PYTHON ASSIGNMENT

BANKING SYSTEM

LOAN ELIGIBILITY:

```python
def check_loan_eligibility(credit_score, annual_income):
    if credit_score > 700 and annual_income >= 50000:
        return "Congratulations! You are eligible for a loan."
    else:
        return "Sorry, you are not eligible for a loan."

def main():
    credit_score = int(input("Enter your credit score: "))
    annual_income = float(input("Enter your annual income: ₹"))

    result = check_loan_eligibility(credit_score, annual_income)
    print(result)

if __name__ == "__main__":
    main()



def check_balance(balance):
    print("Your current balance is: ₹{}".format(balance))

def withdraw(balance, amount):
    if amount > balance:
        print("Insufficient funds. Please enter a valid amount.")
    elif amount % 100 != 0:
        print("Withdrawal amount must be in multiples of 100 or 500.")
    else:
        balance -= amount
        print("Withdrawal successful. Remaining balance: ₹{}".format(balance))
    return balance

def deposit(balance, amount):
    balance += amount
    print("Deposit successful. Current balance: ₹{}".format(balance))
    return balance

def main():
    balance = float(input("Enter your current balance: ₹"))

    while True:
        print("\nOptions:")
        print("1. Check Balance")
        print("2. Withdraw")
        print("3. Deposit")
        print("4. Exit")

        choice = input("Enter your choice (1/2/3/4): ")

        if choice == '1':
            check_balance(balance)
        elif choice == '2':
            amount = float(input("Enter the amount to withdraw:₹ "))
            balance = withdraw(balance, amount)
        elif choice == '3':
            amount = float(input("Enter the amount to deposit:₹ "))
            balance = deposit(balance, amount)
        elif choice == '4':
            print("Thank you for using the ATM. Have a nice day!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

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

FUTURE BALANCE:

```python
def calculate_future_balance(Cbalance, interest_rate, years):

    interest_rate_decimal = interest_rate / 100

    future_balance = Cbalance * (1 + interest_rate_decimal) ** years
    return future_balance


def main():
    Cbalance = float(input("Enter the Current amount (₹): "))
    interest_rate = float(input("Enter the annual interest rate (%): "))
    years = int(input("Enter the number of years: "))


    future_balance = calculate_future_balance(Cbalance, interest_rate, years)

    print("Future balance after {} years: ₹{:.2f}".format(years, future_balance))


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

PASSWORD VALIDATION:

```python
class Bank:
    def __init__(self):
        self.accounts = {}

    def add_account(self, account_number, balance):
        if account_number in self.accounts:
            print("Account with account number {} already exists.".format(account_number))
        else:
            self.accounts[account_number] = balance
            print("Account created successfully.")

    def check_balance(self, account_number):
        if account_number in self.accounts:
            balance = self.accounts[account_number]
            print("Account balance for account number {}: ₹{}".format(account_number, balance))
        else:
            print("Account number {} not found.".format(account_number))

def main():
    bank = Bank()

    while True:
        print("\nOptions:")
        print("1. Create Account")
        print("2. Check Balance")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ")

        if choice == '1':
            account_number = input("Enter the account number: ")
            balance = float(input("Enter the initial balance: ₹"))
            bank.add_account(account_number, balance)
        elif choice == '2':
            account_number = input("Enter the account number: ")
            bank.check_balance(account_number)
        elif choice == '3':
            print("Thank you for using the bank. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

if __name__ == "__main__":
    main()



def validate_password(password):
    if len(password) < 8:
        return False, "Password must be at least 8 characters long."
    elif not any(char.isupper() for char in password):
        return False, "Password must contain at least one uppercase letter."
    elif not any(char.isdigit() for char in password):
        return False, "Password must contain at least one digit."
    else:
        return True, "Password created successfully."

def main():
    account_number = input("Enter your account number: ")
    password = input("Create a password for your bank account: ")

    is_valid, message = validate_password(password)

    if is_valid:
        print(message)
    else:
        print("Invalid Password:", message)

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

BANK TRANSACTION:

```python
class BankTransaction:
    def __init__(self):
        self.transactions = []

    def add_transaction(self, transaction_type, amount):
        self.transactions.append((transaction_type, amount))

    def display_transaction_history(self):
        print("\nTransaction History:")
        for idx, transaction in enumerate(self.transactions, start=1):
            print("{}. {} ₹{}".format(idx, transaction[0], transaction[1]))

def main():
    bank_transaction = BankTransaction()

    account_number = input("Enter your account number: ")

    while True:
        print("\nOptions:")
        print("1. Add Deposit")
        print("2. Add Withdrawal")
        print("3. Display Transaction History")
        print("4. Exit")

        choice = input("Enter your choice (1/2/3/4): ")

        if choice == '1':
            amount = float(input("Enter the deposit amount: "))
            bank_transaction.add_transaction("Deposit", amount)
        elif choice == '2':
            amount = float(input("Enter the withdrawal amount: "))
            bank_transaction.add_transaction("Withdrawal", amount)
        elif choice == '3':
            bank_transaction.display_transaction_history()
        elif choice == '4':
            print("Thank you")
            break
        else:
            print("Invalid choice. Please enter a valid option.")

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

DAO PACKAGES:

```python
from abc import ABC, abstractmethod
import re

class Customer:
    def __init__(self, customer_id="", first_name="", last_name="", email="", 
phone_number="", address=""):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone_number = phone_number
        self.address = address
    def get_customer_id(self):
        return self.customer_id

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

    def get_first_name(self):
        return self.first_name

    def set_first_name(self, first_name):
        self.first_name = first_name

    def get_last_name(self):
        return self.last_name

    def set_last_name(self, last_name):
        self.last_name = last_name

    def get_email(self):
        return self.email

    def set_email(self, email):
        if re.match(r"[^@]+@[^@]+\.[^@]+", email):
            self.email = email
        else:
            print("Invalid email address")

    def get_phone_number(self):
        return self.phone_number

    def set_phone_number(self, phone_number):
        if re.match(r"^\d{10}$", phone_number):
            self.phone_number = phone_number
        else:
            print("Invalid phone number")

    def get_address(self):
        return self.address

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

    def __str__(self):
        return f"Customer ID: {self.customer_id}, Name: {self.first_name}
{self.last_name}, Email: {self.email}, Phone: {self.phone_number}, Address:
{self.address}"


class Account(ABC):
    last_acc_no = 1000  # Static variable to generate account numbers

    def __init__(self, account_type, initial_balance, customer):
        self.account_no = Account.generate_account_number()
        self.account_type = account_type
        self.account_balance = initial_balance
        self.customer = customer

    @staticmethod
    def generate_account_number():
        Account.last_acc_no += 1
        return Account.last_acc_no

    @abstractmethod
    def withdraw(self, amount):
        pass
    @abstractmethod
    def deposit(self, amount):
        pass

    @abstractmethod
    def get_account_details(self):
        pass

    @abstractmethod
    def get_balance(self):
        pass


class SavingsAccount(Account):
    def __init__(self, initial_balance, customer, interest_rate=0.05):
        super().__init__("Savings", initial_balance, customer)
        self.minimum_balance = 500
        self.interest_rate = interest_rate

    def withdraw(self, amount):
        if self.account_balance - amount < self.minimum_balance:
            print("Withdrawal failed: Insufficient balance!")
        else:
            self.account_balance -= amount
            print("Withdrawal successful")
        return self.account_balance

    def deposit(self, amount):
        self.account_balance += amount
        return self.account_balance

    def get_account_details(self):
        return f"Account Number: {self.account_no}, Account Type: {self.account_type},
Balance: {self.account_balance}, Customer: {self.customer.first_name}
