Coding Challenge - Loan Management System

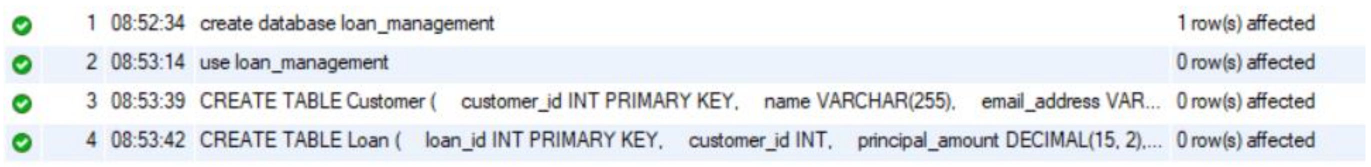
Problem Statement:

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

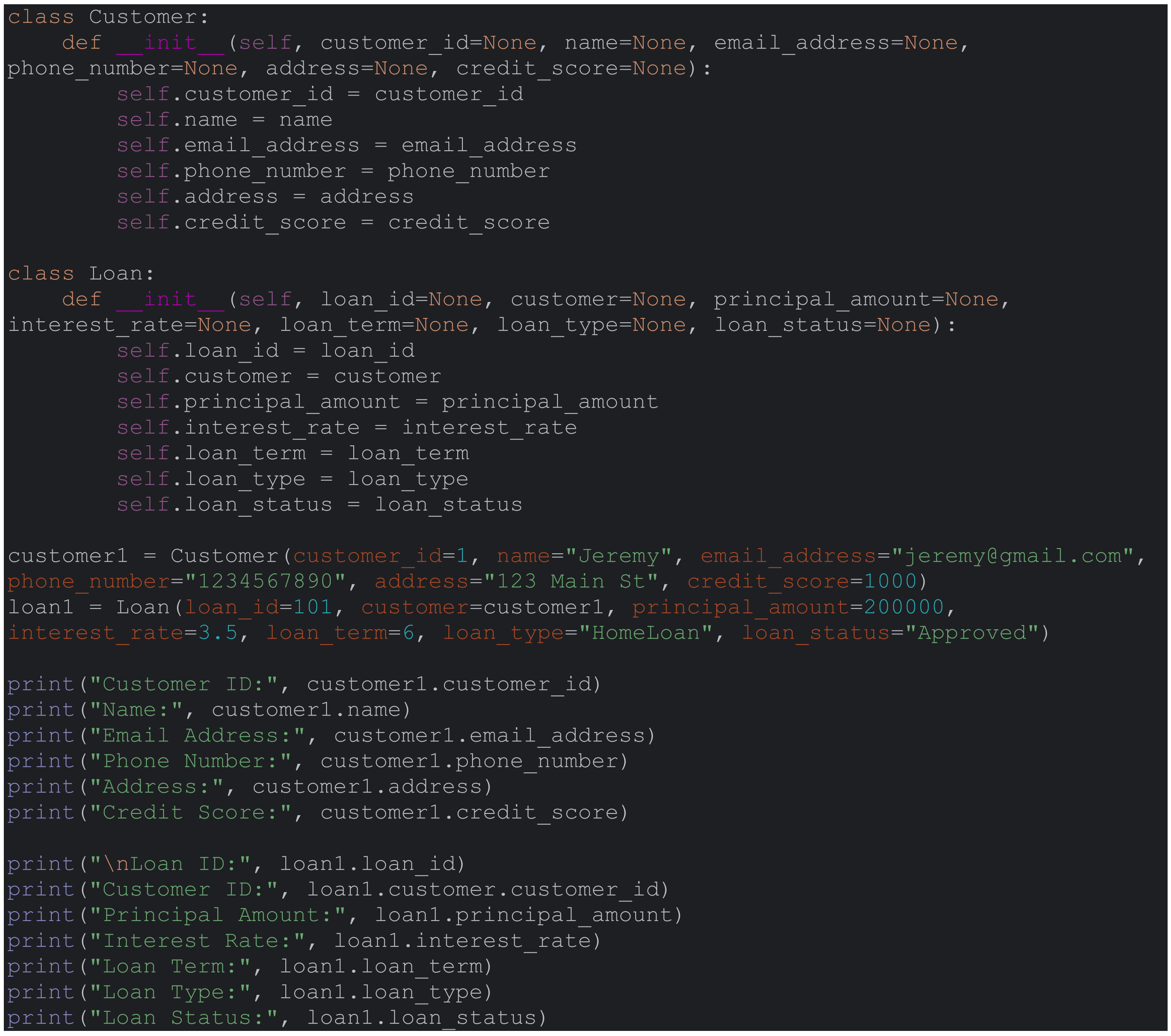
1. Define a **`Customer`** class with the following confidential attributes: a. Customer ID
b. Name
c. Email Address
d. Phone Number
e. Address
f. creditScore

