

Coding Challenge - Loan Management System

NAME : HARSHA K

Problem Statement:

Create SQL Schema from the customer and loan class, use the class attributes for table column names.

1. Define a `Customer` class with the following confidential attributes:

- a. Customer ID
- b. Name
- c. Email Address
- d. Phone Number
- e. Address
- f. creditScore

Code:

```
class Customer:
```

```
    def __init__(self, customer_id, name, email, phone, address, credit_score):  
        self.__customer_id = customer_id  
        self.__name = name  
        self.__email = email  
        self.__phone = phone  
        self.__address = address  
        self.__credit_score = credit_score
```

```
entity > 📁 customer.py > ...  
1   class Customer:  
2       def __init__(self, customer_id, name, email, phone, address, credit_score):  
3           self.__customer_id = customer_id  
4           self.__name = name  
5           self.__email = email  
6           self.__phone = phone  
7           self.__address = address  
8           self.__credit_score = credit_score  
9
```

Sql Schema:

```
CREATE TABLE customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    address VARCHAR(255),
    credit_score INT);
```

```
mysql> use loan;
Database changed
mysql> CREATE TABLE customer (
    ->     customer_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(100),
    ->     email VARCHAR(100) UNIQUE,
    ->     phone VARCHAR(15),
    ->     address VARCHAR(255),
    ->     credit_score INT
    -> );
Query OK, 0 rows affected (0.13 sec)
```

2. Define a base class `Loan` with the following attributes:

- a. loanId
- b. customer (reference of customer class)
- c. principalAmount
- d. interestRate
- e. loanTerm (Loan Tenure in months)
- f. loanType (CarLoan, HomeLoan)
- g. loanStatus (Pending, Approved)

Code:

```
from .customer import Customer

class Loan:
    def __init__(self, loan_id, customer: Customer, principal_amount, interest_rate, loan_term,
                 loan_type, loan_status):
```

```
self.__loan_id = loan_id  
self.__customer = customer  
self.__principal_amount = principal_amount  
self.__interest_rate = interest_rate  
self.__loan_term = loan_term  
self.__loan_type = loan_type  
self.__loan_status = loan_status
```

```
entity > loan.py > ...  
1  from .customer import Customer  
2  class Loan:  
3      def __init__(self, loan_id, customer: Customer, principal_amount, interest_rate, loan_term, loan_type, loan_status):  
4          self.__loan_id = loan_id  
5          self.__customer = customer  
6          self.__principal_amount = principal_amount  
7          self.__interest_rate = interest_rate  
8          self.__loan_term = loan_term  
9          self.__loan_type = loan_type  
10         self.__loan_status = loan_status  
11
```

Sql Schema:

```
CREATE TABLE loan (  
    loan_id INT PRIMARY KEY AUTO_INCREMENT,  
    customer_id INT,  
    principal_amount DECIMAL(12,2),  
    interest_rate DECIMAL(5,2),  
    loan_term INT,  
    loan_type ENUM('CarLoan', 'HomeLoan'),  
    loan_status ENUM('Pending', 'Approved'),  
    FOREIGN KEY (customer_id) REFERENCES customer(customer_id)  
);
```

```

mysql> CREATE TABLE loan (
    ->     loan_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     customer_id INT,
    ->     principal_amount DECIMAL(12,2),
    ->     interest_rate DECIMAL(5,2),
    ->     loan_term INT,
    ->     loan_type ENUM('CarLoan', 'HomeLoan'),
    ->     loan_status ENUM('Pending', 'Approved'),
    ->     FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
    -> );
Query OK, 0 rows affected (0.10 sec)

```

3. Create two subclasses: `HomeLoan` and `CarLoan`. These subclasses should inherit from the `Loan` class and add attributes specific to their loan types. For example:

a. `HomeLoan` should have a `propertyAddress` (String) and `propertyValue` (int) attribute.

Code:

```

from entity.loan import Loan

class HomeLoan(Loan):

    def __init__(self, loan_id, customer, principal_amount, interest_rate, loan_term, loan_type,
                 loan_status, property_address, property_value):

        super().__init__(loan_id, customer, principal_amount, interest_rate, loan_term, loan_type,
                        loan_status)

        self.__property_address = property_address

        self.__property_value = property_value

```

```

entity > home.loan.py > ...
1  from entity.loan import Loan
2
3  class HomeLoan(Loan):
4      def __init__(self, loan_id, customer, principal_amount, interest_rate, loan_term, loan_type, loan_status, property_address, property_value):
5          super().__init__(loan_id, customer, principal_amount, interest_rate, loan_term, loan_type, loan_status)
6          self.__property_address = property_address
7          self.__property_value = property_value
8

```

b. `CarLoan` should have a `carModel` (String) and `carValue` (int) attribute.

Code:

```
from entity.loan import Loan
```

```
class CarLoan(Loan):
```

```
    def __init__(self, loan_id, customer, principal_amount, interest_rate, loan_term, loan_type,
                 loan_status, car_model, car_value):
```

```

super().__init__(loan_id, customer, principal_amount, interest_rate, loan_term, loan_type,
loan_status)

self.__car_model = car_model

self.__car_value = car_value

```

```

entity > car_loan.py > ...
Close (Ctrl+F4)

1 from entity.loan import Loan
2
3 class CarLoan(Loan):
4     def __init__(self, loan_id, customer, principal_amount, interest_rate, loan_term, loan_type, loan_status, car_model, car_value):
5         super().__init__(loan_id, customer, principal_amount, interest_rate, loan_term, loan_type, loan_status)
6         self.__car_model = car_model
7         self.__car_value = car_value
8

```

4. Implement the following for all classes.

a. Write default constructors and overload the constructor with parameters, generate getter and setter, (print all information of attribute) methods for the attributes.