{self.customer.last_name}"

    def get_balance(self):
        return self.account_balance


class CurrentAccount(Account):
    def __init__(self, initial_balance, customer, overdraft_limit=1000):
        super().__init__("Current", initial_balance, customer)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if self.account_balance + self.overdraft_limit < amount:
            print("Withdrawal failed: Exceeds overdraft limit!")
        else:
            self.account_balance -= amount
            print("Withdrawal successful")
        return self.account_balance

    def deposit(self, amount):
        self.account_balance += amount
        return self.account_balance

    def get_account_details(self):
        return f"Account Number: {self.account_no}, Account Type: {self.account_type},
Balance: {self.account_balance}, Customer: {self.customer.first_name}
{self.customer.last_name}"

    def get_balance(self):
        return self.account_balance


class ZeroBalanceAccount(Account):
    def __init__(self, customer):
        super().__init__("Zero Balance", 0, customer)

    def withdraw(self, amount):
        print("Withdrawal failed: Account has zero balance!")
        return self.account_balance

    def deposit(self, amount):
        self.account_balance += amount
        return self.account_balance

    def get_account_details(self):
        return f"Account Number: {self.account_no}, Account Type: {self.account_type},
Balance: {self.account_balance}, Customer: {self.customer.first_name}
{self.customer.last_name}"

    def get_balance(self):
        return self.account_balance


class ICustomerServiceProvider(ABC):
    @abstractmethod
    def get_account_balance(self, account_number):
        pass

    @abstractmethod
    def deposit(self, account_number, amount):
        pass

    @abstractmethod
    def withdraw(self, account_number, amount):
        pass

    @abstractmethod
    def transfer(self, from_account_number, to_account_number, amount):
        pass

    @abstractmethod
    def get_account_details(self, account_number):
        pass


class IBankServiceProvider(ICustomerServiceProvider):
    @abstractmethod
    def create_account(self, customer, account_type, initial_balance):
        pass

    @abstractmethod
    def list_accounts(self):
        pass

    @abstractmethod
    def calculate_interest(self):
        pass


class CustomerServiceProviderImpl(ICustomerServiceProvider):
    def __init__(self):
        self.account_dict = {}

    def get_account_balance(self, account_number):
        if account_number in self.account_dict:
            return self.account_dict[account_number].get_balance()
        else:
            return "Account not found!"

    def deposit(self, account_number, amount):
        if account_number in self.account_dict:
            return self.account_dict[account_number].deposit(amount)
        else:
            return "Account not found!"

    def withdraw(self, account_number, amount):
        if account_number in self.account_dict:
            return self.account_dict[account_number].withdraw(amount)
        else:
            return "Account not found!"

    def transfer(self, from_account_number, to_account_number, amount):
        if from_account_number in self.account_dict and to_account_number in
self.account_dict:
            self.account_dict[from_account_number].withdraw(amount)
            self.account_dict[to_account_number].deposit(amount)
            return "Transfer successful!"
        else:
            return "Account not found!"

    def get_account_details(self, account_number):
        if account_number in self.account_dict:
            return self.account_dict[account_number].get_account_details()
        else:
            return "Account not found!"


class BankServiceProviderImpl(CustomerServiceProviderImpl, IBankServiceProvider):
    def __init__(self, branch_name, branch_address):
        super().__init__()
        self.account_list = []
        self.branch_name = branch_name
        self.branch_address = branch_address

    def create_account(self, customer, account_type, initial_balance):
        if account_type == "Savings":
            new_account = SavingsAccount(initial_balance, customer)
        elif account_type == "Current":
            new_account = CurrentAccount(initial_balance, customer)
        elif account_type == "Zero Balance":
            new_account = ZeroBalanceAccount(customer)