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)

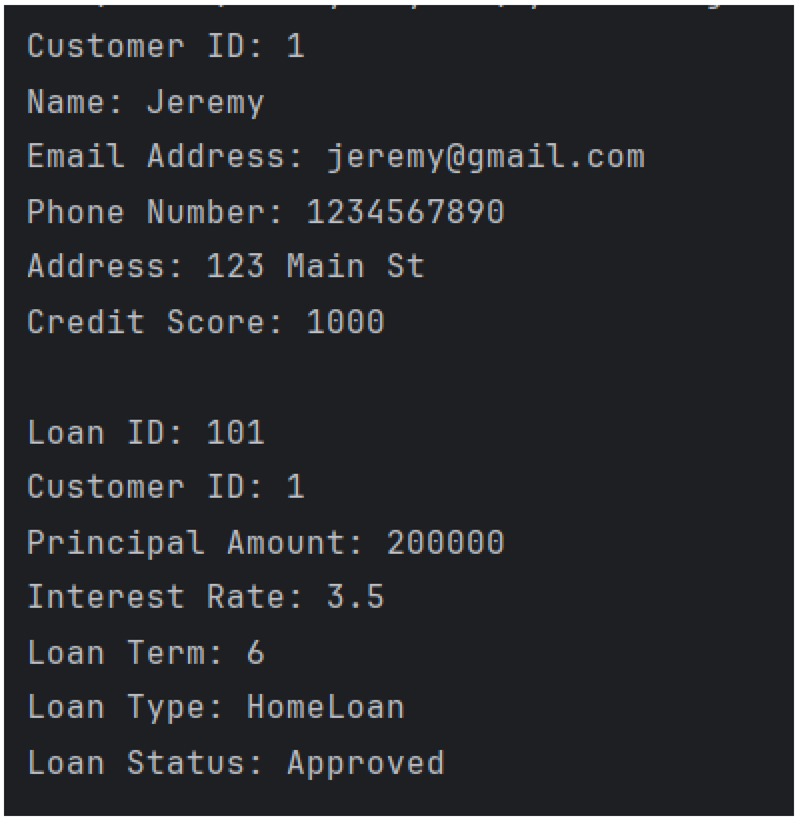
```
create database loan_management;

use loan_management;

CREATE TABLE Customer (

   customer_id INT PRIMARY KEY,

   name VARCHAR(255),

   email_address VARCHAR(255),

   phone_number VARCHAR(20),

   address VARCHAR(255),

   credit_score INT

);


CREATE TABLE Loan (

   loan_id INT PRIMARY KEY,

   customer_id INT,

   principal_amount DECIMAL(15, 2),

   interest_rate DECIMAL(5, 2),

   loan_term_months INT,

   loan_type VARCHAR(50),

   loan_status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);
```



```
✓ 1 08:52:34 create database loan_management                                                                          1 row(s) affected
✓ 2 08:53:14 use loan_management                                                                                      0 row(s) affected
✓ 3 08:53:39 CREATE TABLE Customer (   customer_id INT PRIMARY KEY,   name VARCHAR(255),   email_address VAR...  0 row(s) affected
✓ 4 08:53:42 CREATE TABLE Loan (   loan_id INT PRIMARY KEY,   customer_id INT,   principal_amount DECIMAL(15, 2),...  0 row(s) affected
```



```python
class Customer:
    def __init__(self, customer_id=None, name=None, email_address=None,
phone_number=None, address=None, credit_score=None):
        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

class Loan:
    def __init__(self, loan_id=None, customer=None, principal_amount=None,
interest_rate=None, loan_term=None, loan_type=None, loan_status=None):
        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

customer1 = Customer(customer_id=1, name="Jeremy", email_address="jeremy@gmail.com",
phone_number="1234567890", address="123 Main St", credit_score=1000)
loan1 = Loan(loan_id=101, customer=customer1, principal_amount=200000,
interest_rate=3.5, loan_term=6, loan_type="HomeLoan", loan_status="Approved")

print("Customer ID:", customer1.customer_id)
print("Name:", customer1.name)
print("Email Address:", customer1.email_address)
print("Phone Number:", customer1.phone_number)
print("Address:", customer1.address)
print("Credit Score:", customer1.credit_score)

print("\nLoan ID:", loan1.loan_id)
print("Customer ID:", loan1.customer.customer_id)
print("Principal Amount:", loan1.principal_amount)
print("Interest Rate:", loan1.interest_rate)
print("Loan Term:", loan1.loan_term)
print("Loan Type:", loan1.loan_type)
print("Loan Status:", loan1.loan_status)
```



```
Customer ID: 1
Name: Jeremy
Email Address: jeremy@gmail.com
Phone Number: 1234567890
Address: 123 Main St
Credit Score: 1000

Loan ID: 101
Customer ID: 1
Principal Amount: 200000
Interest Rate: 3.5
Loan Term: 6
Loan Type: HomeLoan
Loan Status: Approved
```

