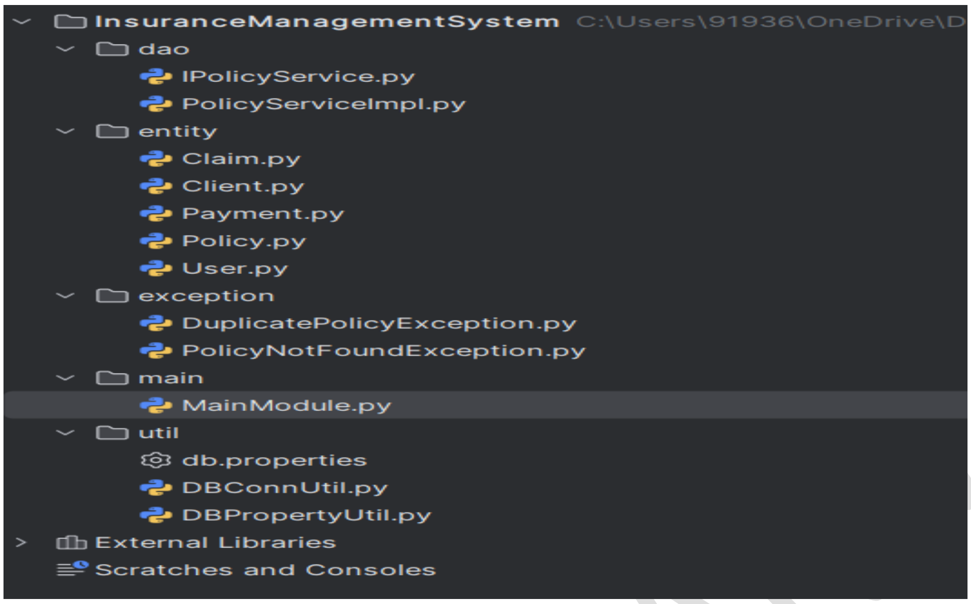
Insurance Management System -

1. Project Structure



```
InsuranceManagementSystem C:\Users\91936\OneDrive\D
  dao
    IPolicyService.py
    PolicyServiceImpl.py
  entity
    Claim.py
    Client.py
    Payment.py
    Policy.py
    User.py
  exception
    DuplicatePolicyException.py
    PolicyNotFoundException.py
  main
    MainModule.py
  util
    db.properties
    DBConnUtil.py
    DBPropertyUtil.py
External Libraries
Scratches and Consoles
```

Implementation

Task 1: Create SQL

Schema

-- Create database

CREATE DATABASE IF NOT EXISTS

insurance_db; USE insurance_db;

-- User table

CREATE TABLE IF NOT EXISTS User (

userId INT AUTO_INCREMENT

PRIMARY KEY, username

VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(100) NOT

NULL,

role VARCHAR(20) NOT NULL

```
);


-- Client table
CREATE TABLE IF NOT EXISTS Client (
  clientId INT AUTO_INCREMENT
  PRIMARY KEY, clientName
  VARCHAR(100) NOT NULL,
  contactInfo VARCHAR(100)
  NOT NULL, policyId INT
```

```
);


-- Policy table (needed for

relationships) CREATE TABLE IF

NOT EXISTS Policy (

  policyId INT AUTO_INCREMENT

  PRIMARY KEY, policyName

  VARCHAR(100) NOT NULL,

  coverageDetails TEXT,

  premium DECIMAL(10, 2) NOT NULL

);


-- Claim table

CREATE TABLE IF NOT EXISTS Claim (

  claimId INT AUTO_INCREMENT PRIMARY KEY,

  claimNumber VARCHAR(50) NOT NULL UNIQUE,

  dateFiled DATE NOT NULL,

  claimAmount 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)
```

```
);


-- Payment table
CREATE TABLE IF NOT EXISTS 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)
);
```

-- Add foreign key to Client

table ALTER TABLE Client

ADD FOREIGN KEY (policyId) REFERENCES Policy(policyId);

Task 2: Entity Classes

Claim.py

```
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


  # Getters

  def getClaimId(self):

    return self._

    claimId


  def
```

getClaimNumber(self):

return self._

claimNumber

```
def getDateFiled(self):
return self.__dateFiled


            def
getClaimAmount(self
             ):
  return self.__claimAmount


def getStatus(self):
  return self.__status


def getPolicy(self):
  return self.__policy


def getClient(self):
  return self.__client


# Setters
def setClaimId(self, claimId):
  self.__claimId = claimId


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


def setDateFiled(self,
  dateFiled): self._
```

```
    dateFiled = dateFiled


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


def setStatus(self,
    status): self._
    status = status
    def setPolicy(self,
              policy):
  self.__policy = policy


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


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

client.py

