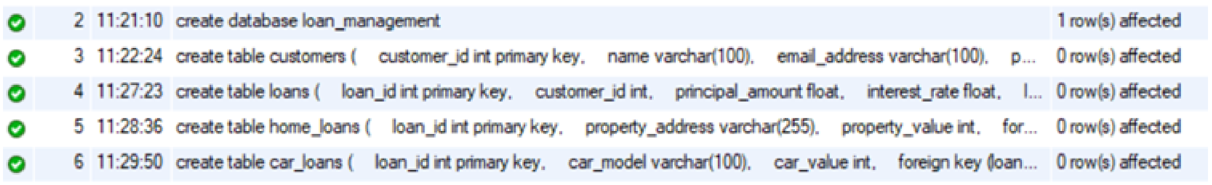
Coding Challenge - Loan Management System

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

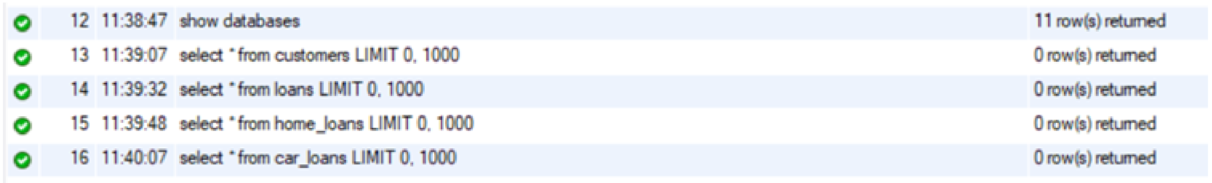
SQL Code

```
create database loan_management;

create table customers (
	customer_id int primary key,
	name varchar(100),
	email_address varchar(100),
	phone_number varchar(15),
	address varchar(255),
	credit_score int
);

create table loans (
	loan_id int primary key,
	customer_id int,
	principal_amount float,
	interest_rate float,
	loan_term int,
	loan_type enum('HomeLoan', 'CarLoan') not null,
	loan_status enum('Pending', 'Approved', 'Rejected') default 'Pending',
	foreign key (customer_id) references customers(customer_id)
);

create table home_loans (
	loan_id int primary key,
	property_address varchar(255),
	property_value int,
	foreign key (loan_id) references loans(loan_id)
);

create table car_loans (
	loan_id int primary key,
	car_model varchar(100),
        car_value int,
        foreign key (loan_id) references loans(loan_id)
);

select * from customers;
select * from loans;
select * from home_loans;
select * from car_loans;
```

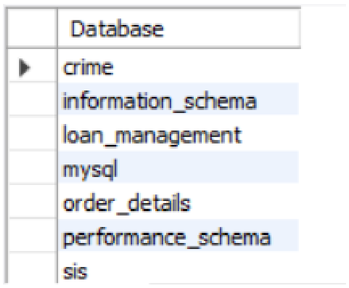
Output



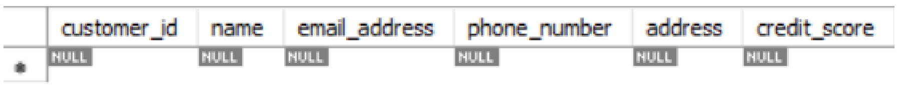
```
2   11:21:10  create database loan_management                                                                              1 row(s) affected
3   11:22:24  create table customers (   customer_id int primary key,   name varchar(100),   email_address varchar(100),   p...   0 row(s) affected
4   11:27:23  create table loans (   loan_id int primary key,   customer_id int,   principal_amount float,   interest_rate float,   l...   0 row(s) affected
5   11:28:36  create table home_loans (   loan_id int primary key,   property_address varchar(255),   property_value int,   for...   0 row(s) affected
6   11:29:50  create table car_loans (   loan_id int primary key,   car_model varchar(100),   car_value int,   foreign key (loan...   0 row(s) affected
```



