a specific policy: ")
        policyID = int(input("Enter the policy ID: "))
        insurance_policy = InsuranceServiceImpl()
        policy = insurance_policy.getPolicy(policyID)
        print(f"\nDetails of Policy ID: {policyID}:\n")
        print(f"Policy ID: {policy.get_policyID()}")
        print(f"Policy Name: {policy.get_policyName()}")
        print(f"Coverage Details: {policy.get_coverageDetails()}")

```

```
        if policy:
            return
        else:
            raise PolicyNotFoundException (f'Policy ID: {policyID} could
not be found. [Error: {e}]")
```

```
except ValueError as e:
    print(f'Enter valid input. [Error: [{e}]]")
    continue
except Exception as e:
    print(f'Unexpected Error. [Error: {e}]]")
    continue
```

```
elif choice == 3:
```

```
    try:
        print("Viewing all policies: ")
        insurance_policy = InsuranceServiceImpl()
        policyData = insurance_policy.getAllPolicies()
        if policyData:
            print(policyData)
            return
```

```
    except DatabaseError as e:
        print(f'Database Error. [{e}]]")
        continue
```

```
    except Exception as e:
        print(f'Unexpected Error")
        continue
```

```

elif choice == 4:
    try:
        print("Updating a Policy: ")
        policyID = input("Enter the ID of the policy you wish to update: ")
        insurance_service = InsuranceServiceImpl()
        policy = insurance_service.getPolicy(policyID)
        print("\nEnter the updated details [Leave the fields blank if you
don't wish to update it]: ")

        updated_policyName = input(f'Enter the policy
name[ {policy.get_policyName()}]: ').strip()
        if not updated_policyName:
            updated_policyName = policy.get_policyName()

        updated_coverageDetails = input(f'Enter the coverageDetails
[ {policy.get_coverageDetails()}]: ').strip()
        if not updated_coverageDetails:
            updated_coverageDetails = policy.get_coverageDetails()

        policy = Policy(policyID, updated_policyName,
updated_coverageDetails)

        if insurance_service.updatePolicy(policy):
            print(f'Updated Policy {policyID} successfully!')
            return
        else:
            print("Policy updation failed. Please try again later.")

```

```
except ValueError as e:
    print(f"Enter valid input. Error: [{e}]" )
    continue
except TypeError as e:
    print(f"Enter data in the expected format. [Error: {e}]" )
    continue
except Exception as e:
    print(f"Unexpected Error. Error: [{e}]" )
    continue

elif choice == 5:
    try:
        print("Deleting a Policy: ")
        policyID = input("Enter the ID of the policy you wish to delete: ")
        insurance_policy = InsuranceServiceImpl()
        if insurance_policy.deletePolicy(policyID):
            print(f"Policy ID: {policyID} deleted successfully")
        else:
            print("Policy deletion failed.")

    except ValueError as e:
        print(f"Enter valid input. Error: [{e}]" )
        continue
    except Exception as e:
        print(f"Unexpected Error. Error: [{e}]" )
        continue
```

```
elif choice == 6 and current_user.get_role.lower() != "admin":  
    print("Exiting the system..")  
    break
```

```
elif choice == '6' and current_user.get_role().lower() == "admin":  
    user_service = UserServiceImpl()  
    user_management_menu(user_service)
```

```
except ValueError as e:  
    print(f'Please enter a number between 1 - 6 [Error: {e}]')
```

```
def user_management_menu(user_service):
```

```
    while True:
```

```
        print("\nUser Management")  
        print("1. List all users")  
        print("2. Add new user")  
        print("3. Update user")  
        print("4. Delete user")  
        print("5. Back to main menu")  
        user_service = UserServiceImpl()
```

```
    choice = input("Enter choice: ")
```

```
    if choice == '1':  
        users = user_service.get_all_users()  
        print("\nAll Users:")  
        for user in users:
```



```
print(user)
```

```
elif choice == '2':
```

```
    print("\nAdd New User")
```

```
    user_id = input("User ID: ")
```

```
    username = input("Username: ")
```

```
    password = input("Password: ")
```

```
    role = input("Role (admin/agent): ")
```

```
    new_user = User(user_id, username, password, role)
```

```
    if user_service.create_user(new_user):
```

```
        print("User created successfully!")
```

```
    else:
```

```
        print("Failed to create user")
```

```
elif choice == '3':
```

```
    user_id = input("Enter user ID to update: ")
```

```
    user = user_service.get_user_by_id(user_id) # You'll need to implement  
this
```

