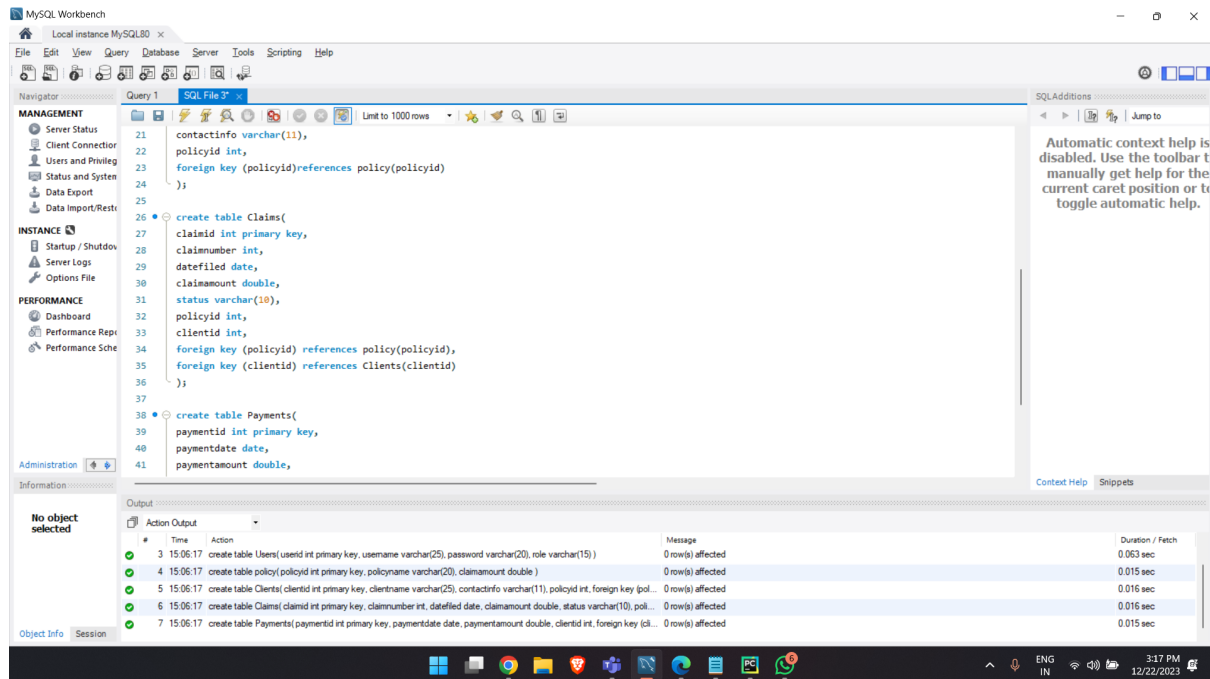
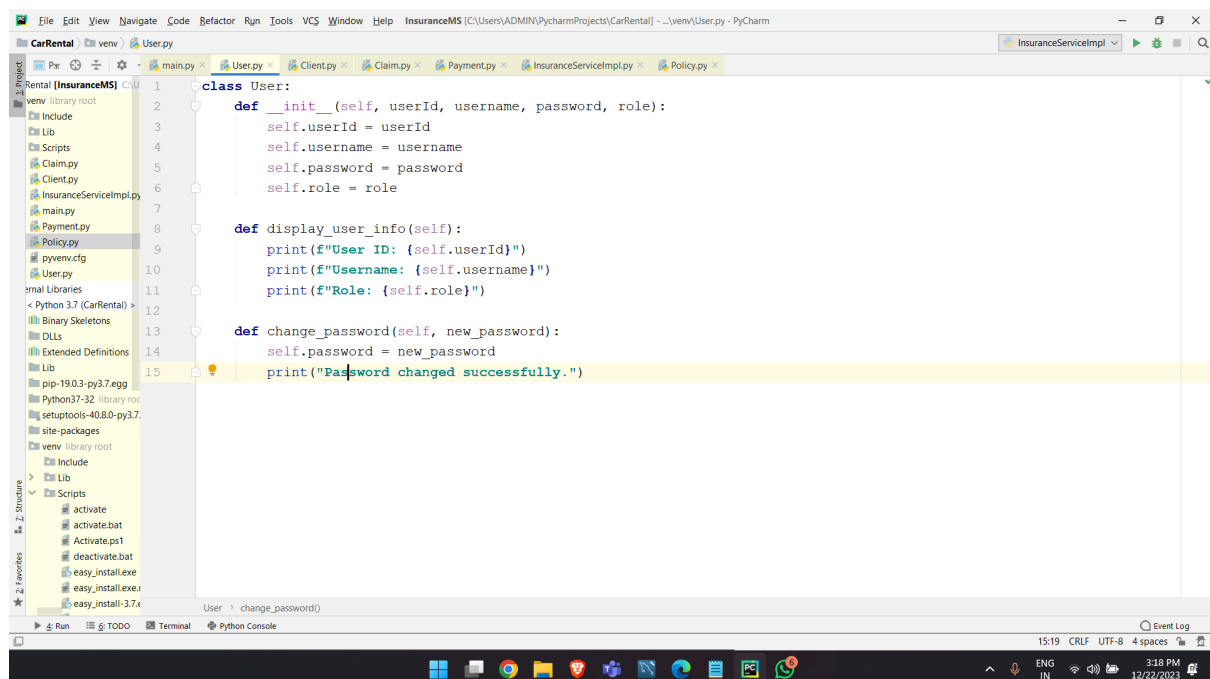