Customer code:

Getter setter

```

def get_customer_id(self): return self.__customer_id

def set_customer_id(self, customer_id): self.__customer_id = customer_id

```

```

def get_name(self): return self.__name

def set_name(self, name): self.__name = name

```

```

def get_email(self): return self.__email

def set_email(self, email): self.__email = email

```

```

def get_phone(self): return self.__phone

def set_phone(self, phone): self.__phone = phone

```

```

def get_address(self): return self.__address

```

```

def set_address(self, address): self.__address = address

def get_credit_score(self): return self.__credit_score

def set_credit_score(self, credit_score): self.__credit_score = credit_score

# Print

def display_info(self):

    print(f"Customer ID: {self.__customer_id}")

    print(f"Name: {self.__name}")

    print(f"Email: {self.__email}")

    print(f"Phone: {self.__phone}")

    print(f"Address: {self.__address}")

    print(f"Credit Score: {self.__credit_score}")

```

```

1   class Customer:
2
3       # Getter setter
4       def get_customer_id(self): return self.__customer_id
5       def set_customer_id(self, customer_id): self.__customer_id = customer_id
6
7       def get_name(self): return self.__name
8       def set_name(self, name): self.__name = name
9
10      def get_email(self): return self.__email
11      def set_email(self, email): self.__email = email
12
13      def get_phone(self): return self.__phone
14      def set_phone(self, phone): self.__phone = phone
15
16      def get_address(self): return self.__address
17      def set_address(self, address): self.__address = address
18
19      def get_credit_score(self): return self.__credit_score
20      def set_credit_score(self, credit_score): self.__credit_score = credit_score
21
22      # Print
23      def display_info(self):
24          print(f"Customer ID: {self.__customer_id}")
25          print(f"Name: {self.__name}")
26          print(f"Email: {self.__email}")
27          print(f"Phone: {self.__phone}")
28          print(f"Address: {self.__address}")
29          print(f"Credit Score: {self.__credit_score}")
30
31
32
33
34
35
36
37

```

Loan Code:

```
def get_loan_id(self): return self.__loan_id

def set_loan_id(self, loan_id): self.__loan_id = loan_id

def get_customer(self): return self.__customer

def set_customer(self, customer): self.__customer = customer

def get_principal_amount(self): return self.__principal_amount

def set_principal_amount(self, amount): self.__principal_amount = amount

def get_interest_rate(self): return self.__interest_rate

def set_interest_rate(self, rate): self.__interest_rate = rate

def get_loan_term(self): return self.__loan_term

def set_loan_term(self, term): self.__loan_term = term

def get_loan_type(self): return self.__loan_type

def set_loan_type(self, loan_type): self.__loan_type = loan_type

def get_loan_status(self): return self.__loan_status

def set_loan_status(self, loan_status): self.__loan_status = loan_status

def display_info(self):
    print(f"Loan ID: {self.__loan_id}")
    print(f"Customer: {self.__customer}")
    print(f"Principal Amount: {self.__principal_amount}")
    print(f"Interest Rate: {self.__interest_rate}")
    print(f"Loan Term: {self.__loan_term} months")
    print(f"Loan Type: {self.__loan_type}")
    print(f"Loan Status: {self.__loan_status}")
```

```

entity > loan.py > Loan > display_info
 1  class Loan:
12      def get_loan_id(self): return self.__loan_id
13      def set_loan_id(self, loan_id): self.__loan_id = loan_id
14
15      def get_customer(self): return self.__customer
16      def set_customer(self, customer): self.__customer = customer
17
18      def get_principal_amount(self): return self.__principal_amount
19      def set_principal_amount(self, amount): self.__principal_amount = amount
20
21      def get_interest_rate(self): return self.__interest_rate
22      def set_interest_rate(self, rate): self.__interest_rate = rate
23
24      def get_loan_term(self): return self.__loan_term
25      def set_loan_term(self, term): self.__loan_term = term
26
27      def get_loan_type(self): return self.__loan_type
28      def set_loan_type(self, loan_type): self.__loan_type = loan_type
29
30      def get_loan_status(self): return self.__loan_status
31      def set_loan_status(self, loan_status): self.__loan_status = loan_status
32
33      def display_info(self):
34          print(f"Loan ID: {self.__loan_id}")
35          print(f"Customer: {self.__customer}")
36          print(f"Principal Amount: {self.__principal_amount}")
37          print(f"Interest Rate: {self.__interest_rate}")
38          print(f"Loan Term: {self.__loan_term} months")
39          print(f"Loan Type: {self.__loan_type}")
40          print(f"Loan Status: {self.__loan_status}")
41

```

Home_loan:

```

def get_property_address(self): return self.__property_address

def set_property_address(self, address): self.__property_address = address

```

```

def get_property_value(self): return self.__property_value

def set_property_value(self, value): self.__property_value = value