        else:
            return "Invalid account type!"

        self.account_dict[new_account.account_no] = new_account
        self.account_list.append(new_account)
        return f"Account created successfully with account number
{new_account.account_no}"

    def list_accounts(self):
        return self.account_list

    def calculate_interest(self):
        for account in self.account_list:
            if isinstance(account, SavingsAccount):
                interest = account.account_balance * account.interest_rate
                account.account_balance += interest
        return "Interest calculated and added to accounts!"


class BankApp:
    def __init__(self):
        self.bank_service_provider = BankServiceProviderImpl("MyBank", "123 Main St")

    def display_menu(self):
        print("\nWelcome to", self.bank_service_provider.branch_name)
        print("1. Create Account")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Transfer")
        print("5. Get Account Balance")
        print("6. List Accounts")
        print("7. Exit")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")
            if choice == "1":
                print("\nCreate Account")
                customer_id = input("Enter Customer ID: ")
                first_name = input("Enter First Name: ")
                last_name = input("Enter Last Name: ")
                email = input("Enter Email: ")
                customer = Customer(customer_id, first_name, last_name, email)

                print("Select Account Type:")
                print("1. Savings")
                print("2. Current")
                print("3. Zero Balance")
                account_type_choice = input("Enter your choice: ")

                if account_type_choice == "1":
                    initial_balance = float(input("Enter Initial Balance: "))
                    message = self.bank_service_provider.create_account(customer,
"Savings", initial_balance)
                    print(message)
                elif account_type_choice == "2":
                    initial_balance = float(input("Enter Initial Balance: "))
                    message = self.bank_service_provider.create_account(customer,
"Current", initial_balance)
                    print(message)
                elif account_type_choice == "3":
                    message = self.bank_service_provider.create_account(customer, "Zero
Balance", 0)
                    print(message)
                else:
                    print("Invalid choice!")

            elif choice == "2":
                print("\nDeposit")
                account_number = int(input("Enter Account Number: "))
                amount = float(input("Enter Amount to Deposit: "))
                balance = self.bank_service_provider.deposit(account_number, amount)
                print(f"New Balance: {balance}")

            elif choice == "3":
                print("\nWithdraw")
                account_number = int(input("Enter Account Number: "))
                amount = float(input("Enter Amount to Withdraw: "))
                balance = self.bank_service_provider.withdraw(account_number, amount)
                print(f"New Balance: {balance}")

            elif choice == "4":
                print("\nTransfer")
                from_account_number = int(input("Enter From Account Number: "))
                to_account_number = int(input("Enter To Account Number: "))
                amount = float(input("Enter Amount to Transfer: "))
                message = self.bank_service_provider.transfer(from_account_number,
to_account_number, amount)
                print(message)

            elif choice == "5":
                print("\nGet Account Balance")
                account_number = int(input("Enter Account Number: "))
                balance =
self.bank_service_provider.get_account_balance(account_number)
                print(f"Current Balance: {balance}")

            elif choice == "6":
                print("\nList Accounts")
                accounts = self.bank_service_provider.list_accounts()
                for account in accounts:
                    print(account.get_account_details())

            elif choice == "7":
                print("Exiting program...")
                break

            else:
                print("Invalid choice! Please enter a valid option.")



if __name__ == "__main__":
    bank_app = BankApp()
    bank_app.run()
```