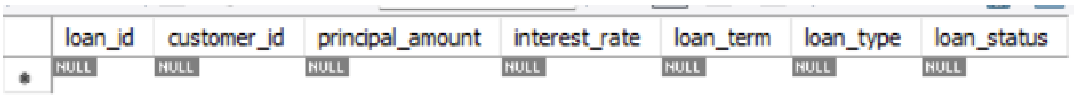
```
12  11:38:47  show databases                                11 row(s) returned
13  11:39:07  select * from customers LIMIT 0, 1000         0 row(s) returned
14  11:39:32  select * from loans LIMIT 0, 1000             0 row(s) returned
15  11:39:48  select * from home_loans LIMIT 0, 1000        0 row(s) returned
16  11:40:07  select * from car_loans LIMIT 0, 1000         0 row(s) returned
```



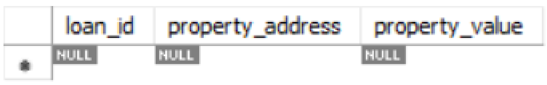
Database
crime
information_schema
loan_management
mysql
order_details
performance_schema
sis



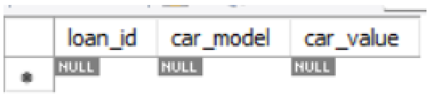
customer_id	name	email_address	phone_number	address	credit_score
NULL	NULL	NULL	NULL	NULL	NULL



loan_id	customer_id	principal_amount	interest_rate	loan_term	loan_type	loan_status
NULL	NULL	NULL	NULL	NULL	NULL	NULL



loan_id	property_address	property_value
NULL	NULL	NULL



loan_id	car_model	car_value
NULL	NULL	NULL

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

entity/customer.py

```
class Customer:
   def __init__(self, customer_id=None, name=None, email=None, phone=None,
address=None, credit_score=None):
       self.__customer_id = customer_id
       self.__name = name
       self.__email = email
       self.__phone = phone
       self.__address = address
       self.__credit_score = credit_score


   @property
   def customer_id(self):
       return self.__customer_id

   @customer_id.setter
   def customer_id(self, value):
       self.__customer_id = value


   @property
   def name(self):
       return self.__name

   @name.setter
   def name(self, value):
       self.__name = value


   @property
   def email(self):
       return self.__email

   @email.setter
   def email(self, value):
       self.__email = value


   @property
   def phone(self):
       return self.__phone
   @phone.setter
   def phone(self, value):
       self.__phone = value


   @property
   def address(self):
       return self.__address

   @address.setter
   def address(self, value):
       self.__address = value

   @property
   def credit_score(self):
       return self.__credit_score

   @credit_score.setter
   def credit_score(self, value):
       self.__credit_score = value

   def __str__(self):
       return (
           f"[Customer] ID: {self.customer_id}, Name: {self.name}, Email:
{self.email}, "
           f"Phone: {self.phone}, Address: {self.address}, Credit Score:
{self.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)

Code
entity/loan.py

```
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
```

```python
        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

    @property
    def loan_id(self):
        return self.__loan_id
    @loan_id.setter
    def loan_id(self, loan_id):
        self.__loan_id = loan_id

    @property
    def customer(self):
        return self.__customer
    @customer.setter
    def customer(self, customer):
        self.__customer = customer

    @property
    def principal_amount(self):
        return self.__principal_amount
    @principal_amount.setter
    def principal_amount(self, principal_amount):
        self.__principal_amount = principal_amount

    @property
    def interest_rate(self):
        return self.__interest_rate
    @interest_rate.setter
    def interest_rate(self, interest_rate):
        self.__interest_rate = interest_rate

    @property
    def loan_term(self):
        return self.__loan_term
    @loan_term.setter
    def loan_term(self, loan_term):
        self.__loan_term = loan_term

    @property
    def loan_type(self):
        return self.__loan_type
    @loan_type.setter
    def loan_type(self, loan_type):
        self.__loan_type = loan_type

    @property
    def loan_status(self):
        return self.__loan_status
    @loan_status.setter
    def loan_status(self, loan_status):
        self.__loan_status = loan_status

    def __str__(self):
        return "[Loan] ID: {}, Customer ID: {}, Principal: {}, Interest Rate: 
{}, Term: {} months, Type: {}, Status: {}".format(
            self.loan_id,
            self.customer.customer_id if self.customer else "N/A",
            self.principal_amount,
            self.interest_rate,
            self.loan_term,
            self.loan_type,
            self.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.