```
    if user:
```

```
        print(f"Current: {user}")
```

```
        user.set_username(input(f"Username ( {user.get_username()} ): ") or  
user.get_username())
```

```
        user.set_password(input("New password: ") or user.get_password())
```

```
        user.set_role(input(f"Role ( {user.get_role()} ): ") or user.get_role())
```

```
    if user_service.update_user(user):
```

```
        print("User updated successfully!")
```

```

        else:
            print("Failed to update user")
    else:
        print("User not found")

elif choice == '4':
    user_id = input("Enter user ID to delete: ")
    if user_service.delete_user(user_id):
        print("User deleted successfully!")
    else:
        print("Failed to delete user")

elif choice == '5':
    break

else:
    print("Invalid choice")

def main():
    user_service = UserServiceImpl()
    current_user = None
    while not current_user:
        current_user = login(user_service)
        if not current_user:
            print("Invalid credentials. Try again.")
            break
    print("Welcome to the insurance system!")
    while True:

```

```
display_menu(current_user)
```

```
if __name__ == "__main__":
```

```
    main()
```

## Program Output:

### 1. Creating a Policy:

#### Query:

```
if choice == 1:
    try:
        print("Creating a Policy: ")
        policyName = input("Enter the name of your policy: ")
        coverageDetails = input("Enter the coverage details of your policy: ")
        policy = Policy(policyName, coverageDetails)
        insurance_policy = InsuranceServiceImpl()
        if insurance_policy.createPolicy(policy):
            return
        else:
            print("Policy creation failed. Please try again later.")
    except ValueError as e:
        print(f"Enter valid input. Error: [{e}]")
        continue
    except Exception as e:
        print(f"Unexpected Error. Error: [{e}]")
        continue
```

## Output:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Coding Challenge\Insurance> python main.py
MySQL Database Connection has been established successfully
MySQL Database Connection has been established successfully

Login
Username: client1
Password: client123
<class 'entity.User.User'>
<class 'str'>
Welcome to the insurance system!
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): 1
Creating a Policy:
Enter the name of your policy: Health Policy
Enter the coverage details of your policy: Health and Medical Expenses
MySQL Database Connection has been established successfully
Policy created successfully, ID: 8
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): █
```

## 2. Getting a Policy:

### Query:

```
elif choice == 2:
    try:
        print("Viewing a specific policy: ")
        policyID = int(input("Enter the policy ID: "))
        insurance_policy = InsuranceServiceImpl()
        policy = insurance_policy.getPolicy(policyID)
        print(f"\nDetails of Policy ID: {policyID}:\n")
        print(f"Policy ID: {policy.get_policyID()}")
        print(f"Policy Name: {policy.get_policyName()}")
        print(f"Coverage Details: {policy.get_coverageDetails()}")

        if policy:
            return
        else:
            raise PolicyNotFoundException(f"Policy ID: {policyID} could not be found. [Error: {e}]")

    except ValueError as e:
        print(f"Enter valid input. [Error: [{e}]]")
        continue
    except Exception as e:
        print(f"Unexpected Error. [Error: {e}]]")
        continue
```

### Output:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Coding Challenge\Insurance> python main.py
MySQL Database Connection has been established successfully
MySQL Database Connection has been established successfully

Login
Username: client1
Password: client123
<class 'entity.User.User'>
<class 'str'>
Welcome to the insurance system!
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): 2
Viewing a specific policy:
Enter the policy ID: 1
MySQL Database Connection has been established successfully

Details of Policy ID: 1:

Policy ID: 1
Policy Name: Basic Health
Coverage Details: Covers hospitalization up to $50,000
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit
```

### 3. Get all policies:

#### Query:

```
elif choice == 3:
    try:
        print("Viewing all policies: ")
        insurance_policy = InsuranceServiceImpl()
        policyData = insurance_policy.getAllPolicies()
        if policyData:
            print(policyData)
            return
    except DatabaseError as e:
        print(f"Database Error. [{e}]")
        continue
    except Exception as e:
        print(f"Unexpected Error")
        continue
```

#### Output:

```
Password: client123
<class 'entity.User.User'>
<class 'str'>
Welcome to the insurance system!
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): 3
Viewing all policies:
MySQL Database Connection has been established successfully
-----
PolicyID      : 1
PolicyName    : Basic Health
CoverageDetails : Covers hospitalization up to $50,000
-----
PolicyID      : 2
PolicyName    : Premium Health
CoverageDetails : Full coverage including dental and vision
-----
PolicyID      : 3
PolicyName    : Auto Basic
CoverageDetails : Covers collision damage up to $25,000
-----
PolicyID      : 4
PolicyName    : Auto Premium
CoverageDetails : Full coverage including roadside assistance
-----
PolicyID      : 5
PolicyName    : Homeowners
CoverageDetails : Property damage and liability coverage
-----
PolicyID      : 7
PolicyName    : Health expenses
CoverageDetails : Health expensee
-----
PolicyID      : 8
```

```

PolicyID      : 7
PolicyName    : Health expenses
CoverageDetails : Health expensee
-----
PolicyID      : 8
PolicyName    : Health and Medical Expenses
CoverageDetails : None
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): █

```

## 4. Updating a Policy:

### Query:

```

elif choice == 4:
    try:
        print("Updating a Policy: ")
        policyID = input("Enter the ID of the policy you wish to update: ")
        insurance_service = InsuranceServiceImpl()
        policy = insurance_service.getPolicy(policyID)
        print("\nEnter the updated details [Leave the fields blank if you don't wish to update it]: ")

        updated_policyName = input(f"Enter the policy name [{policy.get_policyName()}]: ").strip()
        if not updated_policyName:
            updated_policyName = policy.get_policyName()

        updated_coverageDetails = input(f"Enter the coverageDetails [{policy.get_coverageDetails()}]: ").strip()
        if not updated_coverageDetails:
            updated_coverageDetails = policy.get_coverageDetails()

        policy = Policy(policyID, updated_policyName, updated_coverageDetails)

        if insurance_service.updatePolicy(policy):
            print(f"Updated Policy {policyID} successfully!")
            return
        else:
            print("Policy updation failed. Please try again later.")

    except ValueError as e:
        print(f"Enter valid input. Error: [{e}]")
        continue
    except TypeError as e:
        print(f"Enter data in the expected format. [Error: {e}]")
        continue
    except Exception as e:
        print(f"Unexpected Error. Error: [{e}]")
        continue

```

### Output:

```

PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Coding Challenge\Insurance> python main.py
MySQL Database Connection has been established successfully
MySQL Database Connection has been established successfully

Login
Username: client1
Password: client123
<class 'entity.User.User'>
<class 'str'>
Welcome to the insurance system!
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): 4
Updating a Policy:
Enter the ID of the policy you wish to update: 2
MySQL Database Connection has been established successfully

Enter the updated details [leave the fields blank if you don't wish to update it]:
Enter the policy name[Premium Health]: Health Policy
Enter the coverageDetails [Full coverage including dental and vision]: Full coverage of health details
Name: Health Policy <class 'str'>
Coverage: Full coverage of health details <class 'str'>
ID: 2 <class 'str'>
Updated Policy 2 successfully!
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): █

```

## 5. Deleting a Policy:

### Query:

```

elif choice == 5:
    try:
        print("Deleting a Policy: ")
        policyID = input("Enter the ID of the policy you wish to delete: ")
        insurance_policy = InsuranceServiceImpl()
        if insurance_policy.deletePolicy(policyID):
            print(f"Policy ID: {policyID} deleted successfully")
        else:
            print("Policy deletion failed.")

    except ValueError as e:
        print(f"Enter valid input. Error: [{e}]")
        continue
    except Exception as e:
        print(f"Unexpected Error. Error: [{e}]")
        continue

elif choice == 6 and current_user.get_role().lower() != "admin":
    print("Exiting the system..")
    break

elif choice == '6' and current_user.get_role().lower() == "admin":
    user_service = UserServiceImpl()
    user_management_menu(user_service)

except ValueError as e:
    print(f"Please enter a number between 1 - 6 [Error: {e}]")

```

## Output:

```
PS D:\Victus Laptop\Downloads\Hexaware\Python Training\Coding Challenge\Insurance> python main.py
MySQL Database Connection has been established successfully
MySQL Database Connection has been established successfully

Login
Username: client1
Password: client123
<class 'entity.User.User'>
<class 'str'>
Welcome to the insurance system!
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): 5
Deleting a Policy:
Enter the ID of the policy you wish to delete: 7
MySQL Database Connection has been established successfully
Policy ID: 7 deleted successfully
Here are the things you can do:
1. Create a policy
2. View a specific policy
3. View all policies
4. Update a policy
5. Delete a policy
6. Exit

Enter your choice(1-6): █
```