EXCEPTION PACKAGES:

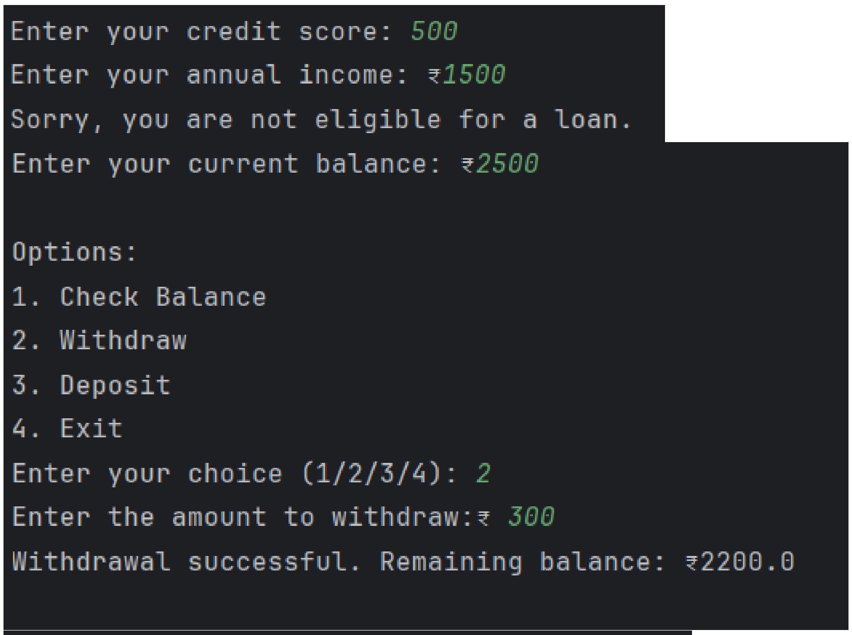
```
class InsufficientFundException(Exception):
    pass

class InvalidAccountException(Exception):
    pass

class OverDraftLimitExceededException(Exception):
    pass

class HMBank:
    def __init__(self):
        self.accounts = {}

    def withdraw(self, account_number, amount):
        if account_number not in self.accounts:
            raise InvalidAccountException("Invalid account number")

        account = self.accounts[account_number]
        if account.get_balance() < amount:
            raise InsufficientFundException("Insufficient funds in the account")

        if account.get_account_type() == "Current":
            # Check overdraft limit
            if amount > account.get_balance() + account.get_overdraft_limit():
                raise OverDraftLimitExceededException("Withdrawal amount exceeds
overdraft limit")

        account.withdraw(amount)
        return account.get_balance()

    def transfer(self, from_account_number, to_account_number, amount):
        if from_account_number not in self.accounts or to_account_number not in
self.accounts:
            raise InvalidAccountException("Invalid account number(s)")

        from_account = self.accounts[from_account_number]
        to_account = self.accounts[to_account_number]
        if from_account.get_balance() < amount:
            raise InsufficientFundException("Insufficient funds in the account")

        from_account.withdraw(amount)
        to_account.deposit(amount)
        return from_account.get_balance()


if __name__ == "__main__":
    bank = HMBank()

    try:
        account_number = "1234567890"
        amount = 1000
        new_balance = bank.withdraw(account_number, amount)
        print(f"Withdrawal successful. New balance: {new_balance}")

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

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

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

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

DATABASE CONNECTIVITY:

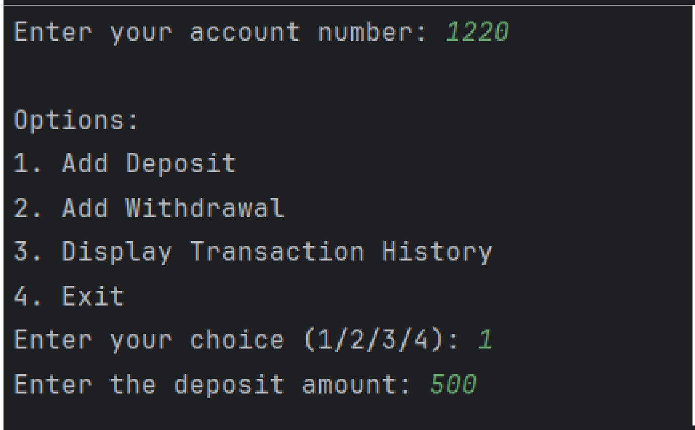
```
import mysql.connector

from mysql.connector import Error

class Customer:
    def __init__(self, customer_id, first_name, last_name, email, phone_number,
address):
        self.customer_id = customer_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone_number = phone_number
        self.address = address

    def __str__(self):
        return f"Customer ID: {self.customer_id}\n" \
               f"First Name: {self.first_name}\n" \
               f"Last Name: {self.last_name}\n" \
               f"Email: {self.email}\n" \
               f"Phone Number: {self.phone_number}\n" \
               f"Address: {self.address}"

    def customer_info(self):
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="HARSHA1@singh",
            port="3306",
            database="HMBank"

        )
        cursor = con.cursor()

        cursor.execute("CREATE TABLE IF NOT EXISTS Customer ("
                       "customer_id INT AUTO_INCREMENT PRIMARY KEY,"
                       "first_name VARCHAR(255),"
                       "last_name VARCHAR(255),"
                       "email VARCHAR(255),"
                       "phone_number VARCHAR(15),"
                       "address VARCHAR(255)"
                       ")")
        con.commit()
        cursor.close()
        con.close()


class Account:
    last_account_number = 1000

    def __init__(self, account_type, balance, customer):
        Account.last_account_number += 1
        self.account_number = Account.last_account_number
        self.account_type = account_type
        self.balance = balance
        self.customer = customer

    def account_info(self):
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="HARSHA1@singh",
            port="3306",
            database="HMBank"