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

entity/home_loan.py

```
from entity.loan import Loan

class HomeLoan(Loan):
  def __init__(self, loan_id=None, customer=None, principal_amount=None, interest_rate=None, 
loan_term=None, loan_type="HomeLoan", loan_status="Pending", property_address=None, 
property_value=None):
    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

  @property
  def property_address(self):
    return self.__property_address
  @property_address.setter
  def property_address(self, property_address):
    self.__property_address = property_address

  @property
  def property_value(self):
    return self.__property_value
  @property_value.setter
  def property_value(self, property_value):
    self.__property_value = property_value

  def __str__(self):
    return super().__str__() + ", Property Address: {}, Property Value: {}".format(
        self.property_address,
        self.property_value
    )
```

b. CarLoan should have a carModel (String) and carValue (int) attribute.
entity/car_loan.py

```
from entity.loan import Loan

class CarLoan(Loan):
    def __init__(self, loan_id=None, customer=None, principal_amount=None, interest_rate=None,
loan_term=None, loan_type="CarLoan", loan_status="Pending", car_model=None, car_value=None):
        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

    @property
    def car_model(self):
        return self.__car_model
    @car_model.setter
    def car_model(self, car_model):
        self.__car_model = car_model

    @property
    def car_value(self):
        return self.__car_value
    @car_value.setter
    def car_value(self, car_value):
        self.__car_value = car_value

    def __str__(self):
        return super().__str__() + ", Car Model: {}, Car Value: {}".format(
            self.car_model,
            self.car_value
        )
```

4. Implement the following for all classes.

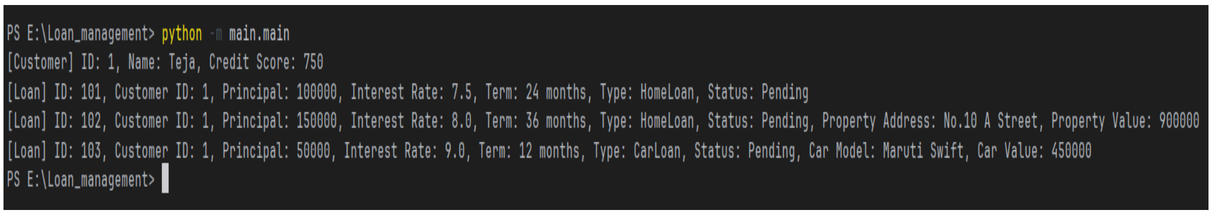
a. Write default constructors and overload the constructor with parameters, generate getter and setter, (print all information of attribute) methods for the attributes.

main/main.py

```
from entity.customer import Customer
from entity.loan import Loan
from entity.home_loan import HomeLoan
from entity.car_loan import CarLoan
```

```python
def main():
    customer = Customer(1, "Teja", "teja@email.com", "9876543210", "Chennai", 750)
    loan = Loan(101, customer, 100000, 7.5, 24, "HomeLoan", "Pending")
    home_loan = HomeLoan(102, customer, 150000, 8.0, 36, "HomeLoan", "Pending",
                         "No.10 A Street", 900000)
    car_loan = CarLoan(103, customer, 50000, 9.0, 12, "CarLoan", "Pending",
                       "Maruti Swift", 450000)

    print(customer)
    print(loan)
    print(home_loan)
    print(car_loan)

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

Output



```
PS E:\Loan_management> python -m main.main
[Customer] ID: 1, Name: Teja, Credit Score: 750
[Loan] ID: 101, Customer ID: 1, Principal: 100000, Interest Rate: 7.5, Term: 24 months, Type: HomeLoan, Status: Pending
[Loan] ID: 102, Customer ID: 1, Principal: 150000, Interest Rate: 8.0, Term: 36 months, Type: HomeLoan, Status: Pending, Property Address: No.10 A Street, Property Value: 900000
[Loan] ID: 103, Customer ID: 1, Principal: 50000, Interest Rate: 9.0, Term: 12 months, Type: CarLoan, Status: Pending, Car Model: Maruti Swift, Car Value: 450000
PS E:\Loan_management>
```

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
def apply_loan(self, loan):
   conn = None
   cursor = None
   try:
       confirm = input("Do you want to apply for the loan? (Yes/No):
").strip().lower()
       if confirm != 'yes':
           print("Loan application cancelled.")
           return

       conn = DBUtil.get_connection()
       if conn is None:
           print("Error: MySQL Connection not available.")
           return
```

```
        cursor = conn.cursor()

        query = """
                INSERT INTO loans (loan_id, customer_id, principal_amount,
interest_rate, loan_term, loan_type, loan_status)
                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
        )

        cursor.execute(query, values)
        conn.commit()
        print("Loan application submitted successfully.")

    except Exception as e:
        print("Error applying loan:", e)

    finally:
        if 'cursor' in locals() and cursor:
            cursor.close()
```

