

# INSURANCE MANAGEMENT SYSTEM

1. 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 ())
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.

## **Classes Creation**

1. Define `User` class with the following confidential attributes:

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

```

insurance > entity > user.py > User
1  # entity/user.py
2  class User:
3      def __init__(self):
4          pass
5
6      def __init__(self, userId, username, password, role):
7          self.__userId = userId
8          self.__username = username
9          self.__password = password
10         self.__role = role
11
12         @property
13         def userId(self):
14             return self.__userId
15
16         @userId.setter
17         def userId(self, userId):
18             self.__userId = userId
19
20         @property
21         def username(self):
22             return self.__username
23
24         @username.setter
25         def username(self, username):
26             self.__username = username
27
28         @property
29         def password(self):
30             return self.__password
31
32         @password.setter
33         def password(self, password):
34             self.__password = password
35
36         @property
37         def role(self):
38             return self.__role
39
40         @role.setter
41         def role(self, role):
42             self.__role = role
43
44         def __str__(self):
45             return f"User(ID: {self.__userId}, Username: {self.__username}, Role: {self.__role})"
46

```

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

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

```

Insurance > entity > client.py > Client > clientId
1  # entity/client.py
2  class Client:
3      def __init__(self):
4          pass
5
6      def __init__(self, clientId, clientName, contactInfo, policy):
7          self.__clientId = clientId
8          self.__clientName = clientName
9          self.__contactInfo = contactInfo
10         self.__policy = policy
11
12     @property
13     def clientId(self):
14         return self.__clientId
15
16     @clientId.setter
17     def clientId(self, clientId):
18         self.__clientId = clientId
19
20     @property
21     def clientName(self):
22         return self.__clientName
23
24     @clientName.setter
25     def clientName(self, clientName):
26         self.__clientName = clientName
27
28     @property
29     def contactInfo(self):
30         return self.__contactInfo
31
32     @contactInfo.setter
33     def contactInfo(self, contactInfo):
34         self.__contactInfo = contactInfo
35
36     @property
37     def policy(self):
38         return self.__policy
39
40     @policy.setter
41     def policy(self, policy):
42         self.__policy = policy
43
44     def __str__(self):
45         return f"Client(ID: {self.__clientId}, Name: {self.__clientName})"
46

```

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

```
Insurance > entity > claim.py > Claim
1  # entity/claim.py
2  class Claim:
3      def __init__(self):
4          pass
5
6      def __init__(self, claimId, claimNumber, dateFiled, claimAmount, status, policy, client):
7          self.__claimId = claimId
8          self.__claimNumber = claimNumber
9          self.__dateFiled = dateFiled
10         self.__claimAmount = claimAmount
11         self.__status = status
12         self.__policy = policy
13         self.__client = client
14
15     @property
16     def claimId(self):
17         return self.__claimId
18
19     @claimId.setter
20     def claimId(self, claimId):
21         self.__claimId = claimId
22
23     @property
24     def claimNumber(self):
25         return self.__claimNumber
26
27     @claimNumber.setter
28     def claimNumber(self, claimNumber):
29         self.__claimNumber = claimNumber
30
31     @property
32     def dateFiled(self):
33         return self.__dateFiled
34
35     @dateFiled.setter
36     def dateFiled(self, dateFiled):
37         self.__dateFiled = dateFiled
38
39     @property
40     def claimAmount(self):
41         return self.__claimAmount
42
```

```

@claimAmount.setter
def claimAmount(self, claimAmount):
    self.__claimAmount = claimAmount

@property
def status(self):
    return self.__status

@status.setter
def status(self, status):
    self.__status = status

@property
def policy(self):
    return self.__policy

@policy.setter
def policy(self, policy):
    self.__policy = policy

@property
def client(self):
    return self.__client

@client.setter
def client(self, client):
    self.__client = client

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

```

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

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

Insurance > entity > payment.py > Payment

```
1  # entity/payment.py
2  class Payment:
3      def __init__(self):
4          pass
5
6      def __init__(self, paymentId, paymentDate, paymentAmount, client):
7          self.__paymentId = paymentId
8          self.__paymentDate = paymentDate
9          self.__paymentAmount = paymentAmount
10         self.__client = client
11
12     @property
13     def paymentId(self):
14         return self.__paymentId
15
16     @paymentId.setter
17     def paymentId(self, paymentId):
18         self.__paymentId = paymentId
19
20     @property
21     def paymentDate(self):
22         return self.__paymentDate
23
24     @paymentDate.setter
25     def paymentDate(self, paymentDate):
26         self.__paymentDate = paymentDate
27
28     @property
29     def paymentAmount(self):
30         return self.__paymentAmount
31
32     @paymentAmount.setter
33     def paymentAmount(self, paymentAmount):
34         self.__paymentAmount = paymentAmount
35
36     @property
37     def client(self):
38         return self.__client
39
40     @client.setter
41     def client(self, client):
42         self.__client = client
43
44     def __str__(self):
45         return f"Payment(ID: {self.__paymentId}, Amount: {self.__paymentAmount})"
46
```