        )
        cursor = con.cursor()
        cursor.execute("CREATE TABLE IF NOT EXISTS Account ("
                       "account_number INT AUTO_INCREMENT PRIMARY KEY,"
                       "account_type VARCHAR(255),"
                       "account_balance DECIMAL(10, 2),"
                       "customer_id INT,"
                       "FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)"
                       ")")

class SavingsAccount(Account):
    min_balance = 500

    def __init__(self, balance, customer, interest_rate):
        super().__init__("Savings", balance, customer)
        self.interest_rate = interest_rate


class CurrentAccount(Account):
    def __init__(self, balance, customer, overdraft_limit):
        super().__init__("Current", balance, customer)
        self.overdraft_limit = overdraft_limit


class ZeroBalanceAccount(Account):
    def __init__(self, customer):
        super().__init__("ZeroBalance", 0, customer)


# bean/transaction.py
class Transaction:
    def __init__(self, account, description, date_time, transaction_type,
transaction_amount):
        self.account = account
        self.description = description
        self.date_time = date_time
        self.transaction_type = transaction_type
```

```python
        self.transaction_amount = transaction_amount

    def transactions_info(self):
        con = mysql.connector.connect(
            host="localhost",
            user="root",
            password="HARSHA1@singh",
            port="3306",
            database="HMBank"

        )
        cursor = con.cursor()
        cursor.execute("CREATE TABLE IF NOT EXISTS TRANSACTION ("
                       "transaction_id INT AUTO_INCREMENT PRIMARY KEY,"
                       "account_number INT,"
                       "description TEXT,"
                       "date_time DATETIME,"
                       "transaction_type ENUM('Withdraw', 'Deposit', 'Transfer'),"
                       "transaction_amount DECIMAL(10, 2),"
                       "FOREIGN KEY (account_number) REFERENCES
Account(account_number)"
                       ")")

# service/icustomerserviceprovider.py
from abc import ABC, abstractmethod

class ICustomerServiceProvider(ABC):
    @abstractmethod
    def get_account_balance(self, account_number):
        pass

    @abstractmethod
    def deposit(self, account_number, amount):
        pass

    @abstractmethod
    def withdraw(self, account_number, amount):
        pass

    @abstractmethod
    def transfer(self, from_account_number, to_account_number, amount):
        pass

    @abstractmethod
    def get_account_details(self, account_number):
        pass

    @abstractmethod
    def get_transactions(self, account_number, from_date, to_date):
        pass
```

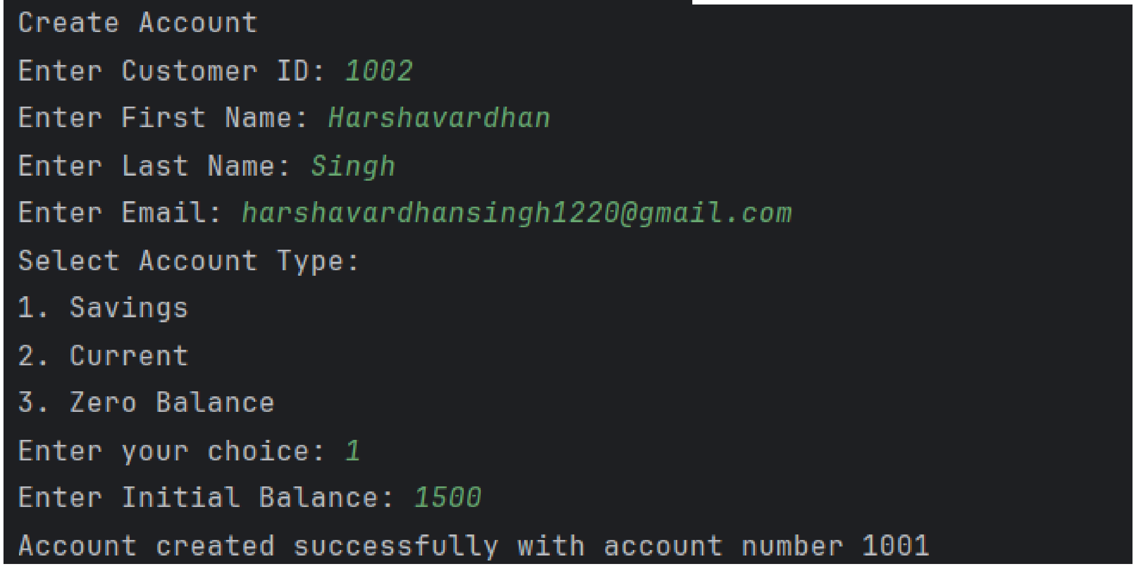
OUTPUT:



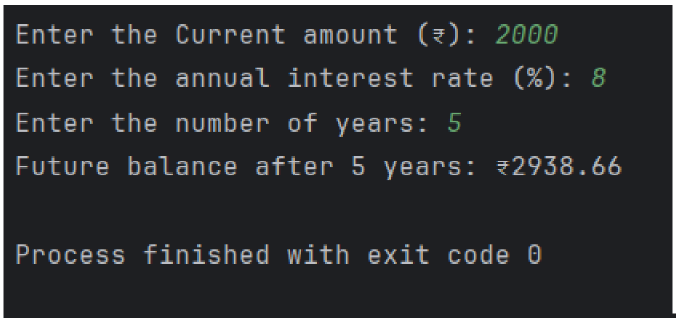
```
Enter your credit score: 500
Enter your annual income: ₹1500
Sorry, you are not eligible for a loan.
Enter your current balance: ₹2500