1. Create SQL Schema from the following classes, use the class attributes for table column names.

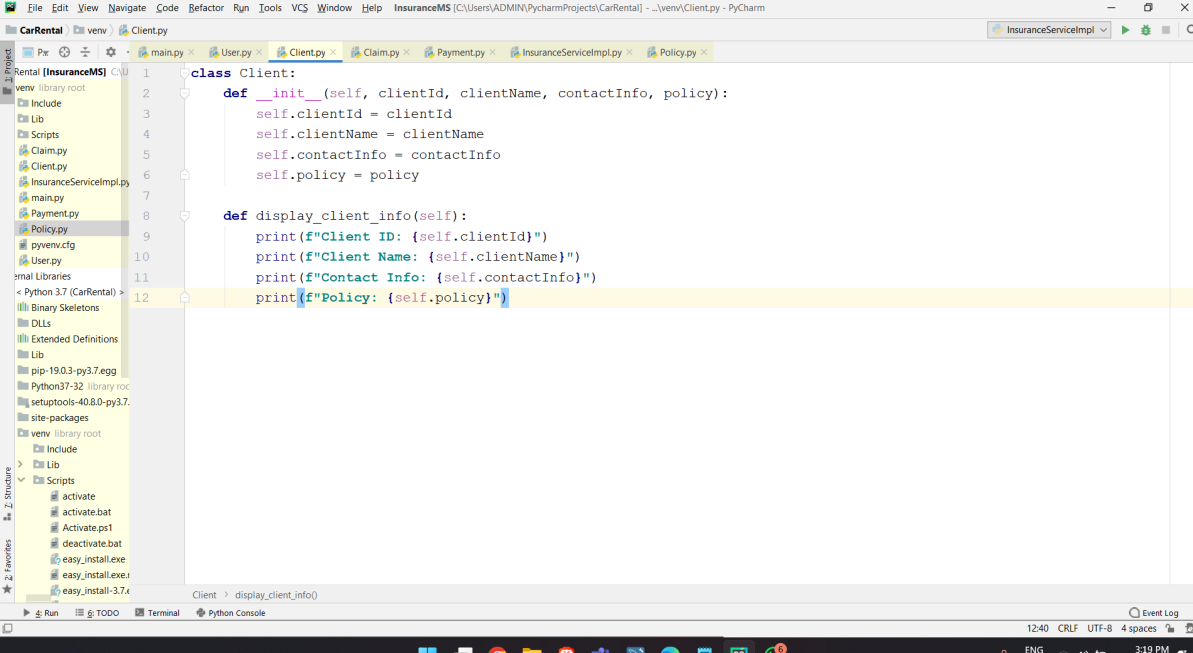


1. Create the following model/entity classes within package entity with variables declared private, constructors.

1. Define `User` class with the following confidential attributes: a. userId; b. username; c. password; d. role;



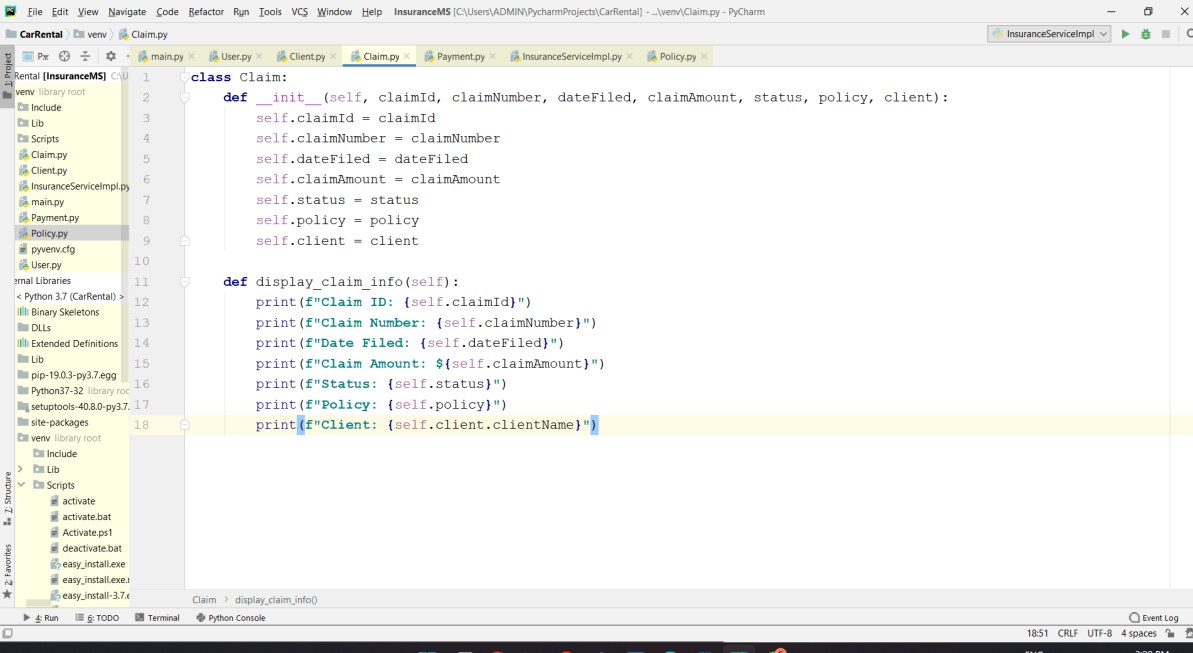
2. Define `Client` class with the following confidential attributes: a. clientId; b. clientName; c. contactInfo; d. policy; //Represents the policy associated with the client



```
1 class Client:
2     def __init__(self, clientId, clientName, contactInfo, policy):
3         self.clientId = clientId
4         self.clientName = clientName
5         self.contactInfo = contactInfo
6         self.policy = policy
7
8     def display_client_info(self):
9         print(f"Client ID: {self.clientId}")
10        print(f"Client Name: {self.clientName}")
11        print(f"Contact Info: {self.contactInfo}")
12        print(f"Policy: {self.policy}")
```

The screenshot shows the PyCharm IDE with the 'Client.py' file open. The code defines a 'Client' class with an '__init__' method that takes four arguments: 'clientId', 'clientName', 