Loan Management System

Name : Jegadeeswaran

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

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

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)

```python
class Loan:
    def __init__(self, loan_id, 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
```

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.

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

```python
class HomeLoan(Loan):
    def __init__(self, loan_id, customer, principal_amount, interest_rate, loan_term,
                 loan_status, property_address, property_value):
        super().__init__(loan_id, customer, principal_amount, interest_rate, loan_term, 'HomeLoan', loan_status)
        self.property_address = property_address
        self.property_value = property_value
```

```python
class CarLoan(Loan):
    def __init__(self, loan_id, customer, principal_amount, interest_rate,
                 loan_term, loan_status, car_model, car_value):
        super().__init__(loan_id, customer, principal_amount, interest_rate, loan_term, 'CarLoan', loan_status)
        self.car_model = car_model
        self.car_value = car_value
```

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
class Customer:
    def __init__(self):
        self.customerId = 0
        self.name = ""
        self.emailAddress = ""
        self.phoneNumber = ""
        self.address = ""
        self.creditScore = 0

    def __init__(self, customerId, name, emailAddress, phoneNumber, address, creditScore):
        self.customerId = customerId
        self.name = name
        self.emailAddress = emailAddress
        self.phoneNumber = phoneNumber
        self.address = address
        self.creditScore = creditScore
```

```python
# Getter and Setter

def getCustomerId(self):
    return self.customerId

def setCustomerId(self, customerId):
    self.customerId = customerId

def getName(self):
    return self.name

def setName(self, name):
    self.name = name
```

```python
def getEmailAddress(self):
    return self.emailAddress

def setEmailAddress(self, emailAddress):
    self.emailAddress = emailAddress

def getPhoneNumber(self):
    return self.phoneNumber

def setPhoneNumber(self, phoneNumber):
    self.phoneNumber = phoneNumber
```

```
def getAddress(self):
    return self.address

def setAddress(self, address):
    self.address = address

def getCreditScore(self):
    return self.creditScore

def setCreditScore(self, creditScore):
    self.creditScore = creditScore
```

```
    def printCustomerInfo(self):
        print(f"Customer ID: {self.customerId}")
        print(f"Name: {self.name}")
        print(f"Email Address: {self.emailAddress}")
        print(f"Phone Number: {self.phoneNumber}")
        print(f"Address: {self.address}")
        print(f"Credit Score: {self.creditScore}")

class Loan:
    def __init__(self):
        self.loanId = 0
        self.customer = Customer()
        self.principalAmount = 0
        self.interestRate = 0
        self.loanTerm = 0
        self.loanType = ""
        self.loanStatus = ""
```

```
def __init__(self, loanId, customer, principalAmount, interestRate, loanTerm, loanType, loanStatus):
    self.loanId = loanId
    self.customer = customer
    self.principalAmount = principalAmount
    self.interestRate = interestRate
    self.loanTerm = loanTerm
    self.loanType = loanType
    self.loanStatus = loanStatus
```

```
# Getter and Setter

def getLoanId(self):
    return self.loanId

def setLoanId(self, loanId):
    self.loanId = loanId

def getCustomer(self):
    return self.customer

def setCustomer(self, customer):
    self.customer = customer

def getPrincipalAmount(self):
    return self.principalAmount

def setPrincipalAmount(self, principalAmount):
    self.principalAmount = principalAmount

def getInterestRate(self):
    return self.interestRate

def setInterestRate(self, interestRate):
    self.interestRate = interestRate
```

```
def getLoanTerm(self):
    return self.loanTerm

def setLoanTerm(self, loanTerm):
    self.loanTerm = loanTerm

def getLoanType(self):
    return self.loanType

def setLoanType(self, loanType):
    self.loanType = loanType
```

```
def getLoanStatus(self):
    return self.loanStatus

def setLoanStatus(self, loanStatus):
    self.loanStatus = loanStatus
```

```
def printLoanInfo(self):
    print(f"Loan ID: {self.loanId}")
    print("Customer Information:")
    self.customer.printCustomerInfo()
    print(f"Principal Amount: {self.principalAmount}")
    print(f"Interest Rate: {self.interestRate}")
    print(f"Loan Term: {self.loanTerm}")
    print(f"Loan Type: {self.loanType}")
    print(f"Loan Status: {self.loanStatus}")
```

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

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

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.

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.

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.

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.

