Python Coding Challenge

Swathi Baskaran

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;

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

a. clientId;

b. clientName;

c. contactInfo;

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

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

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

a. paymentId;

b. paymentDate;

c. paymentAmount;

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

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.

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.

II.

parameters: PolicyId

return type: boolean

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

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.

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.

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}"
```

Client.py:

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


    @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}"
```

Claim.py:

```
class Claim:
    def __init__(self, claimID=None, claimNumber=None, dateFilled=None,
claimAmount=None, status=None, policy=None, client=None):
        self.__claimID = claimID
        self.__claimNumber = claimNumber
```

```
        self.__dateFilled = dateFilled
        self.__claimAmount = claimAmount
        self.__status = status
        self.__policy = policy
        self.__client = client

    # Getters
    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
```

```
    # Setters
    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})"
```

Payment.py:

```
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})"
```

Policy.py:

```
class Policy:
    def __init__(self, policyID=None, policyName=None, coverageDetails=None):
        self.__policyID = policyID
        self.__policyName = policyName
        self.__coverageDetails = coverageDetails

    # Getters
    def get_policyID(self):
        return self.__policyID

    def get_policyName(self):
        return self.__policyName

    def get_coverageDetails(self):
        return self.__coverageDetails
```

```
    # Setters
    def set_policyID(self, policyID):
        self.__policyID = policyID

    def set_policyName(self, policyName):
        self.__policyName = policyName

    def set_coverageDetails(self, coverageDetails):
        self.__coverageDetails = coverageDetails

    def __str__(self):
        return f"Policy(ID: {self.__policyID}, Name: {self.__policyName},
Coverage: {self.__coverageDetails})"
```

Methods:

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
```

InsuranceServiceImp.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()
```

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
```

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
```

Output:

1. **Createpolicy():**

```
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. getPolicy():

```python
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

Enter your choice(1-6):
```

3. getAllPolicies():

```python
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. updatePolicy():

```
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. deletePolicy():

```
        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):
```