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, property_address=None, property_value=None, **kwargs):
        super().__init__(**kwargs)
        self.property_address = property_address
        self.property_value = property_value

class CarLoan(Loan):
    def __init__(self, car_model=None, car_value=None, **kwargs):
        super().__init__(**kwargs)
        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.

```
class HomeLoan(Loan):
    def __init__(self, property_address=None, property_value=None, **kwargs):
        super().__init__(**kwargs)
        self.property_address = property_address
        self.property_value = property_value

class CarLoan(Loan):
    def __init__(self, car_model=None, car_value=None, **kwargs):
        super().__init__(**kwargs)
        self.car_model = car_model
        self.car_value = car_value
```

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.

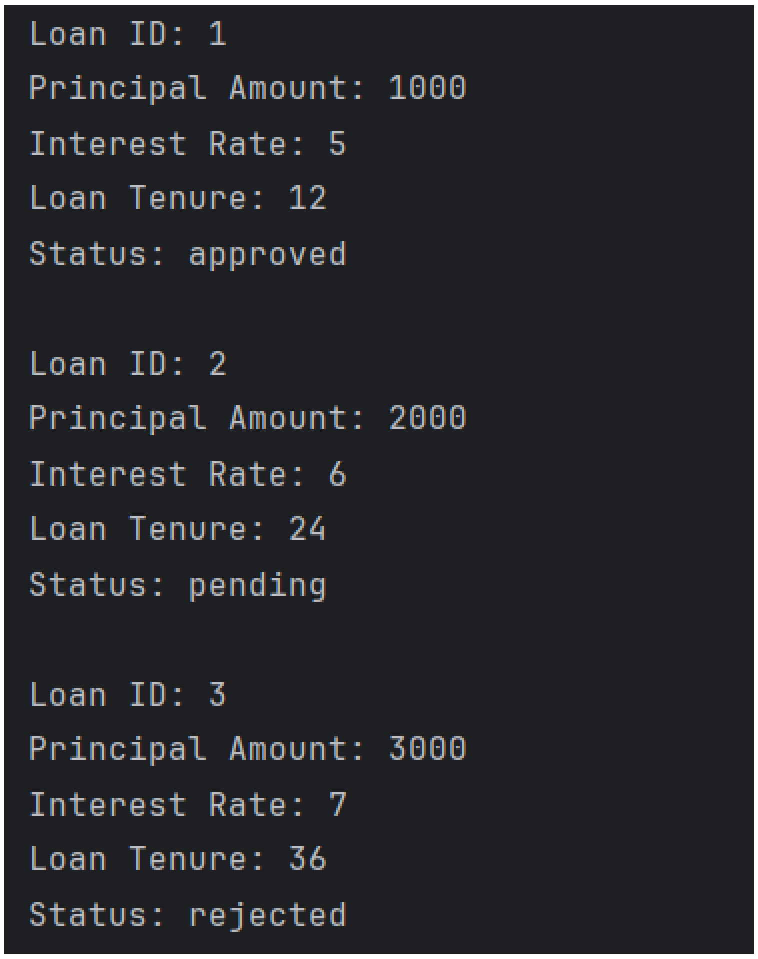
```
class HomeLoan(Loan):
    def __init__(self, property_address=None, property_value=None, **kwargs):
        super().__init__(**kwargs)
        self.property_address = property_address
        self.property_value = property_value

class CarLoan(Loan):
    def __init__(self, car_model=None, car_value=None, **kwargs):
        super().__init__(**kwargs)
        self.car_model = car_model
        self.car_value = car_value
```