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):
    conn = None
    cursor = None
    try:
        conn = DBUtil.get_connection()
        if conn is None:
            print("Error: MySQL Connection not available.")
            return

        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM
loans 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(f"Loan not found for ID: {loan_id}")

    except InvalidLoanException as e:
        print(f"Error: {e}")

    finally:
        if 'cursor' in locals() and cursor:
            cursor.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.

```
def loan_status(self, loan_id):
    conn = None
    cursor = None
    try:
        conn = DBUtil.get_connection()
        if conn is None:
            print("Error: MySQL Connection not available.")
            return

        cursor = conn.cursor()
        cursor.execute("""
            SELECT c.credit_score FROM loans l
            JOIN customers 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 loans 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(f"Loan not found for ID: {loan_id}")

    except InvalidLoanException as e:
        print(f"Error: {e}")

    finally:
        if 'cursor' in locals() and cursor:
            cursor.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.

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.

```
def calculate_emi(self, loan_id):
    conn = None
    cursor = None
    try:
        conn = DBUtil.get_connection()
        if conn is None:
            print("Error: MySQL Connection not available.")
            return

        cursor = conn.cursor()
        cursor.execute("SELECT principal_amount, interest_rate, loan_term FROM
loans WHERE loan_id = %s", (loan_id,))
        row = cursor.fetchone()

        if row:
            P, R_annual, N = row
            R = R_annual / (12 * 100)
            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(f"Loan not found for ID: {loan_id}")

    except InvalidLoanException as e:
        print(f"Error: {e}")

    finally:
        if 'cursor' in locals() and cursor:
            cursor.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"Error: {e}")
   return emi
```

```
def calculate_emi_overload(self, *args):
   if len(args) == 1:
       return self.calculate_emi(args[0])
   elif len(args) == 3:
       return self.calculate_emi_manual(args[0], args[1], args[2])
   else:
       raise ValueError("Invalid arguments for calculate_emi_overload")
```

```
def calculate_interest_overload(self, *args):
   if len(args) == 1:
       return self.calculate_interest(args[0])
   elif len(args) == 3:
       return self.calculate_interest_manual(args[0], args[1], args[2])
   else:
       raise ValueError("Invalid arguments for calculate_interest_overload")
```

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

```
def get_all_loans(self):
   conn = None
   cursor = None
   try:
       conn = DBUtil.get_connection()
       if conn is None:
           print("Error: MySQL Connection not available.")
           return

       cursor = conn.cursor()
       cursor.execute("SELECT * FROM loans")
       rows = cursor.fetchall()
       print("All Loans:")
       for row in rows:
           print(row)
   except Exception as e:
       print("Error fetching loans:", e)
   finally:
       if 'cursor' in locals() and cursor:
           cursor.close()
```

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

```
def get_loan_by_id(self, loan_id):
   conn = None
```

```
    cursor = None
    try:
        conn = DBUtil.get_connection()
        if conn is None:
            print("Error: MySQL Connection not available.")
            return

        cursor = conn.cursor()
        cursor.execute("SELECT * FROM loans 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(f"Loan not found for ID: {loan_id}")

    except InvalidLoanException as e:
        print(f"Error: {e}")

    finally:
        if 'cursor' in locals() and cursor:
            cursor.close()
```

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

dao/iloan_repository

```
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_overload(self, principal, rate, term):
        pass

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

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

   @abstractmethod
   def calculate_emi_overload(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
```

exception/exceptions.py