```python
from abc import ABC, abstractmethod
from entity.loan import Loan

class ILoanRepository(ABC):

    @abstractmethod
    def applyLoan(self, loan: Loan):
        pass

    @abstractmethod
    def calculateInterest(self, loanId: int):
        pass

    @abstractmethod
    def loanStatus(self, loanId: int):
        pass

    @abstractmethod
    def calculateEMI(self, loanId: int):
        pass

    @abstractmethod
    def loanRepayment(self, loanId: int, amount: float):
        pass

    @abstractmethod
    def getAllLoan(self):
        pass

    @abstractmethod
    def getLoanById(self, loanId: int):
        pass
```

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

```python
from typing import List
from step5 import ILoanRepository
from entity import Loan, Customer
from exception.exceptions import InvalidLoanException
from util.DBUtil import DBUtil

class ILoanRepositoryImpl(ILoanRepository):
    def __init__(self, db_name):
        self.db_name = db_name
        self.db_conn = DBUtil.getDBCon(db_name)

    def applyLoan(self, loan: Loan):
        cursor = self.db_conn.cursor()
        try:
            cursor.execute("SELECT * FROM loan WHERE loanId=?", (loan.loanId,))
            existing_loan = cursor.fetchone()
            if existing_loan:
                raise InvalidLoanException(f"Loan with ID {loan.loanId} already exists.")

            cursor.execute("INSERT INTO loan VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",
                           (loan.loanId, loan.customer.customerId, loan.principalAmount,
                            loan.interestRate, loan.loanTerm, loan.loanType, loan.loanStatus,
                            loan.propertyAddress, loan.propertyValue))

            self.db_conn.commit()
            print("Loan application successful.")
        except Exception as e:
            print(f"Error applying loan: {str(e)}")
        finally:
            cursor.close()
```

```python
def calculateInterest(self, loanId: int):
    cursor = self.db_conn.cursor()
    try:
        cursor.execute("SELECT * FROM loan WHERE loanId=?", (loanId,))
        loan_data = cursor.fetchone()
        if not loan_data:
            raise InvalidLoanException(f"Loan with ID {loanId} not found.")

        principal_amount = loan_data[2]
        interest_rate = loan_data[3]
        loan_tenure = loan_data[4]
        interest = (principal_amount * interest_rate * loan_tenure) / 12
        print(f"Interest calculated for Loan {loanId}: {interest}")
        return interest
    except Exception as e:
        print(f"Error calculating interest: {str(e)}")
    finally:
        cursor.close()
```

```python
def loanStatus(self, loanId: int):
    cursor = self.db_conn.cursor()
    try:
        cursor.execute("SELECT creditScore FROM customer WHERE customerId=(SELECT customerId FROM loans WHERE loanId=?)", (loanId,))
        credit_score = cursor.fetchone()[0]
        if credit_score > 650:
            loan_status = "Approved"
        else:
            loan_status = "Rejected"

        cursor.execute("UPDATE loan SET loanStatus=? WHERE loanId=?", (loan_status, loanId,))
        self.db_conn.commit()
        print(f"Loan {loanId} status: {loan_status}")
        return loan_status
    except Exception as e:
        print(f"Error checking loan status: {str(e)}")
    finally:
        cursor.close()
```

```python
def calculateEMI(self, loanId: int):
    cursor = self.db_conn.cursor()
    try:
        cursor.execute("SELECT * FROM loan WHERE loanId=?", (loanId))
        loan_data = cursor.fetchone()
        if not loan_data:
            raise InvalidLoanException(f"Loan with ID {loanId} not found.")

        principal_amount = loan_data[2]
        interest_rate = loan_data[3]
        loan_tenure = loan_data[4]
        emi = (principal_amount * interest_rate * (1 + interest_rate) ** loan_tenure) / ((1 + interest_rate) ** loan_tenure - 1)
        print(f"EMI calculated for Loan {loanId}: {emi}")
        return emi
    except Exception as e:
        print(f"Error calculating EMI: {str(e)}")
    finally:
        cursor.close()
```

```python
def loanRepayment(self, loanId: int, amount: float):
    cursor = self.db_conn.cursor()
    try:
        cursor.execute("SELECT * FROM loans WHERE loanId=?", (loanId,))
        loan_data = cursor.fetchone()
        if not loan_data:
            raise InvalidLoanException(f"Loan with ID {loanId} not found.")