'contactInfo', and 'policy'. It also has a 'display_client_info' method that prints out these attributes. The left sidebar shows the project structure, and the bottom status bar indicates the current line is 12.

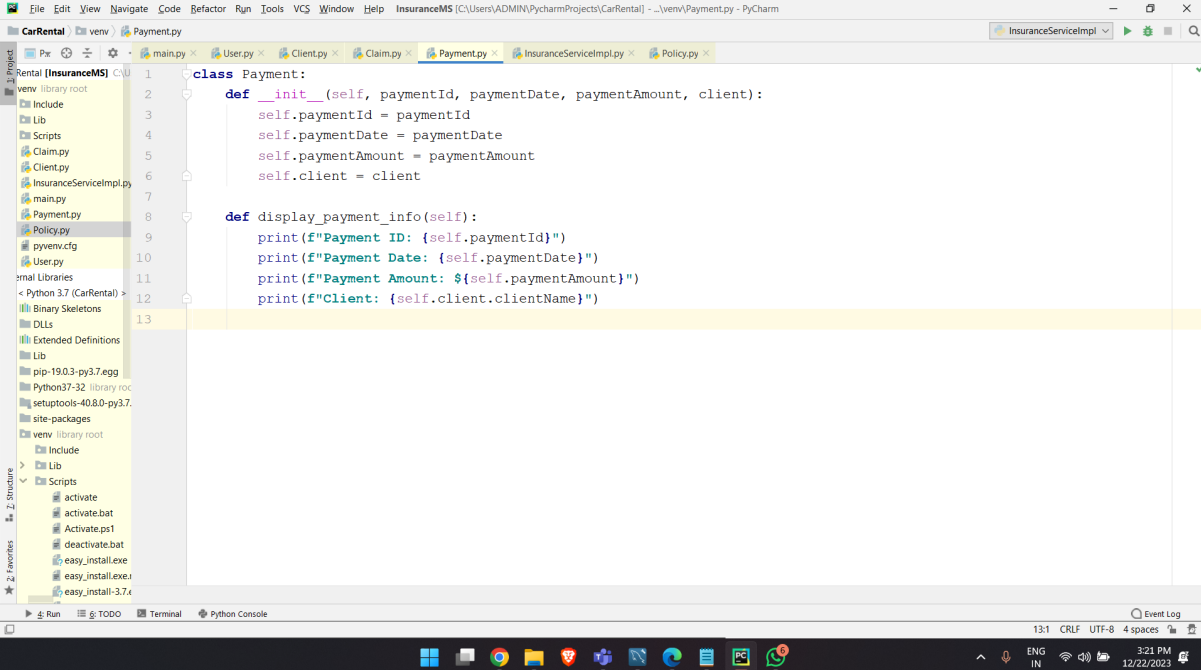
3. Define `Claim` class with the following confidential attributes: a. claimId; b. claimNumber; c. dateFiled; d. claimAmount; e. status; f. policy; //Represents the policy associated with the claim g. client; // Represents the client associated with the claim



```
1 class Claim:
2     def __init__(self, claimId, claimNumber, dateFiled, claimAmount, status, policy, client):
3         self.claimId = claimId
4         self.claimNumber = claimNumber
5         self.dateFiled = dateFiled
6         self.claimAmount = claimAmount
7         self.status = status
8         self.policy = policy
9         self.client = client
10
11     def display_claim_info(self):
12        print(f"Claim ID: {self.claimId}")
13        print(f"Claim Number: {self.claimNumber}")
14        print(f>Date Filed: {self.dateFiled}")
15        print(f"Claim Amount: ${self.claimAmount}")
16        print(f>Status: {self.status}")
17        print(f"Policy: {self.policy}")
18        print(f"Client: {self.client.clientName}")
```