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.

```
class ILoanRepositoryImpl(ILoanRepository):

    def __init__(self):
        self.loans = []

    def apply_loan(self, loan):
        self.loans.append(loan)

    def calculate_interest(self, loan_id):
        for loan in self.loans:
            if loan.loan_id == loan_id:
                return
        raise InvalidLoanException("Loan not found with ID: " + str(loan_id))

    def loan_status(self, loan_id):
        for loan in self.loans:
            if loan.loan_id == loan_id:
                return
        raise InvalidLoanException("Loan not found with ID: " + str(loan_id))

    def calculate_emi(self, loan_id):
        for loan in self.loans:
            if loan.loan_id == loan_id:
                return
        raise InvalidLoanException("Loan not found with ID: " + str(loan_id))

    def calculate_emi_with_params(self, principal_amount, interest_rate,
loan_tenure):
        r = interest_rate / (12 * 100)
        n = loan_tenure
        emi = (principal_amount * r * (1 + r) ** n) / ((1 + r) ** n - 1)
        return emi

    def loan_repayment(self, loan_id, amount):
        for loan in self.loans:
            if loan.loan_id == loan_id:
                return
        raise InvalidLoanException("Loan not found with ID: " + str(loan_id))

    def get_all_loan(self):
        for loan in self.loans:
            print("Loan ID:", loan.loan_id)
            print("Principal Amount:", loan.principal_amount)
            print("Interest Rate:", loan.interest_rate)
            print("Loan Tenure:", loan.loan_tenure)
            print("Status:", loan.status)
            print()

    def get_loan_by_id(self, loan_id):
        for loan in self.loans:
            if loan.loan_id == loan_id:
                return loan
        raise InvalidLoanException("Loan not found with ID: " + str(loan_id))
epository = ILoanRepositoryImpl()

loan = Loan(1, 1000, 5, 12, "pending")
repository.apply_loan(loan)
```

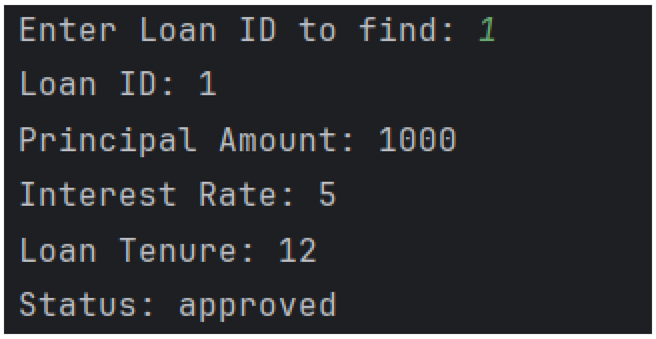
```python
repository.calculate_interest(1)

repository.loan_status(1)

repository.calculate_emi(1)

repository.loan_repayment(1, 500)

repository.get_all_loan()

loan_id = 1
found_loan = repository.get_loan_by_id(loan_id)
print("Found loan:", found_loan)
```



```
Loan ID: 1
Principal Amount: 1000
Interest Rate: 5
Loan Tenure: 12
Status: approved

Loan ID: 2
Principal Amount: 2000
Interest Rate: 6
Loan Tenure: 24
Status: pending

Loan ID: 3
Principal Amount: 3000
Interest Rate: 7
Loan Tenure: 36
Status: rejected
```

```python
from abc import ABC, abstractmethod

class InvalidLoanException(Exception):
    pass

class Loan:
    def __init__(self, loan_id, principal_amount, interest_rate, loan_tenure,
status="pending"):
        self.loan_id = loan_id
        self.principal_amount = principal_amount
        self.interest_rate = interest_rate
        self.loan_tenure = loan_tenure
        self.status = status

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 calculate_emi_with_params(self, principal_amount, interest_rate,
loan_tenure):
        pass

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

    @abstractmethod
    def get_all_loan(self):
        pass

    @abstractmethod
    def get_loan_by_id(self, loan_id):
        pass

class LoanRepository(ILoanRepository):

    def __init__(self):
        self.loans = []

    def apply_loan(self, loan):
        pass

    def calculate_interest(self, loan_id):
        pass

    def loan_status(self, loan_id):
        pass

    def calculate_emi(self, loan_id):
        pass

    def calculate_emi_with_params(self, principal_amount, interest_rate,
loan_tenure):
        r = interest_rate / (12 * 100)
        n = loan_tenure
        emi = (principal_amount * r * (1 + r) ** n) / ((1 + r) ** n - 1)
        return emi

    def loan_repayment(self, loan_id, amount):
        pass

    def get_all_loan(self):
        for loan in self.loans:
```