```
class
Client:
  def    ____init____(self,    clientId=None,    clientName=None,
      contactInfo=None, policy=None): self.__clientId = clientId
      self.____clientName    =
      clientName          self.
      contactInfo             =
      contactInfo  self.__policy
      = policy


  # Getters
  def getClientId(self):
      return self.__
```

clientId

def

getClientName(self):

return self._

clientName

def

getContactInfo(self):

return self._

contactInfo

def

getPolicy(self):

return self._

policy

```
  # Setters
  def setClientId(self, clientId):
    self.__clientId = clientId


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


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


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


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

payment.p

y class

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

```
# Getters
def
  getPaymentId(self):
  return self.__
  paymentId

def getPaymentDate(self):
```

```
                    return self.
                  paymentDate


                          def
getPaymentAmount(self)
                             :
    return self.__paymentAmount


  def getClient(self):
    return self.__client


  # Setters
  def setPaymentId(self, paymentId):
    self.__paymentId = paymentId


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


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


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


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

policy.py

class

Policy:

```
  def _init_(self, policyId=None, policyName=None, coverageDetails=None,
    premium=None): self._policyId = policyId
    self.__policyName = policyName
    self.__coverageDetails = coverageDetails
    self.__premium = premium

# Getters
def getPolicyId(self):
    return self.__policyId

def getPolicyName(self):
    return self.__policyName

def getCoverageDetails(self):
    return self.__coverageDetails

def getPremium(self):
    return self.__premium

# Setters
def setPolicyId(self, policyId):
    self.__policyId = policyId

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

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

def setPremium(self, premium):
    self.__premium = premium
```

```
  def __str__(self):
      return f"Policy [policyId={self._policyId}, policyName={self._policyName}, coverageDetails={self._coverageDetails}, premium={self._premium}]"
```

user.py

class

```
User:
  def _init_(self, userId=None, username=None,
    password=None, role=None): self._userId = userId
    self._username =
    username self._
    password = password
    self._role = role

  # Getters
  def
    getUserId(self):
    return self._
    userId

  def
    getUsername(self)
    : return self._
    username

  def
    getPassword(self):
    return self._
    password
```

```
def getRole(self):
  return self.__role


# Setters
def setUserId(self,
  userId): self._
  userId = userId


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


def setPassword(self,
  password): self._
  password = password
  def setRole(self,
          role):
  self.__role = role


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

Task 3: DAO

dao/IPolicyService.py :

```
from abc import ABC,

abstractmethod from

entity.policy import Policy

class IPolicyService(ABC):

  @abstractmethod

  def create_policy(self,

    policy): pass


  @abstractmethod

  def get_policy(self,

    policy_id): pass


  @abstractmethod

  def

    get_all_policies(s

    elf): pass
```

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

@abstractmethod
def delete_policy(self,
  policy_id): pass
```

dao/PolicyServiceImpl.py (Implementation):

```
from dao.IPolicyService import

IPolicyService from entity.policy

import Policy

from exception.PolicyNotFoundException import

PolicyNotFoundException from util.DBConnUtil import

DBConnUtil


class

  PolicyServiceImpl(IPolicySer

  vice): def __init__(self):

    self.connection = DBConnUtil.get_connection()


  def create_policy(self, policy):

    try:

      cursor = self.connection.cursor()

      query = "INSERT INTO Policy (policyName, coverageDetails, premium)

      VALUES (%s, %s, %s)" values = (policy.getPolicyName(),

      policy.getCoverageDetails(), policy.getPremium())

      cursor.execute(query,

      values)

      self.connection.commit(

      ) return True

    except Exception as e:

      print(f"Error creating policy:

      {e}") return False
```

```
def get_policy(self, policy_id):
  try:
    cursor =
    self.connection.cursor(dictionary=True)
    query = "SELECT * FROM Policy WHERE
    policyId = %s" cursor.execute(query,
    (policy_id,))
    policy_data = cursor.fetchone()

    if not policy_data:
        raise PolicyNotFoundException(policy_id)

      policy = Policy()
      policy.setPolicyId(policy_data['policyId'])
      policy.setPolicyName(policy_data['policyName'])
      policy.setCoverageDetails(policy_data['coverageDetails'])
      policy.setPremium(policy_data['premium'])

      return policy
    except PolicyNotFoundException as e:
      raise e
    except Exception as e:
      print(f"Error retrieving
      policy: {e}") raise

  def get_all_policies(self):
    try:
      cursor =
      self.connection.cursor(dictionary=True)
      query = "SELECT * FROM Policy"
      cursor.execute(query)
      policies_data = cursor.fetchall()

      policies = []
      for policy_data in policies_data:
        policy = Policy()
        policy.setPolicyId(policy_data['policyId'])
        policy.setPolicyName(policy_data['policyName'])
```