```

```

def display_info(self):
    super().display_info()

    print(f"Property Address: {self.__property_address}")

    print(f"Property Value: {self.__property_value}")

```

```
entity > home_loan.py > ...
3   class HomeLoan(Loan):
4
5       def get_property_address(self): return self.__property_address
6       def set_property_address(self, address): self.__property_address = address
7
8       def get_property_value(self): return self.__property_value
9       def set_property_value(self, value): self.__property_value = value
10
11      def display_info(self):
12          super().display_info()
13          print(f"Property Address: {self.__property_address}")
14          print(f"Property Value: {self.__property_value}")
15
16
17
18
19
20
```

Car loan:

```
def get_car_model(self): return self.__car_model
def set_car_model(self, model): self.__car_model = model

def get_car_value(self): return self.__car_value
def set_car_value(self, value): self.__car_value = value

def display_info(self):
    super().display_info()
    print(f"Car Model: {self.__car_model}")
    print(f"Car Value: {self.__car_value}")
```

```
entity > car_loan.py > ...
3   class CarLoan(Loan):
4
5       def get_car_model(self): return self.__car_model
6       def set_car_model(self, model): self.__car_model = model
7
8       def get_car_value(self): return self.__car_value
9       def set_car_value(self, value): self.__car_value = value
10
11      def display_info(self):
12          super().display_info()
13          print(f"Car Model: {self.__car_model}")
14          print(f"Car Value: {self.__car_value}")
15
16
17
18
19
20
```

5. Define ILoanRepository interface/abstract class with following methods to interact with database.

Code:

```
from abc import ABC, abstractmethod

class ILoanRepository(ABC):

    @abstractmethod
    def apply_loan(self, loan):
        pass

    @abstractmethod
    def calculate_interest(self, loan_id):
        pass

    @abstractmethod
    def calculate_interest_manual(self, principal, rate, term):
        pass

    @abstractmethod
    def loan_status(self, loan_id):
        pass

    @abstractmethod
    def calculate_emi(self, loan_id):
        pass

    @abstractmethod
    def calculate_emi_manual(self, principal, rate, term):
        pass

    @abstractmethod
    def loan_repayment(self, loan_id, amount):
        pass

    @abstractmethod
    def get_all_loans(self):
        pass

    @abstractmethod
    def get_loan_by_id(self, loan_id):
        pass
```

```
dao > interface > i_loan_repository.py > ...
1   from abc import ABC, abstractmethod
2
3   class ILoanRepository(ABC):
4
5       @abstractmethod
6       def apply_loan(self, loan):
7           pass
8
9       @abstractmethod
10      def calculate_interest(self, loan_id):
11          pass
12
13      @abstractmethod
14      def calculate_interest_manual(self, principal, rate, term):
15          pass
16
17      @abstractmethod
18      def loan_status(self, loan_id):
19          pass
20
21      @abstractmethod
22      def calculate_emi(self, loan_id):
23          pass
24
25      @abstractmethod
26      def calculate_emi_manual(self, principal, rate, term):
27          pass
28
29      @abstractmethod
30      def loan_repayment(self, loan_id, amount):
31          pass
32
33      @abstractmethod
34      def get_all_loans(self):
35          pass
36
37      @abstractmethod
38      def get_loan_by_id(self, loan_id):
39          pass
40
```

a.applyLoan(loan Loan): pass appropriate parameters for creating loan. Initially loan status is pending and stored in database. before storing in database get confirmation from the user as Yes/No

Code:

```
import mysql.connector
from dao.interface.i_loan_repository import ILoanRepository
from exception.invalid_loan_exception import InvalidLoanException
from util.db_conn_util import DBConnUtil
from entity.loan import Loan

class LoanRepositoryImpl(ILoanRepository):

    def apply_loan(self, loan):
        try:
            confirm = input("Do you want to apply for the loan? (Yes/No): ").strip().lower()
            if confirm != 'yes':
                print("Loan application cancelled.")
                return
            conn = DBConnUtil.get_connection()
            cursor = conn.cursor()
            query = """
                INSERT INTO loan (customer_id, principal_amount, interest_rate, loan_term, loan_type,
                loan_status)
                VALUES (%s, %s, %s, %s, %s, %s)
            """
            values = (
                loan.get_customer().get_customer_id(),
                loan.get_principal_amount(),
                loan.get_interest_rate(),
                loan.get_loan_term(),
                loan.get_loan_type(),
```

```

'Pending'

)

cursor.execute(query, values)
conn.commit()

print(" Loan application submitted successfully.")

except Exception as e:

    print(" Error applying loan:", e)

finally:

    cursor.close()

    conn.close()