Policy:

```

insurance > entity > policy.py > Policy
1 # entity/policy.py
2 class Policy:
3     def __init__(self):
4         pass
5
6     def __init__(self, policyId=None, policyName="", coverage="", premium=0.0):
7         self.__policyId = policyId
8         self.__policyName = policyName
9         self.__coverage = coverage
10        self.__premium = premium
11
12    @property
13    def policyId(self):
14        return self.__policyId
15
16    @policyId.setter
17    def policyId(self, policyId):
18        self.__policyId = policyId
19
20    @property
21    def policyName(self):
22        return self.__policyName
23
24    @policyName.setter
25    def policyName(self, policyName):
26        self.__policyName = policyName
27
28    @property
29    def coverage(self):
30        return self.__coverage
31
32    @coverage.setter
33    def coverage(self, coverage):
34        self.__coverage = coverage
35
36    @property
37    def premium(self):
38        return self.__premium
39
40    @premium.setter
41    def premium(self, premium):
42        self.__premium = premium
43
44    def __str__(self):
45        return f"Policy(ID: {self.__policyId}, Name: {self.__policyName}, Coverage: {self.__coverage}, Premium: {self.__premium})"
46

```

## Database Creation:

CREATE DATABASE IF NOT EXISTS insurance;

USE insurance;

## Tables Creation:

```

CREATE TABLE policies (
    policy_id INT AUTO_INCREMENT PRIMARY KEY,
    policy_name VARCHAR(255) NOT NULL,
    coverage VARCHAR(255) NOT NULL,
    premium DECIMAL(10, 2) NOT NULL
);

```

```
mysql> CREATE TABLE policies (
->     policy_id INT AUTO_INCREMENT PRIMARY KEY,
->     policy_name VARCHAR(255) NOT NULL,
->     coverage VARCHAR(255) NOT NULL,
->     premium DECIMAL(10, 2) NOT NULL
-> );
```

Query OK, 0 rows affected (0.04 sec)

```
mysql> desc policies;
```

Field	Type	Null	Key	Default	Extra
policy_id	int	NO	PRI	NULL	auto_increment
policy_name	varchar(255)	NO		NULL	
coverage	varchar(255)	NO		NULL	
premium	decimal(10,2)	NO		NULL	

4 rows in set (0.00 sec)

```
CREATE TABLE User (
```

```
    userId INT auto_increment PRIMARY KEY,
```

```
    username VARCHAR(255),
```

```
    password VARCHAR(255),
```

```
    role VARCHAR(50)
```

```
);
```

```
mysql> CREATE TABLE User (
->     userId INT auto_increment PRIMARY KEY,
->     username VARCHAR(255),
->     password VARCHAR(255),
->     role VARCHAR(50)
-> );
```

Query OK, 0 rows affected (0.03 sec)

```
mysql> desc User;
```

Field	Type	Null	Key	Default	Extra
userId	int	NO	PRI	NULL	auto_increment
username	varchar(255)	YES		NULL	
password	varchar(255)	YES		NULL	
role	varchar(50)	YES		NULL	

4 rows in set (0.02 sec)

```
CREATE TABLE Client (
```



```

clientId INT auto_increment PRIMARY KEY,

clientName VARCHAR(255),

contactInfo VARCHAR(255),

policyId INT,

FOREIGN KEY (policyId) REFERENCES policies(policy_id)

);

```

```

mysql> CREATE TABLE Client (
  ->   clientId INT auto_increment PRIMARY KEY,
  ->   clientName VARCHAR(255),
  ->   contactInfo VARCHAR(255),
  ->   policyId INT,
  ->   FOREIGN KEY (policyId) REFERENCES policies(policy_id)
  -> );
Query OK, 0 rows affected (0.07 sec)

mysql> desc client;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| clientId   | int           | NO   | PRI | NULL    | auto_increment |
| clientName | varchar(255)  | YES  |     | NULL    |                |
| contactInfo | varchar(255)  | YES  |     | NULL    |                |
| policyId   | int           | YES  | MUL | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

