Coding Challenge - Loan Management System

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

```
mysql> desc customer;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| customerId  | int          | NO   | PRI | NULL    |       |
| name        | varchar(100) | YES  |     | NULL    |       |
| email       | varchar(100) | YES  |     | NULL    |       |
| phoneNumber | varchar(15)  | YES  |     | NULL    |       |
| address     | varchar(255) | YES  |     | NULL    |       |
| creditScore | int          | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

```python
class Customer:
    def __init__(self, customerId=0,
                 name='',
                 emailAddress='',
                 phoneNumber='',
                 address='',
                 creditScore=0):
        self.customerId = customerId
        self.name = name
        self.emailAddress = emailAddress
        self.phoneNumber = phoneNumber
        self.address = address
        self.creditScore = creditScore

    def __str__(self):
        return (f"Customer[ID: {self.customerId}, Name: {self.name}, "
                f"Email: {self.emailAddress}, Phone: {self.phoneNumber}, "
                f"Address: {self.address}, Credit Score: {self.creditScore}]")
```

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)

```
mysql> desc loan;
+-----------------+---------------+------+-----+---------+-------+
| Field           | Type          | Null | Key | Default | Extra |
+-----------------+---------------+------+-----+---------+-------+
| loanId          | int           | NO   | PRI | NULL    |       |
| customerId      | int           | YES  | MUL | NULL    |       |
| principalAmount | decimal(10,2) | YES  |     | NULL    |       |
| interestRate    | decimal(5,2)  | YES  |     | NULL    |       |
| loanTerm        | int           | YES  |     | NULL    |       |
| loanType        | varchar(20)   | YES  |     | NULL    |       |
| loanStatus      | varchar(20)   | YES  |     | NULL    |       |
+-----------------+---------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

```python
class Loan:
    def __init__(self, loanId=0, customer=None, principalAmount=0, interestRate=0, loanTerm=0, loanType='', loanStatus='Pending'):
        self.loanId = loanId
        self.customer = customer
        self.principalAmount = principalAmount
        self.interestRate = interestRate
        self.loanTerm = loanTerm
        self.loanType = loanType
        self.loanStatus = loanStatus

    def __str__(self):
        return (f"Loan[ID: {self.loanId}, "
                f"Customer: {self.customer}, "
                f"Principal: {self.principalAmount},"
                f" Interest: {self.interestRate}, "
                f"Term: {self.loanTerm} months, "
                f"Type: {self.loanType}, "
                f"Status: {self.loanStatus}]")
```

excep ×

```python
class HomeLoan(Loan):
    def __init__(self, loanId=0,
                 customer=None,
                 principalAmount=0,
                 interestRate=0,
                 loanTerm=0, loanType='HomeLoan',
                 loanStatus='Pending',
                 propertyAddress='',
                 propertyValue=0):
        super().__init__(loanId, customer, principalAmount, interestRate, loanTerm, loanType, loanStatus)
        self.propertyAddress = propertyAddress
        self.propertyValue = propertyValue

    def __str__(self):
        return (f"{super().__str__()}, "
                f"Property Address: {self.propertyAddress}, Property Value: {self.propertyValue}")
```

```python
class CarLoan(Loan):
    def __init__(self, loanId=0,
                 customer=None,
                 principalAmount=0,
                 interestRate=0,
                 loanTerm=0,
                 loanType='CarLoan',
                 loanStatus='Pending',
                 carModel='', carValue=0):
        super().__init__(loanId, customer, principalAmount, interestRate, loanTerm, loanType, loanStatus)
        self.carModel = carModel
        self.carValue = carValue

    def __str__(self):
        return (f"{super().__str__()}, "
                f"Car Model: {self.carModel}, "
                f"Car Value: {self.carValue}")
```

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.

```python
class Customer:
    def __init__(self, customer_id=0, name='', email='', phone_number='', address='', credit_score=0):
        self.customer_id = customer_id
        self.name = name
        self.email = email
        self.phone_number = phone_number
        self.address = address
        self.credit_score = credit_score
```

