Insurance Management System - Complete Documentation

1. Project Structure

The project follows the directory structure specified

```
insurance_system/
├── entity/          # Entity classes (User, Client, Claim, Payment, Policy)
├── dao/             # Service Provider Interface & Implementation
├── exception/       # Custom exceptions
├── util/            # Database utility classes
├── main/            # Main application module
└── db_properties.ini  # Database configuration
```

2. Task-wise Implementation

Task 1: Create SQL Schema

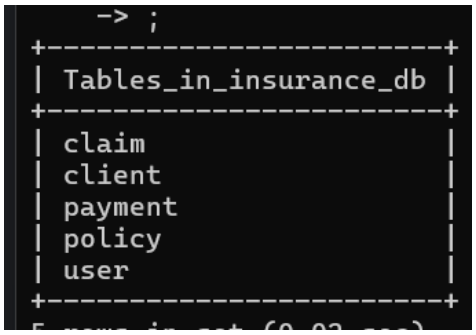
```
-- Create database
CREATE DATABASE IF NOT EXISTS insurance_db;
USE insurance_db;

-- 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.py

```
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 Layer (Service Provider Interface & Implementation)

dao/IPolicyService.py (Interface):

```
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(self):
        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(IPolicyService):
    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()
```

Output:

- Database operations work as expected (tested via MainModule).

Task 4: Utility Classes

util/DBPropertyUtil.py:

```
import configparser
import os

class DBPropertyUtil:
    @staticmethod
    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(Exception):
    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 display_menu():
    print("\nInsurance Management System")
    print("1. Create Policy")
    print("2. Get Policy")
    print("3. Get All Policies")
```

```
    print("4. Update Policy")
    print("5. Delete Policy")
    print("6. Exit")

def main():
    policy_service = PolicyServiceImpl()

    while True:
        display_menu()
        choice = input("Enter your choice: ")

        try:
            if choice == '1':
                # Create Policy
                policy = Policy()
                policy.setPolicyName(input("Enter policy name: "))
                policy.setCoverageDetails(input("Enter coverage details: "))
                policy.setPremium(float(input("Enter premium amount: ")))

                if policy_service.create_policy(policy):
                    print("Policy created successfully!")
                else:
                    print("Failed to create policy.")

            elif choice == '2':
                # Get Policy
                policy_id = int(input("Enter policy ID: "))
                policy = policy_service.get_policy(policy_id)
                print("\nPolicy Details:")
                print(policy)
```

```
        elif choice == '3':
            # Get All Policies
            policies = policy_service.get_all_policies()
            print("\nAll Policies:")
            for policy in policies:
                print(policy)
                print("-" * 50)

        elif choice == '4':
            # Update Policy
            policy_id = int(input("Enter policy ID to update: "))
            policy = policy_service.get_policy(policy_id)

            print("\nCurrent Policy Details:")
            print(policy)

            policy.setPolicyName(input("Enter new policy name (leave blank to keep current): ") or
policy.getPolicyName())
            policy.setCoverageDetails(input("Enter new coverage details (leave blank to keep current):
") or policy.getCoverageDetails())
            new_premium = input("Enter new premium (leave blank to keep current): ")
            if new_premium:
                policy.setPremium(float(new_premium))

            if policy_service.update_policy(policy):
                print("Policy updated successfully!")
            else:
                print("Failed to update policy.")

        elif choice == '5':
            # Delete Policy
            policy_id = int(input("Enter policy ID to delete: "))
```

```
            if policy_service.delete_policy(policy_id):
                print("Policy deleted successfully!")
            else:
                print("Failed to delete policy.")

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

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

    except PolicyNotFoundException as e:
        print(f"Error: {e}")
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

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

Sample Output:

```
(ins) D:\insurance_system>python MainModule.py
Connection established successfully
Insurance Management System
1. Create Policy
2. Get Policy
3. Get All Policies
4. Update Policy
5. Delete Policy
6. Exit
```

Enter your choice: 1

Enter policy name: vaanga mela polaam

Enter coverage details: pora vazhi la paapom

Enter premium amount: 1000

Policy created successfully!

Insurance Management System

1. Create Policy

2. Get Policy

3. Get All Policies

4. Update Policy

5. Delete Policy

6. Exit

Enter your choice: 2

Enter policy ID: 6

Error: Policy with ID 6 not found

Insurance Management System

1. Create Policy

2. Get Policy

3. Get All Policies

4. Update Policy

5. Delete Policy

6. Exit

Enter your choice: 3

All Policies:

Policy [policyId=1, policyName=Health Plus, coverageDetails=Comprehensive health coverage including hospitalization and outpatient care, premium=5000.00]

Policy [policyId=2, policyName=Auto Shield, coverageDetails=Vehicle insurance covering accidents, theft, and third-party liability, premium=3500.00]

Policy [policyId=3, policyName=Home Secure, coverageDetails=Home insurance covering fire, theft, and natural disasters, premium=4200.00]

Policy [policyId=4, policyName=Life Guardian, coverageDetails=Term life insurance with coverage up to 1 crore, premium=8000.00]

Policy [policyId=5, policyName=Travel Safe, coverageDetails=Travel insurance covering medical emergencies and trip cancellations, premium=1500.00]

Policy [policyId=7, policyName=vaanga mela polaam, coverageDetails=pora vazhi la paapom, premium=1000.00]

Insurance Management System

1. Create Policy

2. Get Policy

3. Get All Policies

4. Update Policy

5. Delete Policy

6. Exit

Enter your choice: 4

Enter policy ID to update: 7

Current Policy Details:

Policy [policyId=7, policyName=vaanga mela polaam, coverageDetails=pora vazhi la paapom, premium=1000.00]

Enter new policy name (leave blank to keep current):

Enter new coverage details (leave blank to keep current):

Enter new premium (leave blank to keep current): 1100

Policy updated successfully!

Insurance Management System

1. Create Policy

2. Get Policy

3. Get All Policies

4. Update Policy

5. Delete Policy

6. Exit

Enter your choice: 5

Enter policy ID to delete: 7

Policy deleted successfully!

Insurance Management System

1. Create Policy

2. Get Policy

3. Get All Policies

4. Update Policy

5. Delete Policy

6. Exit

Enter your choice: 6

Exiting the system...