```

CREATE TABLE Claim (

  claimId INT auto_increment PRIMARY KEY,

  claimNumber VARCHAR(50),

  dateFiled DATE,

  claimAmount DECIMAL(10, 2),

  status VARCHAR(50),

  policyId INT,

  clientId INT,

  FOREIGN KEY (policyId) REFERENCES Policies(policy_id),

  FOREIGN KEY (clientId) REFERENCES Client(clientId)

);

```

```
mysql> CREATE TABLE Claim (
->   claimId INT auto_increment PRIMARY KEY,
->   claimNumber VARCHAR(50),
->   dateFiled DATE,
->   claimAmount DECIMAL(10, 2),
->   status VARCHAR(50),
->   policyId INT,
->   clientId INT,
->   FOREIGN KEY (policyId) REFERENCES Policies(policy_id),
->   FOREIGN KEY (clientId) REFERENCES Client(clientId)
-> );
```

Query OK, 0 rows affected (0.10 sec)

```
mysql> desc claim;
```

Field	Type	Null	Key	Default	Extra
claimId	int	NO	PRI	NULL	auto_increment
claimNumber	varchar(50)	YES		NULL	
dateFiled	date	YES		NULL	
claimAmount	decimal(10,2)	YES		NULL	
status	varchar(50)	YES		NULL	
policyId	int	YES	MUL	NULL	
clientId	int	YES	MUL	NULL	

7 rows in set (0.00 sec)

```
CREATE TABLE Payment (
  paymentId INT auto_increment PRIMARY KEY,
  paymentDate DATE,
  paymentAmount DECIMAL(10, 2),
  clientId INT,
  FOREIGN KEY (clientId) REFERENCES Client(clientId)
);
```

```
mysql> CREATE TABLE Payment (
  ->   paymentId INT auto_increment PRIMARY KEY,
  ->   paymentDate DATE,
  ->   paymentAmount DECIMAL(10, 2),
  ->   clientId INT,
  ->   FOREIGN KEY (clientId) REFERENCES Client(clientId)
  -> );
```

Query OK, 0 rows affected (0.11 sec)

```
mysql> desc Payment;
```

Field	Type	Null	Key	Default	Extra
paymentId	int	NO	PRI	NULL	auto_increment
paymentDate	date	YES		NULL	
paymentAmount	decimal(10,2)	YES		NULL	
clientId	int	YES	MUL	NULL	

4 rows in set (0.00 sec)

3. Define IPolicyService interface/abstract class with following methods to interact with database

Keep the interfaces and implementation classes in package dao

a. createPolicy()

I. parameters: Policy Object

II. return type: boolean

b. getPolicy()

I. parameters: policyId

II. return type: Policy Object

c.getAllPolicies()

I. parameters: none

II. return type: Collection of Policy Objects

d.updatePolicy()

I. parameters: Policy Object

II. return type: boolean

e. deletePolicy()

I. parameters: PolicyId

II. return type: Boolean

```

Insurance > dao > policy_dao.py > PolicyService
1  # dao/ipolicy_service.py
2  from abc import ABC, abstractmethod
3  from entity.policy import Policy
4
5  class PolicyService(ABC):
6      @abstractmethod
7      def createPolicy(self, policy: Policy) -> bool:
8          pass
9
10     @abstractmethod
11     def getPolicy(self, policyId) -> Policy:
12         pass
13
14     @abstractmethod
15     def getAllPolicies(self) -> list:
16         pass
17
18     @abstractmethod
19     def updatePolicy(self, policy: Policy) -> bool:
20         pass
21
22     @abstractmethod
23     def deletePolicy(self, policyId) -> bool:
24         pass
25

```

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

insurance > dao > insurance\_service\_imp.py > InsuranceServiceImpl > createPolicy

```
1
2 from dao.policy_dao import PolicyService
3 from entity.policy import Policy
4 from util.db_connection import DBConnection
5 import mysql.connector
6
7 class InsuranceServiceImpl(PolicyService):
8     def createPolicy(self, policy: Policy) -> bool:
9         try:
10             connection = DBConnection.getConnection()
11             cursor = connection.cursor()
12             cursor.execute("INSERT INTO policies (policy_name, coverage, premium) VALUES (%s, %s, %s)",
13                           (policy.policyName, policy.coverage, policy.premium))
14             connection.commit()
15             cursor.close()
16             return True
17         except mysql.connector.Error as e:
18             print("Error creating policy:", e)
19             return False
20         finally:
21             if connection.is_connected():
22                 connection.close()
23
24     def getPolicy(self, policyId) -> Policy:
25         try:
26             connection = DBConnection.getConnection()
27             cursor = connection.cursor()
28             cursor.execute("SELECT * FROM policies WHERE policy_id = %s", (policyId,))
29             policy_record = cursor.fetchone()
30             cursor.close()
31             if policy_record:
32                 return Policy(policy_record[0], policy_record[1], policy_record[2], policy_record[3])
33             else:
34                 return None
35         except mysql.connector.Error as e:
36             print("Error getting policy:", e)
37             return None
38         finally:
39             if connection.is_connected():
40                 connection.close()
41
```