        principal_amount = loan_data[2]
        interest_rate = loan_data[3]
        loan_tenure = loan_data[4]
        emi = (principal_amount * interest_rate * (1 + interest_rate) ** loan_tenure) / ((1 + interest_rate) ** loan_tenure - 1)
        no_of_emis = int(amount / emi)
        remaining_amount = amount - (no_of_emis * emi)

        if no_of_emis < 1:
            print("Payment amount is less than one EMI. Payment rejected.")
        else:
            cursor.execute("UPDATE loan SET remainingAmount=?, loanStatus=? WHERE loanId=?",
                           (remaining_amount, "Fully Paid" if remaining_amount == 0 else "Partially Paid", loanId,))
            self.db_conn.commit()
            print(f"Payment processed for Loan {loanId}. {no_of_emis} EMIs paid.")
    except Exception as e:
        print(f"Error processing loan repayment: {str(e)}")
    finally:
        cursor.close()
```

```python
def getAllLoan(self) -> List[Loan]:
    cursor = self.db_conn.cursor()
    try:
        cursor.execute("SELECT * FROM loan")
        loans = []
        for row in cursor.fetchall():
            loan = Loan()
            loan.loanId = row[0]
            loan.customer.customerId = row[1]
            loan.principalAmount = row[2]
            loan.interestRate = row[3]
            loan.loanTerm = row[4]
            loan.loanType = row[5]
            loan.loanStatus = row[6]
            loan.propertyAddress = row[7]
            loan.propertyValue = row[8]
            loans.append(loan)
        return loans
    except Exception as e:
        print(f"Error fetching all loans: {str(e)}")
    finally:
        cursor.close()
```

```
def getLoanById(self, loanId: int) -> Loan:
    cursor = self.db_conn.cursor()
    try:
        cursor.execute("SELECT * FROM loan WHERE loanId=?", (loanId,))
        loan_data = cursor.fetchone()
        if not loan_data:
            raise InvalidLoanException(f"Loan with ID {loanId} not found.")

        loan = Loan()
        loan.loanId = loan_data[0]
        loan.customer.customerId = loan_data[1]
        loan.principalAmount = loan_data[2]
        loan.interestRate = loan_data[3]
        loan.loanTerm = loan_data[4]
        loan.loanType = loan_data[5]
        loan.loanStatus = loan_data[6]
        loan.propertyAddress = loan
    finally:
        cursor.close()
```

7. Create DBUtil class and add the following method.

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

```python
import mysql.connector

class DBUtil:
    @staticmethod
    def getDBConn():
        try:
            conn = (mysql.connector.connect(
                host="localhost",
                user="root",
                password="root",
                port='3306',
                database="LoanManagementSystem"
            ))
            return conn
        except mysql.connector.Error as e:
            print(f"Error connecting to MySQL database: {e}")
            return None
```

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."

```python
from entity import Loan
from step6  import ILoanRepositoryImpl

class LoanManagement:
    def __init__(self):
        self.repository = ILoanRepositoryImpl('LoanManagementSystem')

    def main(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice (1-5): ")
            if choice == '1':
                self.applyLoan()
            elif choice == '2':
                self.getAllLoan()
            elif choice == '3':
                self.getLoan()
            elif choice == '4':
                self.loanRepayment()
            elif choice == '5':
                print("Exiting Loan Management System. Goodbye!")
                break
            else:
                print("Invalid choice. Please enter a valid option.")
```

```python
def display_menu(self):
    print("===== Loan Management System =====")
    print("1. Apply for a Loan")
    print("2. View All Loans")
    print("3. View Loan Details")
    print("4. Process Loan Repayment")
    print("5. Exit")

def applyLoan(self):
    loan = Loan()
    loan.loanId = int(input("Enter Loan ID: "))
    loan.customer.customerId = int(input("Enter Customer ID: "))
    loan.principalAmount = float(input("Enter Principal Amount: "))
    loan.interestRate = float(input("Enter Interest Rate: "))
    loan.loanTerm = int(input("Enter Loan Term (in months): "))
    loan.loanType = input("Enter Loan Type: ")
    loan.loanStatus = "Pending"
    self.repository.applyLoan(loan)
```

```python
def getAllLoan(self):
    loans = self.repository.getAllLoan()
    for loan in loans:
        loan.printLoanInfo()

def getLoan(self):
    loanId = int(input("Enter Loan ID: "))
    loan = self.repository.getLoanById(loanId)
    if loan:
        loan.printLoanInfo()
    else:
        print("Loan not found.")

def loanRepayment(self):
    loanId = int(input("Enter Loan ID: "))
    amount = float(input("Enter Repayment Amount: "))
    self.repository.loanRepayment(loanId, amount)
```