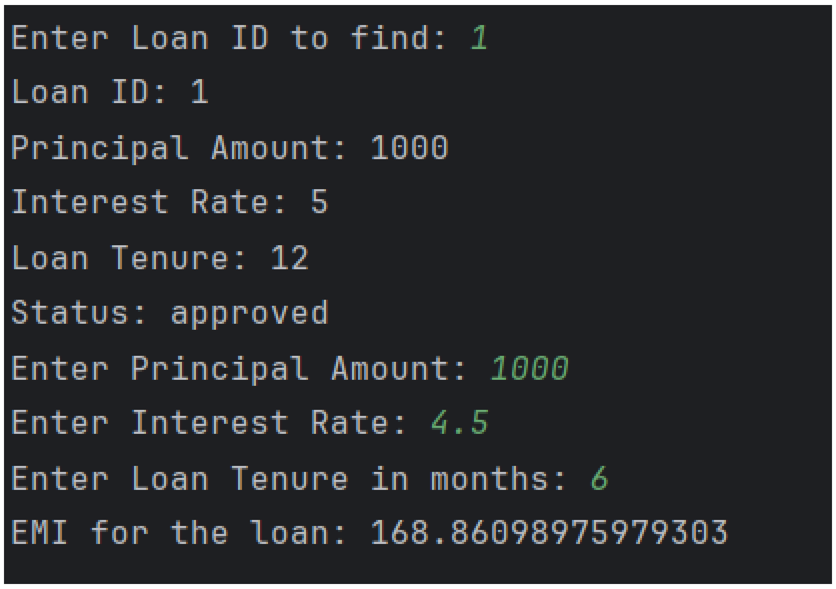
```python
            print("Loan ID:", loan.loan_id)
            print("Principal Amount:", loan.principal_amount)
            print("Interest Rate:", loan.interest_rate)
            print("Loan Tenure:", loan.loan_tenure)
            print("Status:", loan.status)
            print()

    def get_loan_by_id(self, loan_id):
        found_loan = None
        for loan in self.loans:
            if loan.loan_id == loan_id:
                found_loan = loan
                break
        return found_loan


repository = LoanRepository()

repository.loans.append(Loan(1, 1000, 5, 12, "approved"))
repository.loans.append(Loan(2, 2000, 6, 24, "pending"))
repository.loans.append(Loan(3, 3000, 7, 36, "rejected"))

loan_id = int(input("Enter Loan ID to find: "))

found_loan = repository.get_loan_by_id(loan_id)
if found_loan is not None:
    # Display loan details
    print("Loan ID:", found_loan.loan_id)
    print("Principal Amount:", found_loan.principal_amount)
    print("Interest Rate:", found_loan.interest_rate)
    print("Loan Tenure:", found_loan.loan_tenure)
    print("Status:", found_loan.status)
else:
    print("Loan not found with ID:", loan_id)
```



```
Enter Loan ID to find: 1
Loan ID: 1
Principal Amount: 1000
Interest Rate: 5
Loan Tenure: 12
Status: approved
```