```

```

dao > interface > loan_repository_impl.py > LoanRepositoryImpl > calculate_interest
  1  import mysql.connector
  2  from dao.interface.i_loan_repository import ILoanRepository
  3  from exception.invalid_loan_exception import InvalidLoanException
  4  from util.db_conn_util import DBConnUtil
  5  from entity.loan import Loan
  6
  7  class LoanRepositoryImpl(ILoanRepository):
  8
  9      def apply_loan(self, loan):
 10          try:
 11              confirm = input("Do you want to apply for the loan? (Yes/No): ").strip().lower()
 12              if confirm != 'yes':
 13                  print("Loan application cancelled.")
 14                  return
 15
 16              conn = DBConnUtil.get_connection()
 17              cursor = conn.cursor()
 18              query = """
 19                  INSERT INTO loan (customer_id, principal_amount, interest_rate, loan_term, loan_type, loan_status)
 20                  VALUES (%s, %s, %s, %s, %s, %s)
 21
 22              values = (
 23                  loan.get_customer().get_customer_id(),
 24                  loan.get_principal_amount(),
 25                  loan.get_interest_rate(),
 26                  loan.get_loan_term(),
 27                  loan.get_loan_type(),
 28                  'Pending'
 29              )
 30              cursor.execute(query, values)
 31              conn.commit()
 32              print("✓ Loan application submitted successfully.")
 33          except Exception as e:
 34              print("✗ Error applying loan:", e)
 35          finally:
 36              cursor.close()
 37              conn.close()
 38

```

b. calculateInterest(loanId): This method should calculate and return the interest amount for the loan. Loan should be retrieved from database and calculate the interest amount if loan not found generate InvalidLoanException.

i. Overload the same method with required parameters to calculate the loan interest amount. It is used to calculate the loan interest while creating loan.

ii. $\text{Interest} = (\text{Principal Amount} * \text{Interest Rate} * \text{Loan Tenure}) / 12$

Code:

```
def calculate_interest(self, loan_id):

    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            principal, rate, term = row
            interest = (principal * rate * term) / 1200
            print(f"Interest for Loan ID {loan_id}: ₹{interest:.2f}")
            return interest
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("X", e)
    finally:
        cursor.close()
        conn.close()

def calculate_interest_manual(self, principal, rate, term):
    interest = (principal * rate * term) / 1200
    print(f"Manual Interest: ₹{interest:.2f}")
    return interest
```

```

def calculate_interest(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            principal, rate, term = row
            interest = (principal * rate * term) / 1200
            print(f"✓ Interest for Loan ID {loan_id}: ₹{interest:.2f}")
            return interest
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("✗", e)
    finally:
        cursor.close()
        conn.close()

    def calculate_interest_manual(self, principal, rate, term):
        interest = (principal * rate * term) / 1200
        print(f"✓ Manual Interest: ₹{interest:.2f}")
        return interest

```

c. loanStatus(loanId): This method should display a message indicating that the loan is approved or rejected based on credit score, if credit score above 650 loan approved else rejected and should update in database.

Code:

```

def loan_status(self, loan_id):

    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT c.credit_score FROM loan l
            JOIN customer c ON l.customer_id = c.customer_id
            WHERE l.loan_id = %s
        """, (loan_id,))

        row = cursor.fetchone()
        if row:
            credit_score = row[0]
            status = "Approved" if credit_score > 650 else "Rejected"

```

```

        cursor.execute("UPDATE loan SET loan_status = %s WHERE loan_id = %s", (status,
loan_id))

        conn.commit()

        print(f" Loan Status for Loan ID {loan_id}: {status}")

    else:

        raise InvalidLoanException("Loan not found for ID: " + str(loan_id))

except InvalidLoanException as e:

    print("X", e)

finally:

    cursor.close()

    conn.close()

```

```

def loan_status(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT c.credit_score FROM loan l
            JOIN customer c ON l.customer_id = c.customer_id
            WHERE l.loan_id = %s
        """, (loan_id,))
        row = cursor.fetchone()
        if row:
            credit_score = row[0]
            status = "Approved" if credit_score > 650 else "Rejected"
            cursor.execute("UPDATE loan SET loan_status = %s WHERE loan_id = %s", (status, loan_id))
            conn.commit()
            print(f"✓ Loan Status for Loan ID {loan_id}: {status}")
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("X", e)
    finally:
        cursor.close()
        conn.close()

```

d. calculateEMI(loanId): This method will calculate the emi amount for a month to repayment. Loan should be retrieved from database and calculate the interest amount, if loan not found generate InvalidLoanException.

i. Overload the same method with required parameters to calculate the loan EMI amount. It is used to calculate the loan EMI while creating loan.

$$\text{ii. EMI} = [P * R * (1+R)^N] / [(1+R)^N - 1]$$

1. EMI: The Equated Monthly Installment.

2. P: Principal Amount (Loan Amount).

3. R: Monthly Interest Rate (Annual Interest Rate / 12 / 100).

4. N: Loan Tenure in months.

Code:

```
def calculate_emi(self, loan_id):

    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            P, R_annual, N = row
            R = R_annual / (12 * 100) # monthly rate
            emi = (P * R * (1 + R) ** N) / ((1 + R) ** N - 1)
            print(f" EMI for Loan ID {loan_id}: ₹{emi:.2f}")
            return emi
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("X", e)
    finally:
        cursor.close()
        conn.close()

def calculate_emi_manual(self, principal, rate, term):
    R = rate / (12 * 100)
    emi = (principal * R * (1 + R) ** term) / ((1 + R) ** term - 1)
    print(f" Manual EMI: ₹{emi:.2f}")
    return emi
```

```

def calculate_emi(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            P, R_annual, N = row
            R = R_annual / (12 * 100) # monthly rate
            emi = (P * R * (1 + R) ** N) / ((1 + R) ** N - 1)
            print(f"✓ EMI for Loan ID {loan_id}: ₹{emi:.2f}")
            return emi
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("✗", e)
    finally:
        cursor.close()
        conn.close()

def calculate_emi_manual(self, principal, rate, term):
    R = rate / (12 * 100)
    emi = (principal * R * (1 + R) ** term) / ((1 + R) ** term - 1)
    print(f"✓ Manual EMI: ₹{emi:.2f}")
    return emi

```

e. loanRepayment(loanId, amount): calculate the noOfEmi can be paid from the amount if the amount is less than single emi reject the payment or pay the emi in whole number and update the variable.