The screenshot shows the PyCharm IDE with the 'Claim.py' file open. The code defines a 'Claim' class with an '__init__' method that takes seven arguments: 'claimId', 'claimNumber', 'dateFiled', 'claimAmount', 'status', 'policy', and 'client'. It also has a 'display_claim_info' method that prints out these attributes, including the client's name. The left sidebar shows the project structure, and the bottom status bar indicates the current line is 18.

4.. Define `Payment` class with the following confidential attributes: a. paymentId; b. paymentDate; c. paymentAmount; d. client; // Represents the client associated with the payment

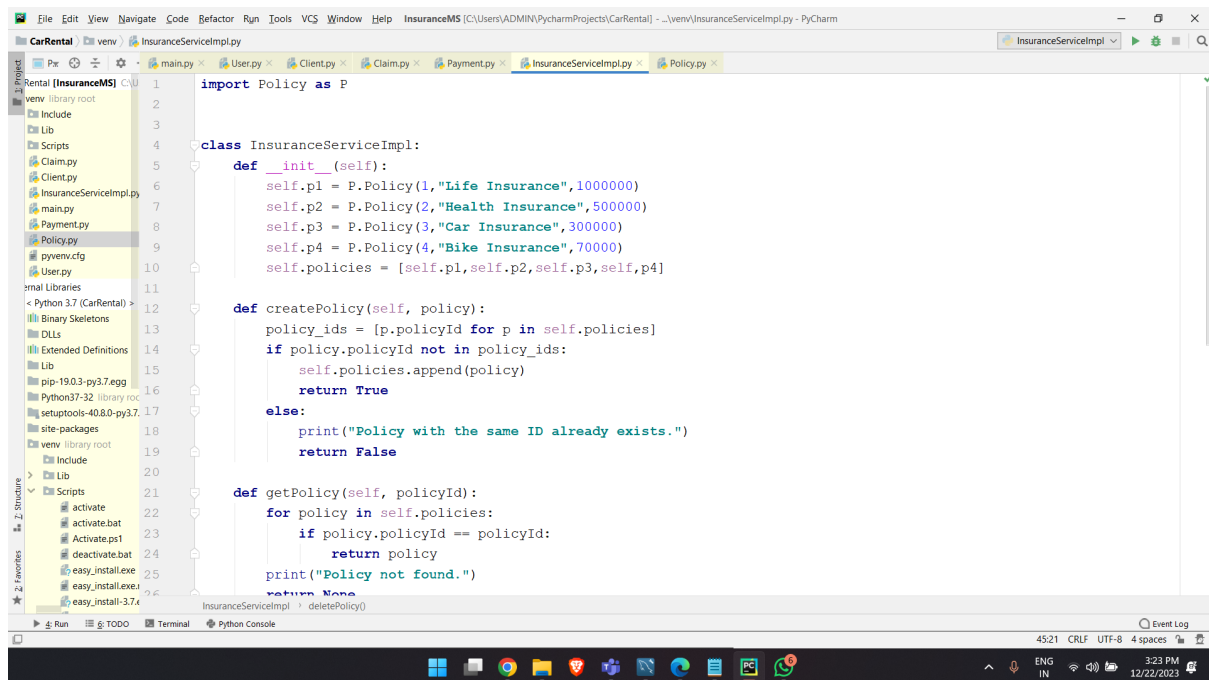


The screenshot shows the PyCharm IDE with the 'Payment.py' file open. The code defines a 'Payment' class with an '__init__' method and a 'display_payment_info' method. The left sidebar shows the project structure, and the bottom status bar indicates the file encoding and line length.