```
42
43     def getAllPolicies(self) -> list:
44         try:
45             connection = DBConnection.getConnection()
46             cursor = connection.cursor()
47             cursor.execute("SELECT * FROM policies")
48             policy_records = cursor.fetchall()
49             cursor.close()
50             policies = []
51             for policy_record in policy_records:
52                 policy = Policy(policy_record[0], policy_record[1], policy_record[2], policy_record[3])
53                 policies.append(policy)
54             return policies
55         except mysql.connector.Error as e:
56             print("Error getting all policies:", e)
57             return []
58         finally:
59             if connection.is_connected():
60                 connection.close()
61
62     def updatePolicy(self, policy: Policy) -> bool:
63         try:
64             connection = DBConnection.getConnection()
65             cursor = connection.cursor()
66             cursor.execute("UPDATE policies SET policy_name = %s, coverage = %s, premium = %s WHERE policy_id = %s",
67                           (policy.policyName, policy.coverage, policy.premium, policy.policyId))
68             connection.commit()
69             cursor.close()
70             return True
71         except mysql.connector.Error as e:
72             print("Error updating policy:", e)
73             return False
74         finally:
75             if connection.is_connected():
76                 connection.close()
77
```

```

75         connection.close()
76
77     def deletePolicy(self, policyId) -> bool:
78         try:
79             connection = DBConnection.getConnection()
80             cursor = connection.cursor()
81             cursor.execute("DELETE FROM policies WHERE policy_id = %s", (policyId,))
82             connection.commit()
83             cursor.close()
84             return True
85         except mysql.connector.Error as e:
86             print("Error deleting policy:", e)
87             return False
88         finally:
89             if connection.is_connected():
90                 connection.close()
91

```

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 file containing connection details like hostname, dbname, username, password, port

number and returns a connection string.

```

Insurance > util > db_connection.py > ...
1  # util/db_connection.py
2  import mysql.connector
3  from util.property_util import PropertyUtil
4  import time
5
6  class DBConnection:
7      __connection = None
8
9      @staticmethod
10     def getConnection():
11         property_file = r"C:\Users\raaji\OneDrive\Documents\hexaware training\Insurance\util\db.properties"
12         connection_string = PropertyUtil().getPropertyString(property_file)
13         try:
14             DBConnection.__connection = mysql.connector.connect(**connection_string)
15             print("Connected to MySQL database successfully.")
16             return DBConnection.__connection
17         except mysql.connector.Error as err:
18             print("Error connecting to MySQL:", err)
19             return None
20
21

```

```

Insurance > util > property_util.py > PropertyUtil > getPropertyString > property_file
1  import configparser
2
3  class PropertyUtil:
4      @staticmethod
5      def getPropertyString(property_file):
6          config = configparser.ConfigParser()
7          config.read(property_file)
8          connection_string = {
9              'user': config['DATABASE']['user'],
10             'host': config['DATABASE']['host'],
11             'port': config['DATABASE'].getint('port'),
12             'passwd': config['DATABASE']['passwd'],
13             'database': config['DATABASE']['database']
14         }
15         return connection_string
16

```

## 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

```

Insurance > exception > custom_exceptions.py > ...
1  # myexceptions/policy_exceptions.py
2  class PolicyNotFoundException(Exception):
3      def __init__(self, message="Policy not found."):
4          self.message = message
5          super().__init__(self.message)
6

```

## 9. Create class named MainModule with main method in package mainmod.

Trigger all the methods in service implementation class

Insurance > main > main\_module.py > MainModule

```
1  import sys
2  import os
3
4  sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
5
6  from dao.insurance_service_imp import InsuranceServiceImpl
7  from entity.policy import Policy
8  from exception.custom_exceptions import PolicyNotFoundException
9
10 class MainModule:
11     def __init__(self):
12         self.insurance_service = InsuranceServiceImpl()
13
14     def display_menu(self):
15         print("Insurance Management System")
16         print("1. Create Policy")
17         print("2. Get Policy")
18         print("3. Get All Policies")
19         print("4. Update Policy")
20         print("5. Delete Policy")
21         print("6. Exit")
22
23     def run(self):
24         while True:
25             self.display_menu()
26             choice = input("Enter your choice: ")
27
28             if choice == "1":
29                 self.create_policy()
30             elif choice == "2":
31                 self.get_policy()
32             elif choice == "3":
33                 self.get_all_policies()
34             elif choice == "4":
35                 self.update_policy()
36             elif choice == "5":
37                 self.delete_policy()
38             elif choice == "6":
39                 print("Exiting...")
40                 break
41             else:
42                 print("Invalid choice. Please try again.")
43
```



Insurance > main > main\_module.py > MainModule