Code:

```

def loan_repayment(self, loan_id, amount):
    try:
        emi = self.calculate_emi(loan_id)
        if emi is None:
            return
        if amount < emi:
            print(" Payment amount is less than single EMI. Payment rejected.")
            return
        num_emis = int(amount // emi)
        print(f" Payment of ₹{amount:.2f} will cover {num_emis} EMI(s).")
    except Exception as e:
        print(" Error in repayment:", e)

```

```

def loan_repayment(self, loan_id, amount):
    try:
        emi = self.calculate_emi(loan_id)
        if emi is None:
            return
        if amount < emi:
            print("✖ Payment amount is less than single EMI. Payment rejected.")
            return
        num_emis = int(amount // emi)
        print(f"✓ Payment of ₹{amount:.2f} will cover {num_emis} EMI(s).")
    except Exception as e:
        print("✖ Error in repayment:", e)

```

f. getAllLoan(): get all loan as list and print the details.

Code:

```

def get_all_loans(self):

    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM loan")
        rows = cursor.fetchall()
        print(" All Loans:")
        for row in rows:
            print(row)
    except Exception as e:
        print(" Error fetching loans:", e)
    finally:
        cursor.close()
        conn.close()

```

```

def get_all_loans(self):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM loan")
        rows = cursor.fetchall()
        print("✓ All Loans:")
        for row in rows:
            print(row)
    except Exception as e:
        print("✗ Error fetching loans:", e)
    finally:
        cursor.close()
        conn.close()

```

g. getLoanById(loanId): get loan and print the details, if loan not found generate InvalidLoanException.

Code:

```

def get_loan_by_id(self, loan_id):

    try:
        conn = DBConnUtil.get_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT * FROM loan WHERE loan_id = %s", (loan_id,))

        row = cursor.fetchone()

        if row:
            print(f" Loan Details for ID {loan_id}:")
            print(row)
            return row
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("X", e)
    finally:
        cursor.close()
        conn.close()

```

```

def get_loan_by_id(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            print(f"✓ Loan Details for ID {loan_id}:")
            print(row)
            return row
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("✗", e)
    finally:
        cursor.close()
        conn.close()

```

6. Define ILoanRepositoryImpl class and implement the ILoanRepository interface and provide implementation of all methods.

Code:

```

import mysql.connector

from dao.interface.i_loan_repository import ILoanRepository

from exception.invalid_loan_exception import InvalidLoanException

from util.db_conn_util import DBConnUtil

from entity.loan import Loan


class LoanRepositoryImpl(ILoanRepository):

    def apply_loan(self, loan):
        try:
            confirm = input("Do you want to apply for the loan? (Yes/No): ").strip().lower()
            if confirm != 'yes':
                print("Loan application cancelled.")
                return
            conn = DBConnUtil.get_connection()
            cursor = conn.cursor()

```

```

query = """
    INSERT INTO loan (customer_id, principal_amount, interest_rate, loan_term, loan_type,
loan_status)
    VALUES (%s, %s, %s, %s, %s, %s)
"""

values = (
    loan.get_customer().get_customer_id(),
    loan.get_principal_amount(),
    loan.get_interest_rate(),
    loan.get_loan_term(),
    loan.get_loan_type(),
    'Pending'
)
cursor.execute(query, values)
conn.commit()
print(" Loan application submitted successfully.")

except Exception as e:
    print("X Error applying loan:", e)
finally:
    cursor.close()
    conn.close()

def calculate_interest(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE
loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            principal, rate, term = row
            interest = (principal * rate * term) / 1200
    
```

```

        print(f" Interest for Loan ID {loan_id}: ₹{interest:.2f}")

    return interest

else:

    raise InvalidLoanException("Loan not found for ID: " + str(loan_id))

except InvalidLoanException as e:

    print("X", e)

finally:

    cursor.close()

    conn.close()

def calculate_interest_manual(self, principal, rate, term):

    interest = (principal * rate * term) / 1200

    print(f" Manual Interest: ₹{interest:.2f}")

    return interest

def loan_status(self, loan_id):

    try:

        conn = DBConnUtil.get_connection()

        cursor = conn.cursor()

        cursor.execute("""

            SELECT c.credit_score FROM loan l

            JOIN customer c ON l.customer_id = c.customer_id

            WHERE l.loan_id = %s

        """ , (loan_id,))

        row = cursor.fetchone()

        if row:

            credit_score = row[0]

            status = "Approved" if credit_score > 650 else "Rejected"

            cursor.execute("UPDATE loan SET loan_status = %s WHERE loan_id = %s", (status, loan_id))

            conn.commit()