```python
class InvalidLoanException(Exception):
   def __init__(self, message="Loan not found with the given ID."):
       self.message = message
       super().__init__(self.message)
```

dao/iloan_repository_implementation

```python
import mysql.connector
from dao.iloan_repository import ILoanRepository
from exeception.exceptions import InvalidLoanException
from util.db_conn_util import DBUtil
from entity.loan import Loan

class LoanRepositoryImpl(ILoanRepository):

   def apply_loan(self, loan):
       conn = None
       cursor = None
       try:
           confirm = input("Do you want to apply for the loan? (Yes/No):
").strip().lower()
           if confirm != 'yes':
               print("Loan application cancelled.")
               return

           conn = DBUtil.get_connection()
           if conn is None:
               print("Error: MySQL Connection not available.")
               return
```

```
            cursor = conn.cursor()

            query = """
                   INSERT INTO loans (loan_id, customer_id, principal_amount,
interest_rate, loan_term, loan_type, loan_status)
                   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
            )

            cursor.execute(query, values)
            conn.commit()
            print("Loan application submitted successfully.")

        except Exception as e:
            print("Error applying loan:", e)

        finally:
            if 'cursor' in locals() and cursor:
                cursor.close()

    def calculate_interest(self, loan_id):
        conn = None
        cursor = None
        try:
            conn = DBUtil.get_connection()
            if conn is None:
                print("Error: MySQL Connection not available.")
                return

            cursor = conn.cursor()
            cursor.execute("SELECT principal_amount, interest_rate, loan_term
FROM loans 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(f"Loan not found for ID:
{loan_id}")

        except InvalidLoanException as e:
            print(f"Error: {e}")
```

```python
        finally:
            if 'cursor' in locals() and cursor:
                cursor.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):
        conn = None
        cursor = None
        try:
            conn = DBUtil.get_connection()
            if conn is None:
                print("Error: MySQL Connection not available.")
                return

            cursor = conn.cursor()
            cursor.execute("""
                SELECT c.credit_score FROM loans l
                JOIN customers 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 loans 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(f"Loan not found for ID:
{loan_id}")

        except InvalidLoanException as e:
            print(f"Error: {e}")

        finally:
            if 'cursor' in locals() and cursor:
                cursor.close()


    def calculate_emi(self, loan_id):
        conn = None
        cursor = None
        try:
            conn = DBUtil.get_connection()
            if conn is None:
                print("Error: MySQL Connection not available.")
                return

            cursor = conn.cursor()
            cursor.execute("SELECT principal_amount, interest_rate, loan_term 
FROM loans WHERE loan_id = %s", (loan_id,))
            row = cursor.fetchone()

            if row:
                P, R_annual, N = row
                R = R_annual / (12 * 100)
                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(f"Loan not found for ID: 
{loan_id}")

        except InvalidLoanException as e:
            print(f"Error: {e}")

        finally:
            if 'cursor' in locals() and cursor:
                cursor.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"Error: {e}")
        return emi

    def calculate_emi_overload(self, *args):
        if len(args) == 1:
            return self.calculate_emi(args[0])
        elif len(args) == 3:
            return self.calculate_emi_manual(args[0], args[1], args[2])
        else:
            raise ValueError("Invalid arguments for calculate_emi_overload")

    def calculate_interest_overload(self, *args):
        if len(args) == 1:
            return self.calculate_interest(args[0])
        elif len(args) == 3:
            return self.calculate_interest_manual(args[0], args[1], args[2])
        else:
            raise ValueError("Invalid arguments for 
calculate_interest_overload")

    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 get_all_loans(self):
        conn = None
        cursor = None
        try:
            conn = DBUtil.get_connection()
            if conn is None:
                print("Error: MySQL Connection not available.")
                return

            cursor = conn.cursor()
            cursor.execute("SELECT * FROM loans")
            rows = cursor.fetchall()
            print("All Loans:")
            for row in rows:
                print(row)
        except Exception as e:
            print("Error fetching loans:", e)
        finally:
            if 'cursor' in locals() and cursor:
                cursor.close()

    def get_loan_by_id(self, loan_id):
        conn = None
        cursor = None
        try:
            conn = DBUtil.get_connection()
            if conn is None:
                print("Error: MySQL Connection not available.")
                return