Options:
1. Check Balance
2. Withdraw
3. Deposit
4. Exit
Enter your choice (1/2/3/4): 2
Enter the amount to withdraw:₹ 300
Withdrawal successful. Remaining balance: ₹2200.0
```



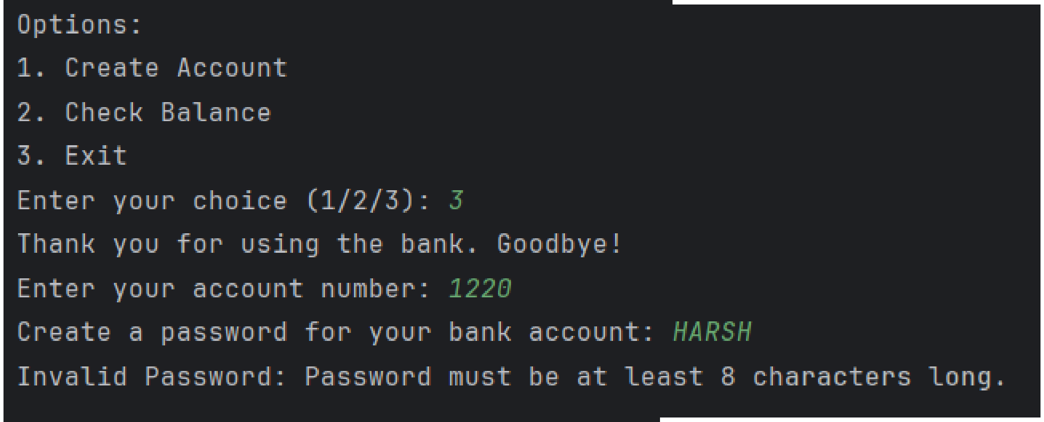
```
Enter your account number: 1220