```

```

        print(f" Loan Status for Loan ID {loan_id}: {status}")

    else:
        raise InvalidLoanException("Loan not found for ID: " + str(loan_id))

    except InvalidLoanException as e:
        print("X", e)

    finally:
        cursor.close()
        conn.close()

def calculate_emi(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            P, R_annual, N = row
            R = R_annual / (12 * 100) # monthly rate
            emi = (P * R * (1 + R) ** N) / ((1 + R) ** N - 1)
            print(f" EMI for Loan ID {loan_id}: ₹{emi:.2f}")
            return emi
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))

    except InvalidLoanException as e:
        print("X", e)

    finally:
        cursor.close()
        conn.close()

def calculate_emi_manual(self, principal, rate, term):

```

```

R = rate / (12 * 100)

emi = (principal * R * (1 + R) ** term) / ((1 + R) ** term - 1)

print(f" Manual EMI: ₹{emi:.2f}")

return emi


def loan_repayment(self, loan_id, amount):

    try:

        emi = self.calculate_emi(loan_id)

        if emi is None:

            return

        if amount < emi:

            print("Payment amount is less than single EMI. Payment rejected.")

            return

        num_emis = int(amount // emi)

        print(f" Payment of ₹{amount:.2f} will cover {num_emis} EMI(s).")

    except Exception as e:

        print("X Error in repayment:", e)


def get_all_loans(self):

    try:

        conn = DBConnUtil.get_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT * FROM loan")

        rows = cursor.fetchall()

        print(" All Loans:")

        for row in rows:

            print(row)

    except Exception as e:

        print("X Error fetching loans:", e)

    finally:

        cursor.close()

```

```
conn.close()

def get_loan_by_id(self, loan_id):
    try:
        conn = DBConnUtil.get_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM loan WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()
        if row:
            print(f" Loan Details for ID {loan_id}:")
            print(row)
            return row
        else:
            raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
    except InvalidLoanException as e:
        print("X", e)
    finally:
        cursor.close()
        conn.close()
```

```

dao > impl > loan_repository_impl.py > LoanRepositoryImpl > apply_loan
1 import mysql.connector
2 from dao.interface.i_loan_repository import ILoanRepository
3 from exception.invalid_loan_exception import InvalidLoanException
4 from util.db_conn_util import DBConnUtil
5 from entity.loan import Loan
6
7 class LoanRepositoryImpl(ILoanRepository):
8
9     def apply_loan(self, loan):
10         try:
11             confirm = input("Do you want to apply for the loan? (Yes/No): ").strip().lower()
12             if confirm != 'yes':
13                 print("Loan application cancelled.")
14                 return
15
16             conn = DBConnUtil.get_connection()
17             cursor = conn.cursor()
18             query = """
19                 INSERT INTO loan (customer_id, principal_amount, interest_rate, loan_term, loan_type, loan_status)
20                 VALUES (%s, %s, %s, %s, %s, %s)
21             """
22             values = (
23                 loan.get_customer().get_customer_id(),
24                 loan.get_principal_amount(),
25                 loan.get_interest_rate(),
26                 loan.get_loan_term(),
27                 loan.get_loan_type(),
28                 'Pending'
29             )
30             cursor.execute(query, values)
31             conn.commit()
32             print("✓ Loan application submitted successfully.")
33         except Exception as e:
34             print("✗ Error applying loan:", e)
35         finally:
36             cursor.close()
37             conn.close()

```

```

dao > impl > loan_repository_impl.py > LoanRepositoryImpl > calculate_interest
7 class LoanRepositoryImpl(ILoanRepository):
39     def calculate_interest(self, loan_id):
40         try:
41             conn = DBConnUtil.get_connection()
42             cursor = conn.cursor()
43             cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
44             row = cursor.fetchone()
45             if row:
46                 principal, rate, term = row
47                 interest = (principal * rate * term) / 1200
48                 print(f"✓ Interest for Loan ID {loan_id}: ₹{interest:.2f}")
49                 return interest
50             else:
51                 raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
52         except InvalidLoanException as e:
53             print("✗", e)
54         finally:
55             cursor.close()
56             conn.close()
57
58     def calculate_interest_manual(self, principal, rate, term):
59         interest = (principal * rate * term) / 1200
60         print(f"✓ Manual Interest: ₹{interest:.2f}")
61         return interest