            cursor = conn.cursor()
            cursor.execute("SELECT * FROM loans 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(f"Loan not found for ID: 
{loan_id}")

        except InvalidLoanException as e:
            print(f"Error: {e}")
```

```
       finally:
           if 'cursor' in locals() and cursor:
               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

util/db_conn_util.py

```
import mysql.connector

class DBUtil:
   @staticmethod
   def get_connection():
       try:
           conn = mysql.connector.connect(
               host="localhost",
               user="root",
               password="Tejashree85!",
               database="loan_management"
           )
           return conn
       except mysql.connector.Error as e:
           print("MySQL Connection Error:", e)
           return None
```

util/db_property_util.py

```
class PropertyUtil:

   @staticmethod
   def get_property_string(file_path='config/db.properties'):
       properties = {}
       try:
           with open(file_path, 'r') as f:
               for line in f:
                   line = line.strip()
                   if line and not line.startswith('#') and '=' in line:
                       key, value = line.split('=', 1)
                       properties[key.strip()] = value.strip()
       except FileNotFoundError:
           print(f"Error: Property file '{file_path}' not found.")
       except Exception as e:
           print(f"Error reading property file: {e}")
```

```
        return properties
```

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."

main/main.py

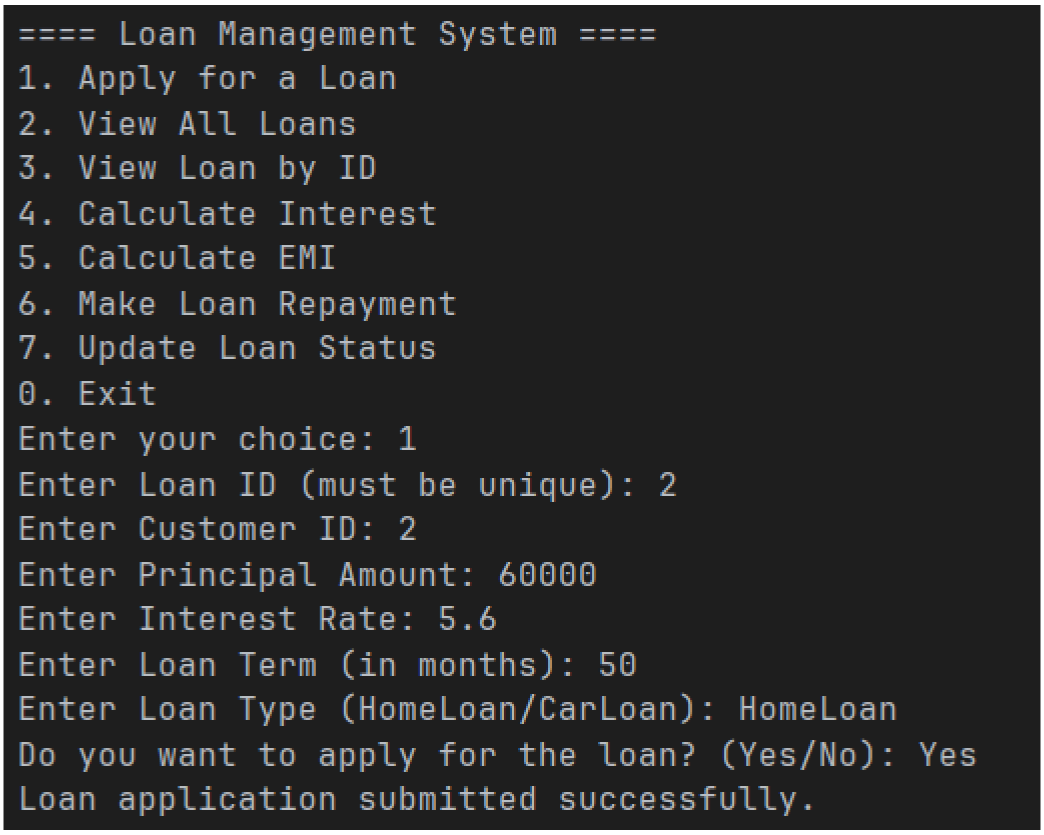
```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

from dao.iloan_repository_implementation 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':
           try:
               loan_id = int(input("Enter Loan ID (must be unique): "))
               customer_id = int(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):
").strip()
```

```
            customer = Customer(
                customer_id=customer_id,
                name="",
                email="",
                phone="",
                address="",
                credit_score=700
            )

            loan = Loan(
                loan_id=loan_id,
                customer=customer,
                principal_amount=principal,
                interest_rate=rate,
                loan_term=term,
                loan_type=loan_type,
                loan_status="Pending"
            )

            repo.apply_loan(loan)

        except Exception as e:
            print(f"Error: {e}")

    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:
```