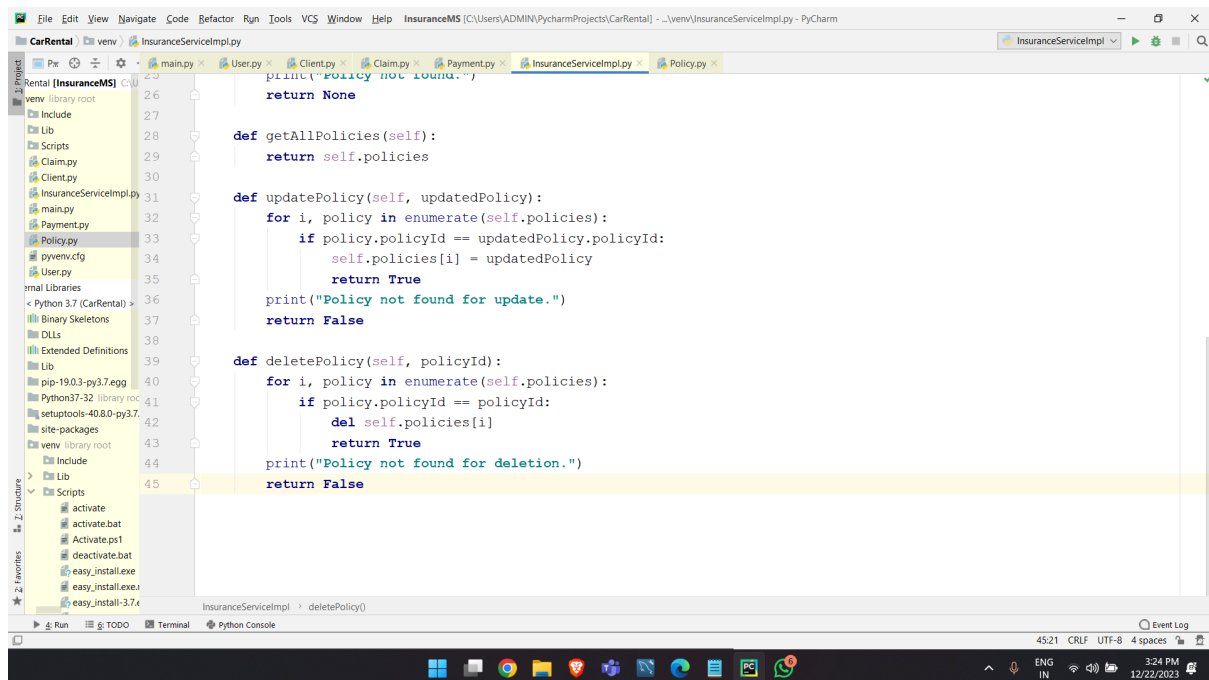
```
1 class Payment:
2     def __init__(self, paymentId, paymentDate, paymentAmount, client):
3         self.paymentId = paymentId
4         self.paymentDate = paymentDate
5         self.paymentAmount = paymentAmount
6         self.client = client
7
8     def display_payment_info(self):
9         print(f"Payment ID: {self.paymentId}")
10        print(f"Payment Date: {self.paymentDate}")
11        print(f"Payment Amount: ${self.paymentAmount}")
12        print(f"Client: {self.client.clientName}")
13
```

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 Object**
- d. updatePolicy() I. parameters: Policy Object II. return type: boolean**
- e. deletePolicy() I. parameters: PolicyId II. return type: boolean**