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

```python
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 calculate_emi_with_params(self, principal_amount, interest_rate,
loan_tenure):
        pass

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

    @abstractmethod
    def get_all_loan(self):
        pass

    @abstractmethod
    def get_loan_by_id(self, loan_id):
        pass

class LoanRepository(ILoanRepository):

    def __init__(self):
        self.loans = []

    def apply_loan(self, loan):
        pass

    def calculate_interest(self, loan_id):
        pass

    def loan_status(self, loan_id):
        pass

    def calculate_emi(self, loan_id):
        pass

    def calculate_emi_with_params(self, principal_amount, interest_rate,
loan_tenure):
        r = interest_rate / (12 * 100)
        n = loan_tenure
        emi = (principal_amount * r * (1 + r) ** n) / ((1 + r) ** n - 1)
        return emi

    def loan_repayment(self, loan_id, amount):
        pass

    def get_all_loan(self):
        for loan in self.loans:
            print("Loan ID:", loan.loan_id)
            print("Principal Amount:", loan.principal_amount)
            print("Interest Rate:", loan.interest_rate)
            print("Loan Tenure:", loan.loan_tenure)
            print("Status:", loan.status)
            print()

    def get_loan_by_id(self, loan_id):
        found_loan = None
        for loan in self.loans:
            if loan.loan_id == loan_id:
                found_loan = loan
                break
        return found_loan


repository = LoanRepository()

repository.loans.append(Loan(1, 1000, 5, 12, "approved"))
repository.loans.append(Loan(2, 2000, 6, 24, "pending"))
repository.loans.append(Loan(3, 3000, 7, 36, "rejected"))

loan_id = int(input("Enter Loan ID to find: "))

found_loan = repository.get_loan_by_id(loan_id)
if found_loan is not None:
    # Display loan details
    print("Loan ID:", found_loan.loan_id)
    print("Principal Amount:", found_loan.principal_amount)
    print("Interest Rate:", found_loan.interest_rate)
    print("Loan Tenure:", found_loan.loan_tenure)
    print("Status:", found_loan.status)
else:
    print("Loan not found with ID:", loan_id)


principal_amount = float(input("Enter Principal Amount: "))
interest_rate = float(input("Enter Interest Rate: "))
loan_tenure = int(input("Enter Loan Tenure in months: "))

emi = repository.calculate_emi_with_params(principal_amount, interest_rate,
loan_tenure)
print("EMI for the loan:", emi)
```

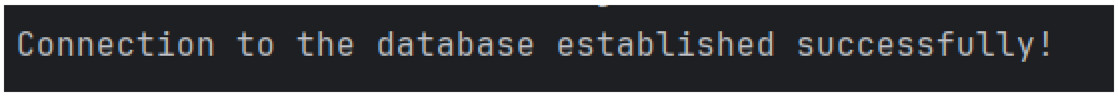


```
Enter Loan ID to find: 1
Loan ID: 1
Principal Amount: 1000
Interest Rate: 5
Loan Tenure: 12
Status: approved
Enter Principal Amount: 1000
Enter Interest Rate: 4.5
Enter Loan Tenure in months: 6
EMI for the loan: 168.86098975979303
```

7. Create **DBUtil** class and add the following method.

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

```
def getDBConn():
    import mysql.connector

    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="root",
            password="root",
            port="3306",
            database="loan_management")

        print("Connection to the database established successfully!")

        return connection

    except mysql.connector.Error as e:
        print("Error connecting to the database:", e)
        return None


database_connection = getDBConn()
```