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

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

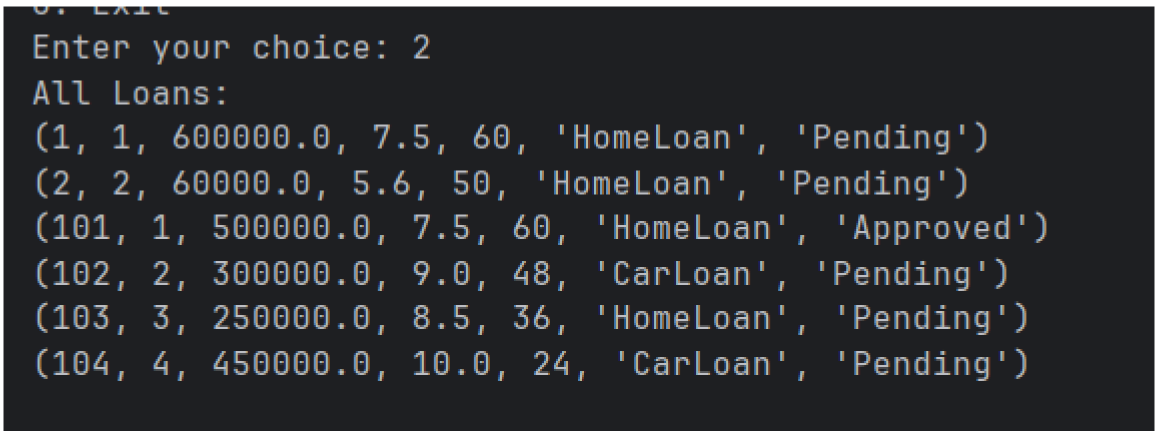
Output

Apply for Loan



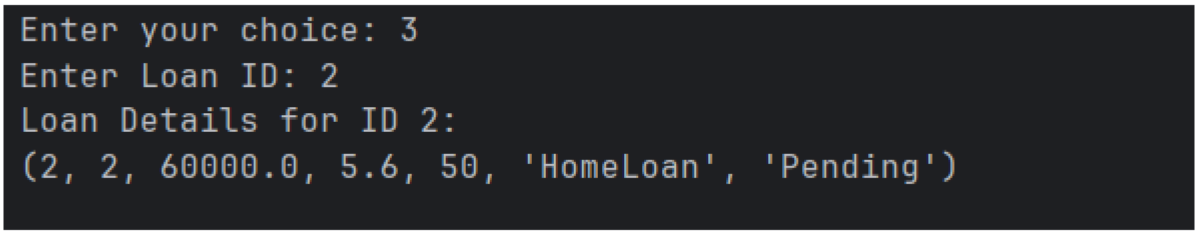
```
==== Loan Management System ====
1. Apply for a Loan
2. View All Loans
3. View Loan by ID
4. Calculate Interest
5. Calculate EMI
6. Make Loan Repayment
7. Update Loan Status
0. Exit
Enter your choice: 1
Enter Loan ID (must be unique): 2
Enter Customer ID: 2
Enter Principal Amount: 60000
Enter Interest Rate: 5.6
Enter Loan Term (in months): 50
Enter Loan Type (HomeLoan/CarLoan): HomeLoan
Do you want to apply for the loan? (Yes/No): Yes
Loan application submitted successfully.
```