Options:
1. Add Deposit
2. Add Withdrawal
3. Display Transaction History
4. Exit
Enter your choice (1/2/3/4): 1
Enter the deposit amount: 500
```



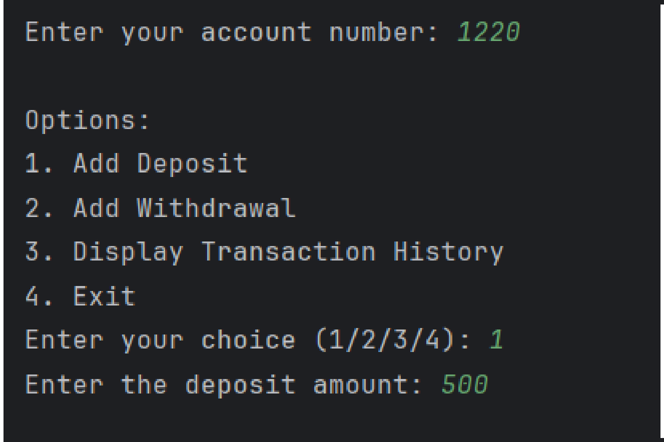
```
Create Account
Enter Customer ID: 1002
Enter First Name: Harshavardhan
Enter Last Name: Singh
Enter Email: harshavardhansingh1220@gmail.com
Select Account Type:
1. Savings
2. Current
3. Zero Balance
Enter your choice: 1
Enter Initial Balance: 1500
Account created successfully with account number 1001
```



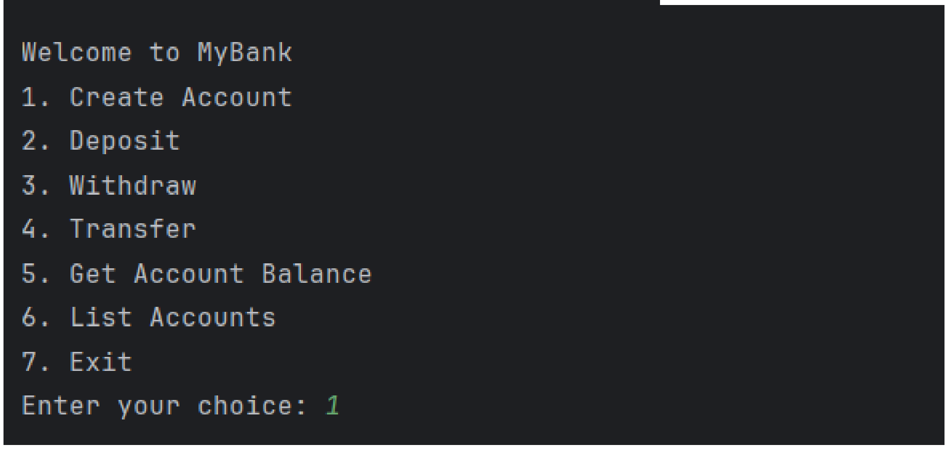
```
Enter the Current amount (₹): 2000
Enter the annual interest rate (%): 8
Enter the number of years: 5
Future balance after 5 years: ₹2938.66

Process finished with exit code 0
```



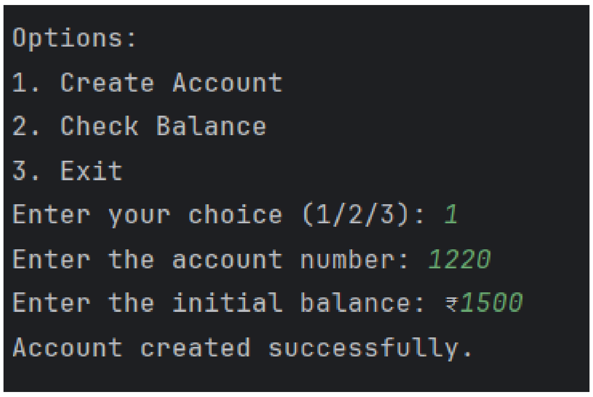
```
Options:
1. Create Account
2. Check Balance
3. Exit
Enter your choice (1/2/3): 3
Thank you for using the bank. Goodbye!
Enter your account number: 1220
Create a password for your bank account: HARSH
Invalid Password: Password must be at least 8 characters long.
```



```
Enter your account number: 1220

Options:
1. Add Deposit
2. Add Withdrawal
3. Display Transaction History
4. Exit
Enter your choice (1/2/3/4): 1
Enter the deposit amount: 500
```



```
Welcome to MyBank
1. Create Account
2. Deposit
3. Withdraw
4. Transfer
5. Get Account Balance
6. List Accounts
7. Exit
Enter your choice: 1
```



```
Options:
1. Create Account
2. Check Balance
3. Exit
Enter your choice (1/2/3): 1
Enter the account number: 1220
Enter the initial balance: ₹1500
Account created successfully.
```