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

```python
class Customer:
    def __init__(self, customer_id, name, email, phone_number, address, credit_score):
        self.customer_id = customer_id
        self.name = name
        self.email = email
        self.phone_number = phone_number
        self.address = address
        self.credit_score = credit_score

    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
```

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.

```python
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 loan_status(self, loan_id):
        pass

    @abstractmethod
    def calculate_emi(self, loan_id):
        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
```

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

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

```python
import mysql.connector


5 usages
class InvalidLoanException(Exception):
    pass


class ILoanRepositoryImpl(ILoanRepository):
    def __init__(self):
        self.conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="LoanManagementDB"
        )
        self.cursor = self.conn.cursor()

    def apply_loan(self, loan):
        confirm = input("Do you want to apply for the loan? (Yes/No): ").strip().lower()
        if confirm == "yes":
            query = ("INSERT INTO Loan (loanId, customerId, principalAmount, interestRate, "
                     "loanTerm, loanType, loanStatus) VALUES (%s, %s, %s, %s, %s, %s, %s)")
            values = (
            loan.loan_id, loan.customer.customer_id, loan.principal_amount, loan.interest_rate, loan.loan_term,
            loan.loan_type, loan.loan_status)
            self.cursor.execute(query, values)
            self.conn.commit()
            print("Loan applied successfully.")

    def calculate_interest(self, loan_id):
        query = "SELECT principalAmount, interestRate, loanTerm FROM Loan WHERE loanId = %s"
```

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. Interest = (Principal Amount * Interest Rate * Loan Tenure) / 12

```
def calculate_interest(self, loan_id):
    query = "SELECT principalAmount, interestRate, loanTerm FROM Loan WHERE loanId = %s
    self.cursor.execute(query, params: (loan_id,))
    loan_data = self.cursor.fetchone()

    if not loan_data:
        raise InvalidLoanException("Loan not found")

    principal_amount, interest_rate, loan_term = loan_data
    interest = (principal_amount * interest_rate * loan_term) / 12
    return interest

1 usage (1 dynamic)
def loan_status(self, loan_id):
    query = "SELECT customerId FROM Loan WHERE loanId = %s"
    self.cursor.execute(query, params: (loan_id,))
    result = self.cursor.fetchone()

    if not result:
        raise InvalidLoanException("Loan not found")

    customer_id = result[0]
    query = "SELECT creditScore FROM Customer WHERE customerId = %s"
    self.cursor.execute(query, params: (customer_id,))
    customer_data = self.cursor.fetchone()
```

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.

```
class ILoanRepositoryImpl(ILoanRepository):
    def loan_status(self, loan_id):
        if not customer_data:
            raise InvalidLoanException("Customer not found")

        credit_score = customer_data[0]
        status = "Approved" if credit_score > 650 else "Rejected"

        query = "UPDATE Loan SET loanStatus = %s WHERE loanId = %s"
        self.cursor.execute(query, params: (status, loan_id))
        self.conn.commit()

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

    1 usage
    def calculate_emi(self, loan_id):
        query = "SELECT principalAmount, interestRate, loanTerm FROM Loan WHERE loanId = %s"
        self.cursor.execute(query, params: (loan_id,))
        loan_data = self.cursor.fetchone()

        if not loan_data:
            raise InvalidLoanException("Loan not found")

        principal_amount, interest_rate, loan_term = loan_data
        monthly_interest_rate = (interest_rate / 12) / 100
        emi = (principal_amount * monthly_interest_rate * (1 + monthly_interest_rate) ** loan_term) / (
                (1 + monthly_interest_rate) ** loan_term - 1)
        return emi

    def loan_repayment(self, loan_id, amount):
        emi = self.calculate_emi(loan_id)
        if amount < emi:
```

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.