View ALL loans



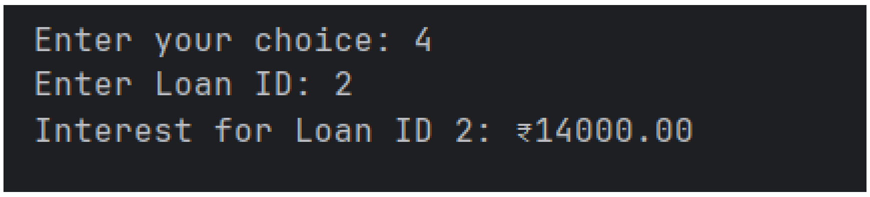
```
Enter your choice: 2
All Loans:
(1, 1, 600000.0, 7.5, 60, 'HomeLoan', 'Pending')
(2, 2, 60000.0, 5.6, 50, 'HomeLoan', 'Pending')
(101, 1, 500000.0, 7.5, 60, 'HomeLoan', 'Approved')
(102, 2, 300000.0, 9.0, 48, 'CarLoan', 'Pending')
(103, 3, 250000.0, 8.5, 36, 'HomeLoan', 'Pending')
(104, 4, 450000.0, 10.0, 24, 'CarLoan', 'Pending')
```

View Loan By ID



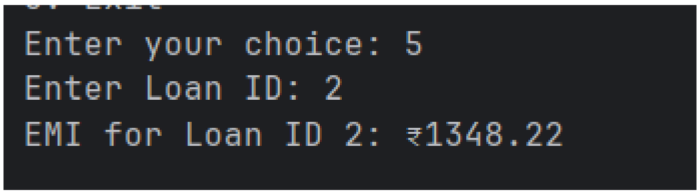
```
Enter your choice: 3
Enter Loan ID: 2
Loan Details for ID 2:
(2, 2, 60000.0, 5.6, 50, 'HomeLoan', 'Pending')
```

Calculate Interest



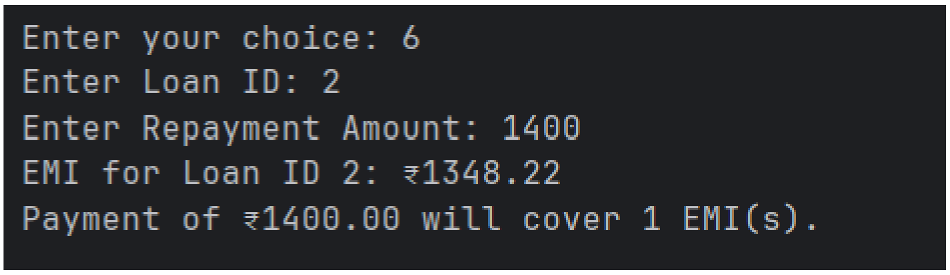
```
Enter your choice: 4
Enter Loan ID: 2
Interest for Loan ID 2: ₹14000.00
```

Calculate EMI



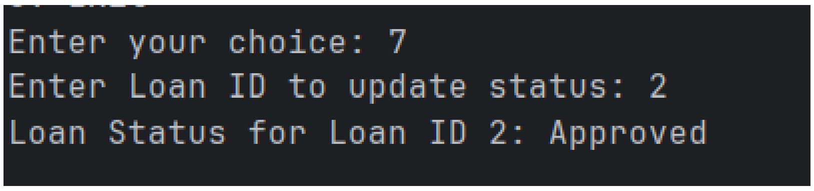
```
Enter your choice: 5
Enter Loan ID: 2
EMI for Loan ID 2: ₹1348.22
```

Make Loan Payment



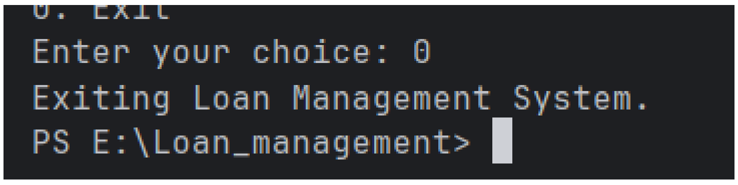
```
Enter your choice: 6
Enter Loan ID: 2
Enter Repayment Amount: 1400
EMI for Loan ID 2: ₹1348.22
Payment of ₹1400.00 will cover 1 EMI(s).
```

Update Loan Status



```
Enter your choice: 7
Enter Loan ID to update status: 2
Loan Status for Loan ID 2: Approved
```

Exit



```
0. Exit
Enter your choice: 0
Exiting Loan Management System.
PS E:\Loan_management>
```