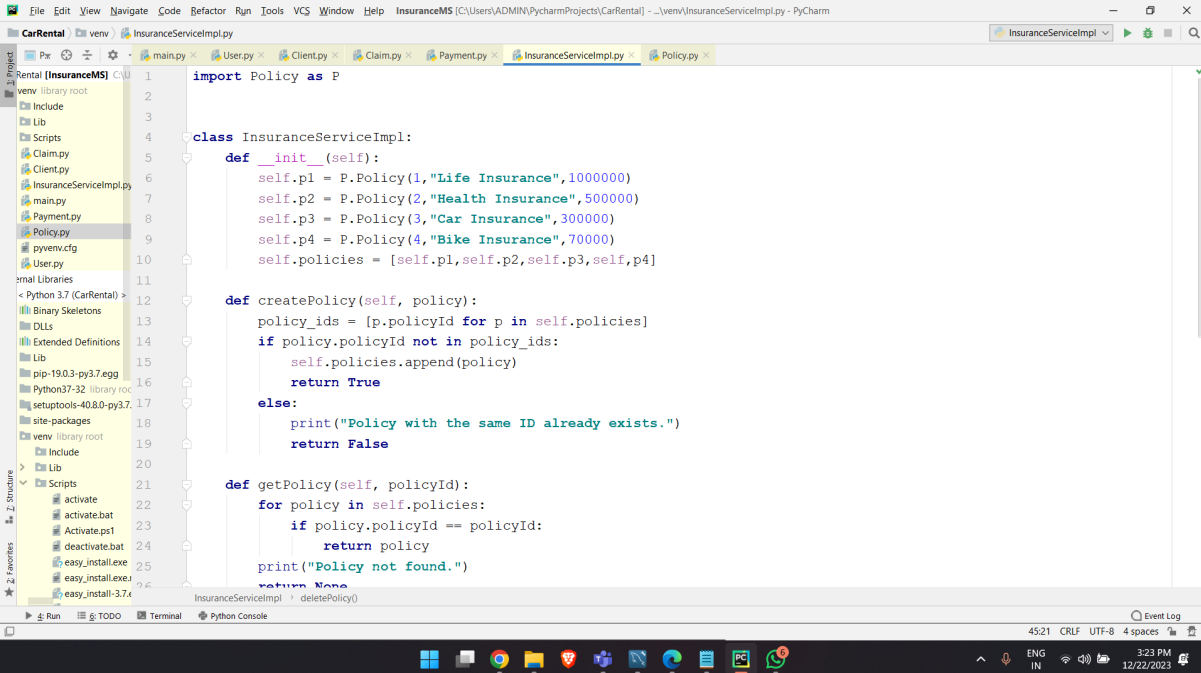


```
1 import Policy as P
2
3
4 class InsuranceServiceImpl:
5     def __init__(self):
6         self.p1 = P.Policy(1,"Life Insurance",1000000)
7         self.p2 = P.Policy(2,"Health Insurance",500000)
8         self.p3 = P.Policy(3,"Car Insurance",300000)
9         self.p4 = P.Policy(4,"Bike Insurance",70000)
10        self.policies = [self.p1,self.p2,self.p3,self.p4]
11
12    def createPolicy(self, policy):
13        policy_ids = [p.policyId for p in self.policies]
14        if policy.policyId not in policy_ids:
15            self.policies.append(policy)
16            return True
17        else:
18            print("Policy with the same ID already exists.")
19            return False
20
21    def getPolicy(self, policyId):
22        for policy in self.policies:
23            if policy.policyId == policyId:
24                return policy
25        print("Policy not found.")
26        return None
27
28    def deletePolicy()
```



```
29    def getAllPolicies(self):
30        return self.policies
31
32    def updatePolicy(self, updatedPolicy):
33        for i, policy in enumerate(self.policies):
34            if policy.policyId == updatedPolicy.policyId:
35                self.policies[i] = updatedPolicy
36                return True
37        print("Policy not found for update.")
38        return False
39
40    def deletePolicy(self, policyId):
41        for i, policy in enumerate(self.policies):
42            if policy.policyId == policyId:
43                del self.policies[i]
44                return True
45        print("Policy not found for deletion.")
46        return False
```

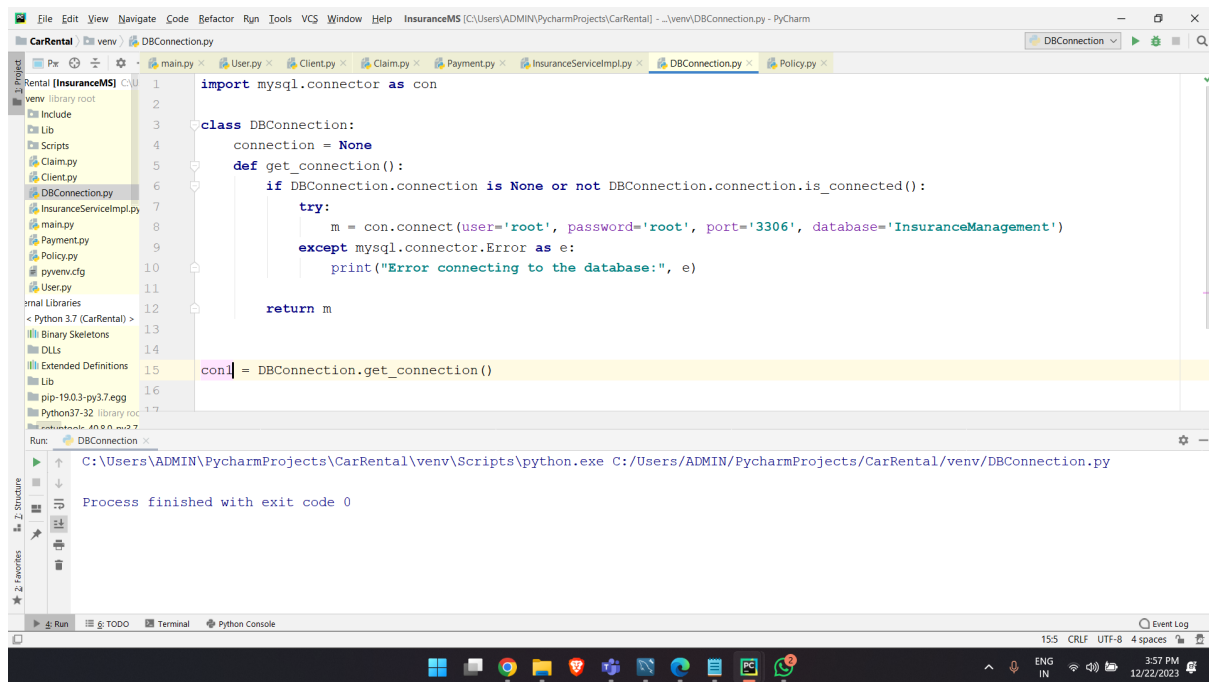
6. Define InsuranceServiceImpl class and implement all the methods InsuranceServiceImpl



```
1 import Policy as P
2
3
4 class InsuranceServiceImpl:
5     def __init__(self):
6         self.p1 = P.Policy(1, "Life Insurance", 1000000)
7         self.p2 = P.Policy(2, "Health Insurance", 500000)
8         self.p3 = P.Policy(3, "Car Insurance", 300000)
9         self.p4 = P.Policy(4, "Bike Insurance", 70000)
10        self.policies = [self.p1, self.p2, self.p3, self.p4]
11
12    def createPolicy(self, policy):
13        policy_ids = [p.policyId for p in self.policies]
14        if policy.policyId not in policy_ids:
15            self.policies.append(policy)
16            return True
17        else:
18            print("Policy with the same ID already exists.")
19            return False
20
21    def getPolicy(self, policyId):
22        for policy in self.policies:
23            if policy.policyId == policyId:
24                return policy
25        print("Policy not found.")
26        return None
```