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.

```
1 usage
def calculate_emi(self, loan_id):
    query = "SELECT principalAmount, interestRate, loanTerm FROM Loan WHERE loanId = %s"
    self.cursor.execute(query, params: (loan_id,))
    loan_data = self.cursor.fetchone()

    if not loan_data:
        raise InvalidLoanException("Loan not found")

    principal_amount, interest_rate, loan_term = loan_data
    monthly_interest_rate = (interest_rate / 12) / 100
    emi = (principal_amount * monthly_interest_rate * (1 + monthly_interest_rate) ** loan_term) / (
            (1 + monthly_interest_rate) ** loan_term - 1)
    return emi

def loan_repayment(self, loan_id, amount):
    emi = self.calculate_emi(loan_id)
    if amount < emi:
        print("Insufficient amount to repay a single EMI.")
        return

    no_of_emi = amount // emi
    if no_of_emi < 1:
        print("Insufficient amount for even a single EMI.")
        return

    print(f"Repaying {no_of_emi} EMIs for loanId {loan_id}.")
```

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.

```
1 usage
def calculate_emi(self, loan_id):
    query = "SELECT principalAmount, interestRate, loanTerm FROM Loan WHERE loanId = %s"
    self.cursor.execute(query, params: (loan_id,))
    loan_data = self.cursor.fetchone()

    if not loan_data:
        raise InvalidLoanException("Loan not found")

    principal_amount, interest_rate, loan_term = loan_data
    monthly_interest_rate = (interest_rate / 12) / 100
    emi = (principal_amount * monthly_interest_rate * (1 + monthly_interest_rate) ** loan_term) / (
            (1 + monthly_interest_rate) ** loan_term - 1)
    return emi

def loan_repayment(self, loan_id, amount):
    emi = self.calculate_emi(loan_id)
    if amount < emi:
        print("Insufficient amount to repay a single EMI.")
        return

    no_of_emi = amount // emi
    if no_of_emi < 1:
        print("Insufficient amount for even a single EMI.")
        return

    print(f"Repaying {no_of_emi} EMIs for loanId {loan_id}.")
```

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

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

```
def get_all_loans(self):
    query = "SELECT * FROM Loan"
    self.cursor.execute(query)
    loans = self.cursor.fetchall()

    for loan in loans:
        print(loan)

def get_loan_by_id(self, loan_id):
    query = "SELECT * FROM Loan WHERE loanId = %s"
    self.cursor.execute(query, params: (loan_id,))
    loan = self.cursor.fetchone()

    if not loan:
        raise InvalidLoanException("Loan not found")

    print(loan)
```

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

```
class DBUtil:
    @staticmethod
    def get_db_conn():
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            database="LoanManagementDB"
        )
        return conn
```