```

```
dao>impl>loan_repository_impl.py>LoanRepositoryImpl>calculate_interest
7     class LoanRepositoryImpl(ILoanRepository):
39         def calculate_interest(self, loan_id):
62             def loan_status(self, loan_id):
63                 try:
64                     conn = DBConnUtil.get_connection()
65                     cursor = conn.cursor()
66                     cursor.execute("""
67                         SELECT c.credit_score FROM loan l
68                         JOIN customer c ON l.customer_id = c.customer_id
69                         WHERE l.loan_id = %s
70                         """, (loan_id,))
71                     row = cursor.fetchone()
72                     if row:
73                         credit_score = row[0]
74                         status = "Approved" if credit_score > 650 else "Rejected"
75                         cursor.execute("UPDATE loan SET loan_status = %s WHERE loan_id = %s", (status, loan_id))
76                         conn.commit()
77                         print(f"✓ Loan Status for Loan ID {loan_id}: {status}")
78                     else:
79                         raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
80                 except InvalidLoanException as e:
81                     print("✗", e)
82             finally:
83                 cursor.close()
84                 conn.close()
85
86         def calculate_emi(self, loan_id):
87             try:
88                 conn = DBConnUtil.get_connection()
89                 cursor = conn.cursor()
90                 cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM loan WHERE loan_id = %s", (loan_id,))
91                 row = cursor.fetchone()
92                 if row:
93                     P, R_annual, N = row
94                     R = R_annual / (12 * 100) # monthly rate
95                     emi = (P * R * (1 + R) ** N) / ((1 + R) ** N - 1)
96                     print(f"✓ EMI for Loan ID {loan_id}: ₹{emi:.2f}")
97                     return emi
98                 else:
99                     raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
100            except InvalidLoanException as e:
101                print("✗", e)
102            finally:
103                cursor.close()
104                conn.close()
```

```
dao > impl > loan_repository.impl.py > LoanRepositoryImpl > calculate_interest > loan_status
  7   class LoanRepositoryImpl(ILoanRepository):
  8     def calculate_interest(self, loan_id):
  96
 107       def calculate_emi_manual(self, principal, rate, term):
 108         R = rate / (12 * 100)
 109         emi = (principal * R * (1 + R) ** term) / ((1 + R) ** term - 1)
 110         print(f"✓ Manual EMI: {emi:.2f}")
 111         return emi
 112
 113       def loan_repayment(self, loan_id, amount):
 114         try:
 115           emi = self.calculate_emi(loan_id)
 116           if emi is None:
 117             return
 118           if amount < emi:
 119             print("✗ Payment amount is less than single EMI. Payment rejected.")
 120             return
 121           num_emis = int(amount // emi)
 122           print(f"✓ Payment of {amount:.2f} will cover {num_emis} EMI(s).")
 123         except Exception as e:
 124           print("✗ Error in repayment:", e)
 125
 126       def get_all_loans(self):
 127         try:
 128           conn = DBConnUtil.get_connection()
 129           cursor = conn.cursor()
 130           cursor.execute("SELECT * FROM loan")
 131           rows = cursor.fetchall()
 132           print("✓ All Loans:")
 133           for row in rows:
 134             print(row)
 135         except Exception as e:
 136           print("✗ Error fetching loans:", e)
 137         finally:
 138           cursor.close()
 139           conn.close()
 140
 141       def get_loan_by_id(self, loan_id):
 142         try:
 143           conn = DBConnUtil.get_connection()
 144           cursor = conn.cursor()
 145           cursor.execute("SELECT * FROM loan WHERE loan_id = %s", (loan_id,))
 146           row = cursor.fetchone()
 147           if row:
 148             print(f"✓ Loan Details for ID {loan_id}:")
 149             print(row)
 150             return row
 151           else:
 152             raise InvalidLoanException("Loan not found for ID: " + str(loan_id))
 153         except InvalidLoanException as e:
 154           print("✗", e)
 155         finally:
 156           cursor.close()
 157           conn.close()
 158
```

7. Create DBUtil class and add the following method.

a. static getDBConn():Connection Establish a connection to the database and return Connection reference

Code:

```
import mysql.connector  
from util.db_property_util import DBPropertyUtil  
  
class DBUtil:  
    @staticmethod  
    def getDBConn():  
        conn_details = DBPropertyUtil.get_connection_string('db.properties.ini')  
        connection = mysql.connector.connect(**conn_details)  
        return connection
```

```
util > db_util.py > ...  
1  import mysql.connector  
2  from util.db_property_util import DBPropertyUtil  
3  
4  class DBUtil:  
5      @staticmethod  
6      def getDBConn():  
7          conn_details = DBPropertyUtil.get_connection_string('db.properties.ini')  
8          connection = mysql.connector.connect(**conn_details)  
9          return connection  
10 |
```

8. Create LoanManagement main class and perform following operation:

a. main method to simulate the loan management system. Allow the user to interact with the system by entering choice from menu such as "applyLoan", "getAllLoan", "getLoan", "loanRepayment", "exit."