The image shows a PyCharm IDE window titled "InsuranceMS [C:\Users\ADMIN\PycharmProjects\CarRental] - \venv\InsuranceServiceImpl.py - PyCharm". The left sidebar displays the project structure for "CarRental", including a "venv" directory with "lib" and "Scripts" folders, and a "src" directory with files like "main.py", "User.py", "Client.py", "Claim.py", "Payment.py", "InsuranceServiceImpl.py", and "Policy.py". The "InsuranceServiceImpl.py" file is open in the editor, showing the Python code for the "InsuranceServiceImpl" class. The code includes an import for "Policy" as "P", an initialization method "__init__" that creates four "Policy" objects (Life Insurance, Health Insurance, Car Insurance, and Bike Insurance) and stores them in a list "self.policies", and three methods: "createPolicy" which checks if a policy ID already exists and appends the policy if not, "getPolicy" which iterates through the policies to find one by ID, and a "deletePolicy" method (partially visible at the bottom). The bottom status bar shows "4521 CRLF UTF-8 4 spaces" and the date "12/22/2023".

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.



```
1 import mysql.connector as con
2
3 class DBConnection:
4     connection = None
5     def get_connection():
6         if DBConnection.connection is None or not DBConnection.connection.is_connected():
7             try:
8                 m = con.connect(user='root', password='root', port='3306', database='InsuranceManagement')
9             except mysql.connector.Error as e:
10                 print("Error connecting to the database:", e)
11
12         return m
13
14
15 con1 = DBConnection.get_connection()
```

Run: DBConnection

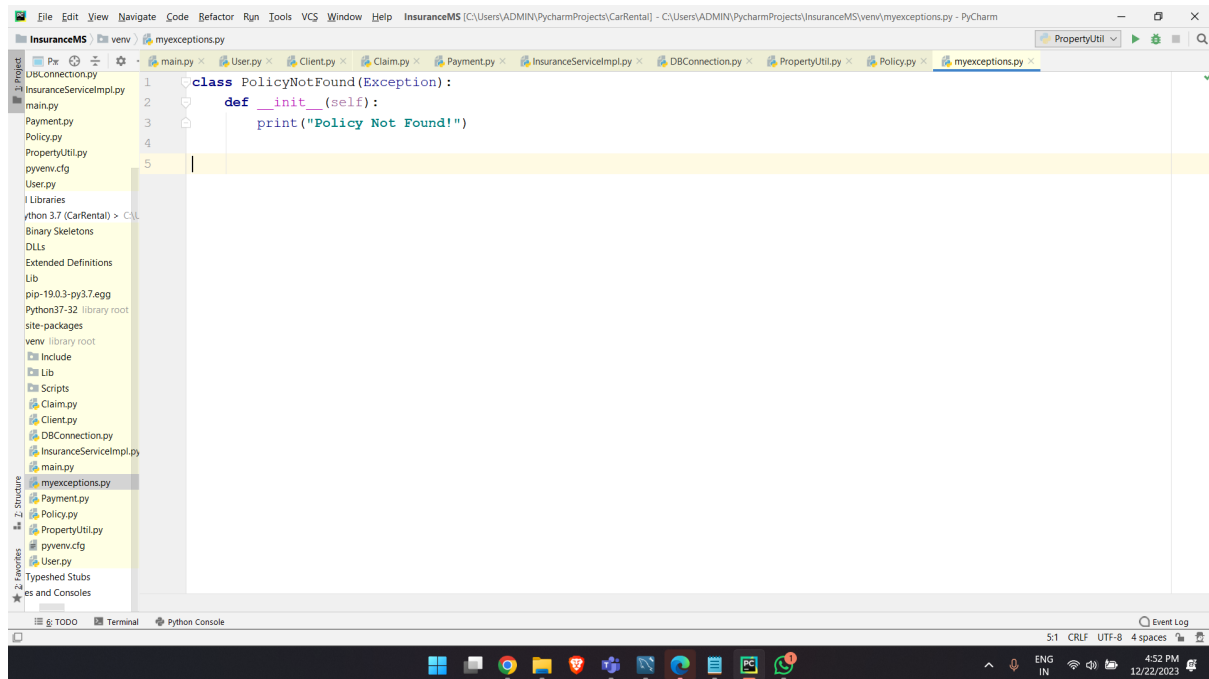
C:\Users\ADMIN\PycharmProjects\CarRental\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/CarRental/venv/DBConnection.py