7. Create DBUtil class and add the following method.

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

```
class LoanManagement:
    def __init__(self):
        self.loan_repository = ILoanRepositoryImpl()

    1 usage
    def main_menu(self):
        while True:
            print("1. Apply Loan")
            print("2. Get All Loans")
            print("3. Get Loan by ID")
            print("4. Calculate EMI")
            print("5. Calculate Interest")
            print("6. Loan Repayment")
            print("7. Loan Status")
            print("8. Exit")

            choice = input("Enter your choice: ")
            if choice == "1":
                self.apply_loan()
            elif choice == "2":
                self.get_all_loans()
            elif choice == "3":
                self.get_loan_by_id()
            elif choice == "4":
                self.calculate_emi()
            elif choice == "5":
                self.calculate_interest()
            elif choice == "6":
                self.loan_repayment()
            elif choice == "7":
                self.loan_status()
```

```
            self.loan_repayment()
        elif choice == "7":
            self.loan_status()
        elif choice == "8":
            break
        else:
            print("Invalid choice, try again.")

2 usages (1 dynamic)
def apply_loan(self):
    loan_id = int(input("Enter loanId: "))
    customer_id = int(input("Enter customerId: "))
    principal_amount = float(input("Enter principalAmount: "))
    interest_rate = float(input("Enter interestRate: "))
    loan_term = int(input("Enter loanTerm (in months): "))
    loan_type = input("Enter loanType (CarLoan/HomeLoan): ")

    customer = Customer(customer_id, "Sample Name", "sample@example.com", "1234567890", "Sample Address", 720)

    if loan_type == "CarLoan":
        car_model = input("Enter carModel: ")
        car_value = int(input("Enter carValue: "))
        loan = CarLoan(loan_id, customer, principal_amount, interest_rate, loan_term, loan_type, "Pending",
                       car_model, car_value)
    elif loan_type == "HomeLoan":
        property_address = input("Enter propertyAddress: ")
        property_value = int(input("Enter propertyValue: "))
        loan = HomeLoan(loan_id, customer, principal_amount, interest_rate, loan_term, loan_type, "Pending",
                        property_address, property_value)
    else:
```

```
                        property_address, property_value)
        else:
            print("Invalid loan type.")
            return

        self.loan_repository.apply_loan(loan)

    2 usages (1 dynamic)
    def get_all_loans(self):
        self.loan_repository.get_all_loans()

    2 usages (1 dynamic)
    def get_loan_by_id(self):
        loan_id = int(input("Enter loanId: "))
        try:
            self.loan_repository.get_loan_by_id(loan_id)
        except InvalidLoanException as e:
            print(e)

    2 usages (1 dynamic)
    def calculate_emi(self):
        loan_id = int(input("Enter loanId: "))
        try:
            emi = self.loan_repository.calculate_emi(loan_id)
            print(f"EMI for loanId {loan_id}: {emi:.2f}")
        except InvalidLoanException as e:
            print(e)
```

```
    2 usages (1 dynamic)
    def calculate_interest(self):
        loan_id = int(input("Enter loanId: "))
        try:
            interest = self.loan_repository.calculate_interest(loan_id)
            print(f"Interest for loanId {loan_id}: {interest:.2f}")
        except InvalidLoanException as e:
            print(e)

    2 usages (1 dynamic)
    def loan_repayment(self):
        loan_id = int(input("Enter loanId: "))
        amount = float(input("Enter repayment amount: "))
        try:
            self.loan_repository.loan_repayment(loan_id, amount)
        except InvalidLoanException as e:
            print(e)

    2 usages (1 dynamic)
    def loan_status(self):
        loan_id = int(input("Enter loanId: "))
        try:
            self.loan_repository.loan_status(loan_id)
        except InvalidLoanException as e:
            print(e)


if __name__ == "__main__":
    LoanManagement().main_menu()
```

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

Output:

1. Apply Loan

2. Get All Loans

3. Get Loan by ID

4. Calculate EMI

5. Calculate Interest

6. Loan Repayment

7. Loan Status

8. Exit

Enter your choice: 1

Enter loanId: 1001

Enter customerId: 1

Enter principalAmount: 50000

Enter interestRate: 5

Enter loanTerm (in months): 60

Enter loanType (CarLoan/HomeLoan): HomeLoan

Enter propertyAddress: 123 Home St

Enter propertyValue: 100000

Do you want to apply for the loan? (Yes/No): Yes

Loan applied successfully.

Enter your choice: 2

(1001, 1, 50000, 5, 60, 'HomeLoan', 'Pending')

Enter your choice: 3

Enter loanId: 1001

(1001, 1, 50000, 5, 60, 'HomeLoan', 'Pending')

Enter your choice: 4

Enter loanId: 1001

EMI for loanId 1001: 943.56

Enter your choice: 5

Enter loanId: 1001

Interest for loanId 1001: 20833.33

Enter your choice: 6

Enter loanId: 1001

Enter repayment amount: 2000

Repaying 2 EMIs for loanId 1001.

Enter your choice: 7

Enter loanId: 1001

Loan status for loanId 1001: Approved

Enter your choice: 8

```
1. Apply Loan
2. Get All Loans
3. Get Loan by ID
4. Calculate EMI
5. Calculate Interest
6. Loan Repayment
7. Loan Status
8. Exit

Enter your choice: 1
Enter loanId: 1001
Enter customerId: 1
Enter principalAmount: 50000
Enter interestRate: 5
Enter loanTerm (in months): 60
Enter loanType (CarLoan/HomeLoan): HomeLoan
Enter propertyAddress: 123 Home St
Enter propertyValue: 100000
Do you want to apply for the loan? (Yes/No): Yes
Loan applied successfully.
```

```
Enter your choice: 6
Enter loanId: 1001
Enter repayment amount: 2000
Repaying 2 EMIs for loanId 1001.
```

```
Enter your choice: 8
```

Choice 8 is exit. So it exit from the code.