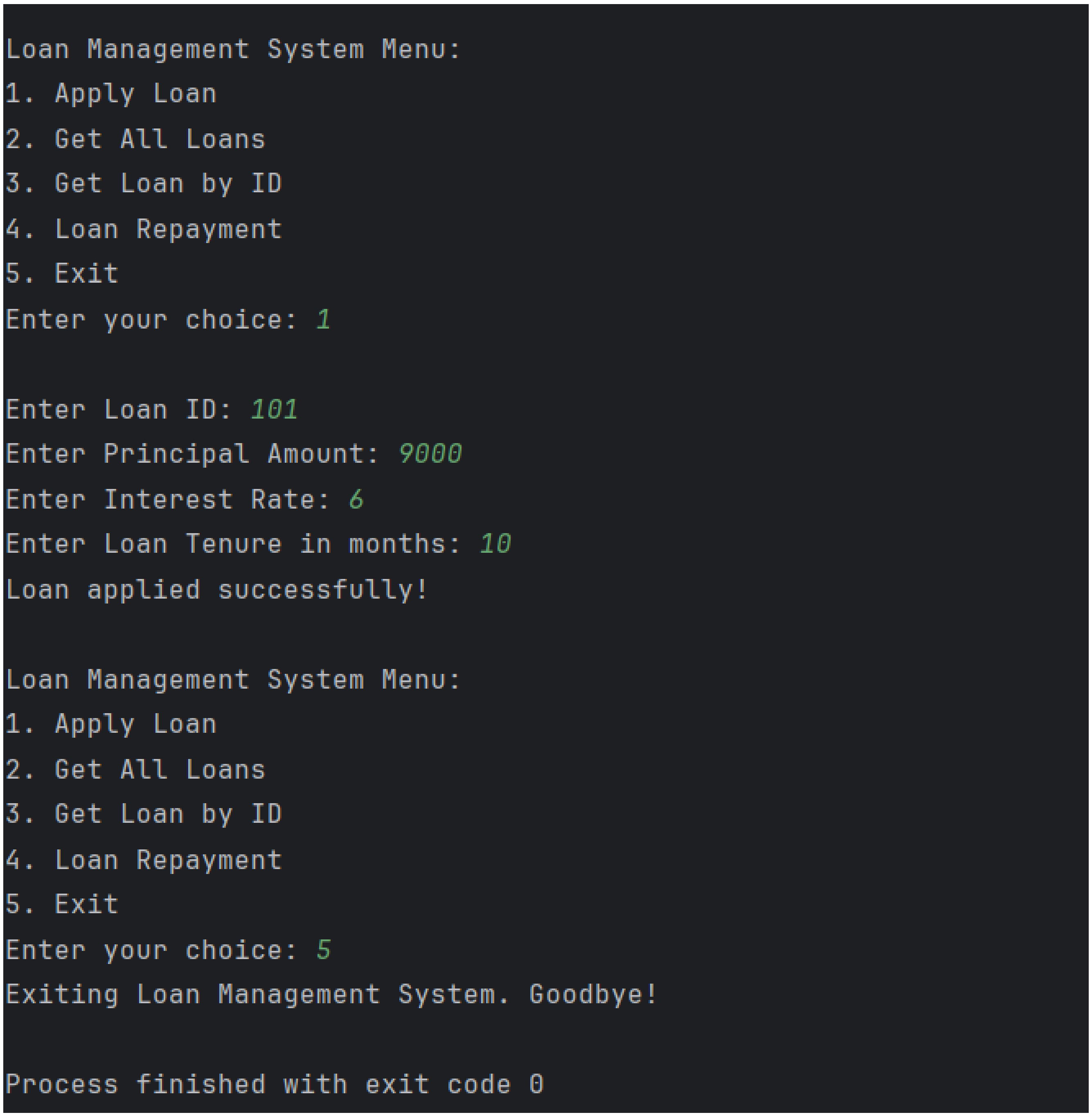
```
Connection to the database established successfully!
```

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."

```
class LoanManagement:
    @staticmethod
    def main():
        repository = LoanRepository()

        while True:
            print("\nLoan Management System Menu:")
            print("1. Apply Loan")
            print("2. Get All Loans")
            print("3. Get Loan by ID")
            print("4. Loan Repayment")
            print("5. Exit")

            choice = input("Enter your choice: ")

            if choice == "1":
                try:
                    loan_id = int(input("\nEnter Loan ID: "))
                    principal_amount = float(input("Enter Principal Amount: "))
                    interest_rate = float(input("Enter Interest Rate: "))
                    loan_tenure = int(input("Enter Loan Tenure in months: "))

                    new_loan = Loan(loan_id, principal_amount, interest_rate,
loan_tenure)
```

```python
                    repository.apply_loan(new_loan)

                    print("Loan applied successfully!")

                except ValueError:
                    print("Invalid input. Please enter valid values.")

            elif choice == "2":
                # Get all loans
                repository.get_all_loan()
            elif choice == "3":
                # Get loan by ID
                loan_id = int(input("Enter Loan ID: "))
                try:
                    loan = repository.get_loan_by_id(loan_id)
                    print("Loan Details:")
                    print("Loan ID:", loan.loan_id)
                    print("Principal Amount:", loan.principal_amount)
                    print("Interest Rate:", loan.interest_rate)
                    print("Loan Tenure:", loan.loan_tenure)
                    print("Status:", loan.status)
                except InvalidLoanException as e:
                    print(e)
            elif choice == "4":
                pass
            elif choice == "5":
                print("Exiting Loan Management System. Goodbye!")
                break
            else:
                print("Invalid choice. Please enter a valid option.")


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


```
Loan Management System Menu:
1. Apply Loan
2. Get All Loans
3. Get Loan by ID
4. Loan Repayment
5. Exit
Enter your choice: 1

Enter Loan ID: 101
Enter Principal Amount: 9000
Enter Interest Rate: 6
Enter Loan Tenure in months: 10
Loan applied successfully!

Loan Management System Menu:
1. Apply Loan
2. Get All Loans
3. Get Loan by ID
4. Loan Repayment
5. Exit
Enter your choice: 5
Exiting Loan Management System. Goodbye!

Process finished with exit code 0
```