Process finished with exit code 0

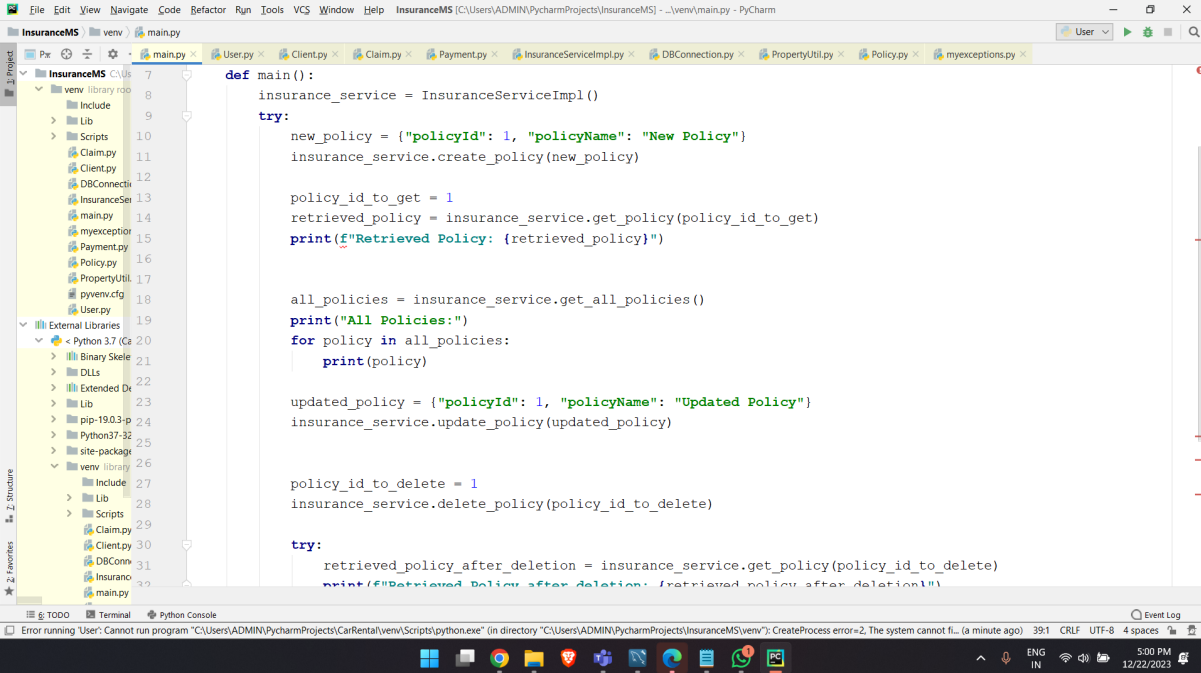
Connection properties supplied in the connection string should be read from a property file.

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.



```
def main():
    insurance_service = InsuranceServiceImpl()

    try:
        new_policy = {"policyId": 1, "policyName": "New Policy"}
        insurance_service.create_policy(new_policy)

        policy_id_to_get = 1
        retrieved_policy = insurance_service.get_policy(policy_id_to_get)
        print(f"Retrieved Policy: {retrieved_policy}")

        all_policies = insurance_service.get_all_policies()
        print("All Policies:")
        for policy in all_policies:
            print(policy)

        updated_policy = {"policyId": 1, "policyName": "Updated Policy"}
        insurance_service.update_policy(updated_policy)

        policy_id_to_delete = 1
        insurance_service.delete_policy(policy_id_to_delete)

    try:
        retrieved_policy_after_deletion = insurance_service.get_policy(policy_id_to_delete)
        print(f"Retrieved Policy after deletion: {retrieved_policy_after_deletion}")
```

The screenshot shows the PyCharm IDE with the 'InsuranceMS' project. The 'main.py' file is open, displaying a `main()` function. This function initializes an `InsuranceServiceImpl` object and calls several methods: `create_policy`, `get_policy`, `get_all_policies`, `update_policy`, and `delete_policy`. It also includes a final `try` block that calls `get_policy` again to verify deletion. The left sidebar shows the project structure, and the bottom status bar indicates an error running the program.