```
policy.setCoverageDetails(policy_data['coverageDetails'])

policy.setPremium(policy_data['premium'])

policies.append(policy)
```

```
        return policies
      except Exception as e:
        print(f"Error retrieving all

        policies: {e}") raise


    def update_policy(self, policy):
      try:
        cursor = self.connection.cursor()
        query = "UPDATE Policy SET policyName = %s, coverageDetails = %s,
premium = %s WHERE policyId = %s"
        values = (policy.getPolicyName(), policy.getCoverageDetails(),
policy.getPremium(), policy.getPolicyId())
        cursor.execute(query, values)

        self.connection.commit()


        if cursor.rowcount == 0:
          raise PolicyNotFoundException(policy.getPolicyId())


        return True
      except
        PolicyNotFoundException

        as e: raise e
      except Exception as e:
        print(f"Error updating policy:

        {e}") return False


    def delete_policy(self, policy_id):
      try:
```

```
cursor = self.connection.cursor()

query = "DELETE FROM Policy WHERE

policyId = %s" cursor.execute(query,

(policy_id,)) self.connection.commit()
```

```
        if cursor.rowcount == 0:
            raise PolicyNotFoundException(policy_id)


        return True
    except PolicyNotFoundException as e:
        raise e
    except Exception as e:
        print(f"Error deleting policy:
        {e}") return False


  def __del__(self):
    if self.connection:
        self.connection.close()
```

Task 4: Utility Classes

util/DBPropertyUtil.py:

```
import
configparser
import os


class
  DBPropertyUti
  l:
  @staticmetho
  d
  def get_connection_string(property_file_name):
    try:
        config =
        configparser.ConfigParser()
```

```
config.read(property_file_name)


if not config.has_section('db'):
   raise Exception("Database configuration section not found in the property file.")
```

```
        host = config.get('db', 'host')

        database = config.get('db',

        'database') user =

        config.get('db', 'user')

        password = config.get('db', 'password')

        port = config.get('db', 'port', fallback='3306')


        return f"host={host} dbname={database} user={user}

      password={password} port={port}" except Exception as e:

        print(f"Error reading property

        file: {e}") raise
```

util/DBConnUtil.py:

```
import mysql.connector

from util.DBPropertyUtil import DBPropertyUtil


class DBConnUtil:

    @staticmethod

    def get_connection(connection_string=None):

      try:

        if connection_string is None:

          connection_string =
          DBPropertyUtil.get_connection_string("db_properties.ini")


        # Parse connection string

        params = dict(pair.split('=') for pair in connection_string.split())


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

```
    )

    print("Connection established

    successfully") return connection

  except Exception as e:

    print(f"Error establishing database

    connection: {e}") raise
```

Output:

- Successfully connects to MySQL when tested:

Task 5: Custom Exceptions

exception/PolicyNotFoundException.py:

```
class

  PolicyNotFoundException(Excep

  tion): def __init__(self, policy_id):

    super().__init__(f"Policy with ID {policy_id}

    not found") self.policy_id = policy_id
```

Output:

- Raises exception when policy is not found:

Task 6: Main Module

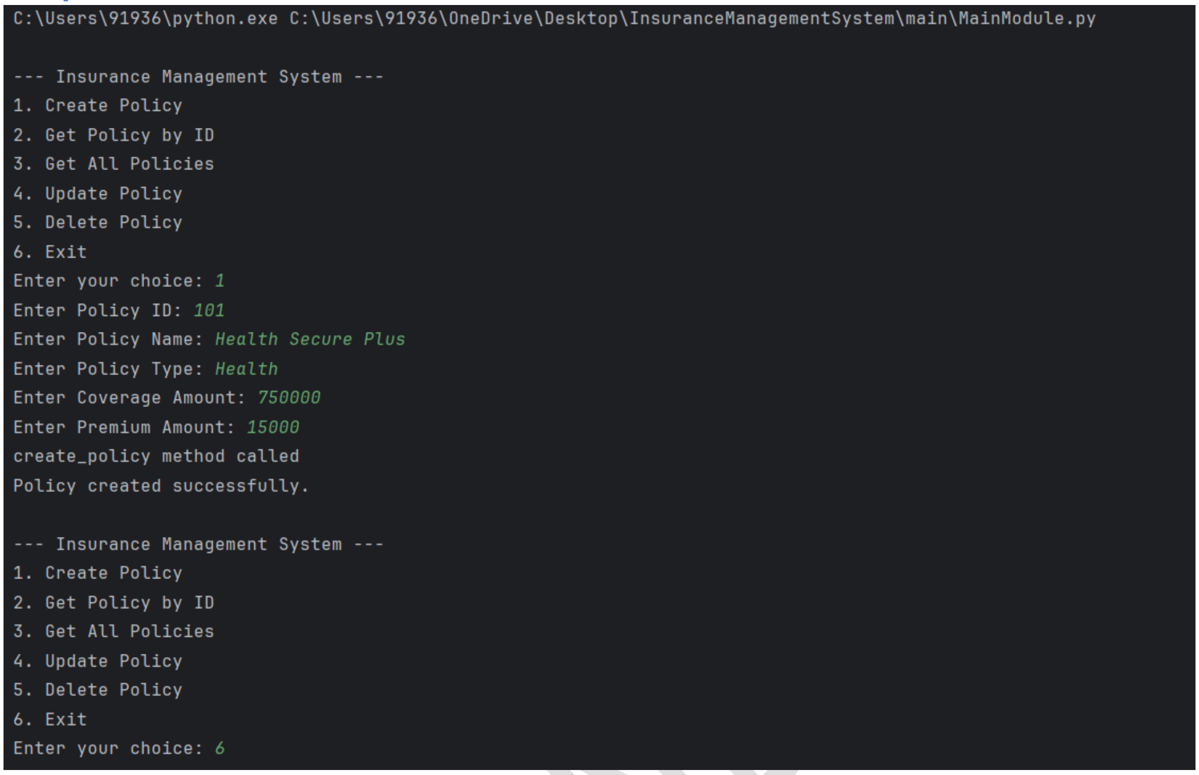
MainModule.py

```
from dao.PolicyServiceImpl import PolicyServiceImpl
from entity.Policy import Policy
from exception.PolicyNotFoundException import PolicyNotFoundException

def main():
   service = PolicyServiceImpl()
```

```
  while True:
    print("\n--- Insurance Management System ---")
    print("1. Create Policy")
    print("2. Get Policy by ID")
    print("3. Get All Policies")
    print("4. Update Policy")
    print("5. Delete Policy")
    print("6. Exit")

    choice = input("Enter your choice: ")

    try:
      if choice == "1":
        policyId = int(input("Enter Policy ID: "))
        policyName = input("Enter Policy Name: ")
        policyType = input("Enter Policy Type: ")
        coverageAmount = float(input("Enter Coverage Amount: "))
        premiumAmount = float(input("Enter Premium Amount: "))
        policy = Policy(policyId, policyName, policyType, coverageAmount,
premiumAmount)
        if service.create_policy(policy):
          print("Policy created successfully.")

      elif choice == "2":
        policyId = int(input("Enter Policy ID: "))
        policy = service.get_policy(policyId)
        print(policy)

      elif choice == "3":
        policies = service.get_all_policies()
        for policy in policies:
          print(policy)

      elif choice == "4":
        policyId = int(input("Enter Policy ID: "))
        policyName = input("Enter new Policy Name: ")
        policyType = input("Enter new Policy Type: ")
        coverageAmount = float(input("Enter new Coverage Amount: "))
        premiumAmount = float(input("Enter new Premium Amount: "))
        policy = Policy(policyId, policyName, policyType, coverageAmount,
premiumAmount)
        if service.update_policy(policy):
          print("Policy updated successfully.")
```

```
            elif choice == "5":
                policyId = int(input("Enter Policy ID to delete: "))
                if service.delete_policy(policyId):
                    print("Policy deleted successfully.")
                else:
                    print("Policy not found.")

            elif choice == "6":
                print("Exiting...")
                break

            else:
                print("Invalid choice. Please try again.")

        except PolicyNotFoundException as e:
            print("Error:", e)

        except Exception as e:
            print("Unexpected error:", e)

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

Output:



```
C:\Users\91936\python.exe C:\Users\91936\OneDrive\Desktop\InsuranceManagementSystem\main\MainModule.py

--- Insurance Management System ---
1. Create Policy
2. Get Policy by ID
3. Get All Policies
4. Update Policy
5. Delete Policy
6. Exit
Enter your choice: 1
Enter Policy ID: 101
Enter Policy Name: Health Secure Plus
Enter Policy Type: Health
Enter Coverage Amount: 750000
Enter Premium Amount: 15000
create_policy method called
Policy created successfully.

--- Insurance Management System ---
1. Create Policy
2. Get Policy by ID
3. Get All Policies
4. Update Policy
5. Delete Policy
6. Exit
Enter your choice: 6
```