```
10 class MainModule:
43
44     def create_policy(self):
45         # policy_id = int(input("Enter policy ID: "))
46         policy_name = input("Enter policy name: ")
47         coverage = input("Enter coverage: ")
48         premium = float(input("Enter premium: "))
49         policy = Policy(policyName=policy_name, coverage=coverage, premium=premium)
50         success = self.insurance_service.createPolicy(policy)
51         if success:
52             print("Policy created successfully.")
53         else:
54             print("Failed to create policy.")
55
56     def get_policy(self):
57         try:
58             policy_id = int(input("Enter policy ID: "))
59             policy = self.insurance_service.getPolicy(policy_id)
60             if policy is None:
61                 raise PolicyNotFoundException()
62             else:
63                 print("Policy details:")
64                 print(policy)
65         except ValueError:
66             print("Invalid input. Policy ID must be an integer.")
67         except PolicyNotFoundException as e:
68             print(e.message)
69
70     def get_all_policies(self):
71         policies = self.insurance_service.getAllPolicies()
72         if not policies:
73             print("No policies found.")
74         else:
75             print("All Policies:")
76             for policy in policies:
77                 print(policy)
78
```

```

78
79     def update_policy(self):
80         policy_id = int(input("Enter policy ID to update: "))
81         policy_name = input("Enter updated policy name: ")
82         coverage = input("Enter updated coverage: ")
83         premium = float(input("Enter updated premium: "))
84         policy = Policy(policy_id, policy_name, coverage, premium)
85         success = self.insurance_service.updatePolicy(policy)
86         if success:
87             print("Policy updated successfully.")
88         else:
89             print("Failed to update policy.")
90
91     def delete_policy(self):
92         policy_id = int(input("Enter policy ID to delete: "))
93         success = self.insurance_service.deletePolicy(policy_id)
94         if success:
95             print("Policy deleted successfully.")
96         else:
97             print("Failed to delete policy.")
98
99 if __name__ == "__main__":
100     main_module = MainModule()
101     main_module.run()
102

```

## Insurance Management System

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\raaji\OneDrive\Documents\hexaware training\Insurance> & C:/Users/raaji/AppData\Local\Programs\Python\Python39-64\python.exe hexaware training/Insurance/main/main_module.py
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.Create Policy

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\raaji\OneDrive\Documents\hexaware training\Insurance> & C:/Users/raaji/AppData/Local/Programs/Python/Python39-64/Python.exe hexaware training/Insurance/main/main_module.py"
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: Car Insurance
Enter coverage: Collision Damage
Enter premium: 500.00
Connected to MySQL database successfully.
Policy created successfully.
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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: Home Protection
Enter coverage: Property Damage
Enter premium: 100.00
Connected to MySQL database successfully.
Policy created successfully.
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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: Home Protection
Enter coverage: Theft Protection
Enter premium: 100
Connected to MySQL database successfully.
Policy created successfully.
```

## 2. Get Policy

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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: 3
Connected to MySQL database successfully.
Policy details:
Policy(ID: 3, Name: Car Insurance, Coverage: Collision Damage, Premium: 500.00)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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: 7
Connected to MySQL database successfully.
Policy not found.
```

### 3. Get All Policies

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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
Connected to MySQL database successfully.
All Policies:
Policy(ID: 3, Name: Car Insurance, Coverage: Collision Damage, Premium: 500.00)
Policy(ID: 4, Name: Home Protection, Coverage: Property Damage, Premium: 100.00)
Policy(ID: 5, Name: Home Protection, Coverage: Theft Protection, Premium: 100.00)
```

### 4. Update Policy

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
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: 5
Enter updated policy name: Home Protection
Enter updated coverage: Theft Protection
Enter updated premium: 200
Connected to MySQL database successfully.
Policy updated successfully.
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
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: 5
Connected to MySQL database successfully.
Policy details:
Policy(ID: 5, Name: Home Protection, Coverage: Theft Protection, Premium: 200.00)
```

## 5. Delete Policy

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
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: 5
Connected to MySQL database successfully.
Policy deleted successfully.
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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: 5
Connected to MySQL database successfully.
Policy not found.
```

## 6. Exit

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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...
```