Code:

```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from dao.impl.loan_repository_impl import LoanRepositoryImpl
from entity.loan import Loan
from entity.customer import Customer

def show_menu():
    print("\n===== Loan Management System =====")
    print("1. Apply for a Loan")
    print("2. View All Loans")
    print("3. View Loan by ID")
    print("4. Calculate Interest")
    print("5. Calculate EMI")
    print("6. Make Loan Repayment")
    print("7. Update Loan Status")
    print("0. Exit")
    return input("Enter your choice: ")

def main():
    repo = LoanRepositoryImpl()

    while True:
        choice = show_menu()
```

```
if choice == '1':  
    customer_id = input("Enter Customer ID: ")  
    principal = float(input("Enter Principal Amount: "))  
    rate = float(input("Enter Interest Rate: "))  
    term = int(input("Enter Loan Term (in months): "))  
    loan_type = input("Enter Loan Type (HomeLoan/CarLoan): ")  
  
    # Basic customer object just for relation  
    customer = Customer(customer_id=customer_id, name="", email="", phone="", address="",  
    credit_score=700)  
    loan = Loan(loan_id=None, customer=customer, principal_amount=principal,  
    interest_rate=rate, loan_term=term, loan_type=loan_type, loan_status="Pending")  
  
    repo.apply_loan(loan)  
  
elif choice == '2':  
    repo.get_all_loans()  
  
elif choice == '3':  
    loan_id = input("Enter Loan ID: ")  
    repo.get_loan_by_id(loan_id)  
  
elif choice == '4':  
    loan_id = input("Enter Loan ID: ")  
    repo.calculate_interest(loan_id)  
  
elif choice == '5':  
    loan_id = input("Enter Loan ID: ")  
    repo.calculate_emi(loan_id)  
  
elif choice == '6':
```

```
loan_id = input("Enter Loan ID: ")

amount = float(input("Enter Repayment Amount: "))

repo.loan_repayment(loan_id, amount)

elif choice == '7':

    loan_id = input("Enter Loan ID to update status: ")

    repo.loan_status(loan_id)

elif choice == '0':

    print(" Exiting Loan Management System.")

    break

else:

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

if __name__ == "__main__":

    main()
```

```
main.py - Loan Management System - m
1 import sys
2 import os
3 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
4
5 from dao.impl.loan_repository_impl import LoanRepositoryImpl
6 from entity.loan import Loan
7 from entity.customer import Customer
8
9 def show_menu():
10     print("\n==== Loan Management System ====")
11     print("1. Apply for a Loan")
12     print("2. View All Loans")
13     print("3. View Loan by ID")
14     print("4. Calculate Interest")
15     print("5. Calculate EMI")
16     print("6. Make Loan Repayment")
17     print("7. Update Loan Status")
18     print("0. Exit")
19     return input("Enter your choice: ")
20
21 def main():
22     repo = LoanRepositoryImpl()
23
24     while True:
25         choice = show_menu()
26
27         if choice == '1':
28             customer_id = input("Enter Customer ID: ")
29             principal = float(input("Enter Principal Amount: "))
30             rate = float(input("Enter Interest Rate: "))
31             term = int(input("Enter Loan Term (in months): "))
32             loan_type = input("Enter Loan Type (HomeLoan/CarLoan): ")
33
34             # Basic customer object just for relation
35             customer = Customer(customer_id=customer_id, name="", email="", phone="", address="", credit_score=700)
36             loan = Loan(loan_id=None, customer=customer, principal_amount=principal,
37                         interest_rate=rate, loan_term=term, loan_type=loan_type, loan_status="Pending")
38
39             repo.apply_loan(loan)
40
41         elif choice == '2':
42             repo.get_all_loans()
43
44         elif choice == '3':
45             loan_id = input("Enter Loan ID: ")
46             repo.get_loan_by_id(loan_id)
```

```
47     elif choice == '4':
48         loan_id = input("Enter Loan ID: ")
49         repo.calculate_interest(loan_id)
50
51     elif choice == '5':
52         loan_id = input("Enter Loan ID: ")
53         repo.calculate_emi(loan_id)
54
55
56     elif choice == '6':
57         loan_id = input("Enter Loan ID: ")
58         amount = float(input("Enter Repayment Amount: "))
59         repo.loan_repayment(loan_id, amount)
60
61     elif choice == '7':
62         loan_id = input("Enter Loan ID to update status: ")
63         repo.loan_status(loan_id)
64
65     elif choice == '0':
66         print("👋 Exiting Loan Management System.")
67         break
68
69     else:
70         print("🔴 Invalid choice. Please try again.")
71
72 if __name__ == "__main__":
73     main()
74
```

OUTPUT:**1.APPLY FOR A LOAN**

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 1
Enter Customer ID: 4
Enter Principal Amount: 300000
Enter Interest Rate: 8.5
Enter Loan Term (in months): 30
Enter Loan Type (HomeLoan/CarLoan): CarLoan
Do you want to apply for the loan? (Yes/No): Yes
 Loan application submitted successfully.

===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 1
Enter Customer ID: 5
Enter Principal Amount: 9000000
Enter Interest Rate: 9.3
Enter Loan Term (in months): 36
Enter Loan Type (HomeLoan/CarLoan): HomeLoan
Do you want to apply for the loan? (Yes/No): Yes
 Loan application submitted successfully.
```

2. VIEW ALL LOANS

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 2
 All Loans:
(104, 2, Decimal('120000.00'), Decimal('9.50'), 24, 'CarLoan', 'Pending')
(105, 2, Decimal('20000.00'), Decimal('9.20'), 30, 'HomeLoan', 'Pending')
(106, 3, Decimal('300000.00'), Decimal('7.20'), 20, 'CarLoan', 'Pending')
(107, 4, Decimal('300000.00'), Decimal('8.50'), 30, 'CarLoan', 'Pending')
(108, 5, Decimal('9000000.00'), Decimal('9.30'), 36, 'HomeLoan', 'Pending')
```

3. VIEW LOAN BY ID

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 3
Enter Loan ID: 104
 Loan Details for ID 104:
(104, 2, Decimal('120000.00'), Decimal('9.50'), 24, 'CarLoan', 'Pending')
```

4. CALCULATE INTEREST

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 4
Enter Loan ID: 104
 Interest for Loan ID 104: ₹22800.00
```

5. CALCULATE EMI

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 5
Enter Loan ID: 104
 EMI for Loan ID 104: ₹5509.74
```

6. LOAN REPAYMENT

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice:
Enter your choice: 6
Enter Repayment Amount: 12000
 EMI for Loan ID 104: ₹5509.74
 Payment of ₹12000.00 will cover 2 EMI(s).
```

7.UPDATE LOAN STATUS

```
===== Loan Management System =====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice:
Enter your choice: 7
Enter Loan ID to update status: 104
 Loan Status for Loan ID 104: Rejected
```