TASK:7

1. Create a `Customer` class with the following confidential attributes:

• Attributes

o Customer ID

o First Name

o Last Name

o Email Address

o Phone Number

o Address

• Constructor and Methods

o Implement default constructors and overload the constructor with Customer attributes, generate getter and setter, (print all information of attribute) methods for the attributes.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace practice.entity
{
    public class Customer
    {
        private int customerId;
        private string firstName;
        private string lastName;
        private string email;
        private string phoneNumber;
        private string address;

        public Customer() { }

        public Customer(int customerId, string firstName, string lastName,
string email, string phoneNumber, string address)
        {
            this.customerId = customerId;
            this.firstName = firstName;
            this.lastName = lastName;
            this.email = email;
            this.phoneNumber = phoneNumber;
            this.address = address;
        }

        // Getters and Setters
        public int CustomerId { get => customerId; set => customerId =
value; }
        public string FirstName { get => firstName; set => firstName =
value; }
        public string LastName { get => lastName; set => lastName =
value; }
        public string Email { get => email; set => email = value; }
        public string PhoneNumber { get => phoneNumber; set => phoneNumber
= value; }
        public string Address { get => address; set => address = value; }

        public void PrintCustomerInfo()
        {
            Console.WriteLine($"Customer ID   : {customerId}");
            Console.WriteLine($"Name          : {firstName} {lastName}");
            Console.WriteLine($"Email         : {email}");
            Console.WriteLine($"Phone         : {phoneNumber}");
```

```
            Console.WriteLine($"Address       : {address}");
        }
    }
}
```

Create an `Account` class with the following confidential attributes:
• Attributes
o Account Number
o Account Type (e.g., Savings, Current)
o Account Balance
• Constructor and Methods
o Implement default constructors and overload the constructor with Account attributes,
o Generate getter and setter, (print all information of attribute) methods for the attributes.
o Add methods to the `Account` class to allow deposits and withdrawals.
- deposit(amount: float): Deposit the specified amount into the account.

- withdraw(amount: float): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.
- calculate_interest(): method for calculating interest amount for the available balance. interest rate is fixed to 4.5%

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace practice.entity
{
    public class Account
    {
        private int accountNumber;
        private string accountType;
        private double accountBalance;

        public Account() { }

        public Account(int accountNumber, string accountType, double
accountBalance)
        {
            this.accountNumber = accountNumber;
            this.accountType = accountType;
            this.accountBalance = accountBalance;
        }

        public int AccountNumber { get => accountNumber; set =>
accountNumber = value; }
        public string AccountType { get => accountType; set => accountType
= value; }
        public double AccountBalance { get => accountBalance; set =>
accountBalance = value; }

        public void PrintAccountInfo()
        {
            Console.WriteLine($"Account Number: {accountNumber}");
            Console.WriteLine($"Account Type  : {accountType}");
            Console.WriteLine($"Balance       : ${accountBalance}");
        }
```

```
        public void Deposit(double amount)
        {
            if (amount > 0)
            {
                accountBalance += amount;
                Console.WriteLine($"Deposited ${amount}. New Balance:
${accountBalance}");
            }
            else
            {
                Console.WriteLine("Invalid deposit amount.");
            }
        }

        public void Withdraw(double amount)
        {
            if (amount <= 0)
            {
                Console.WriteLine("Invalid withdrawal amount.");
            }
            else if (amount > accountBalance)
            {
                Console.WriteLine("Insufficient funds.");
            }
            else
            {
                accountBalance -= amount;
                Console.WriteLine($"Withdrew ${amount}. Remaining Balance:
${accountBalance}");
            }
        }

        public void CalculateInterest()
        {
            if (accountType.ToLower() == "savings")
            {
                double interest = accountBalance * 4.5 / 100;
                accountBalance += interest;
                Console.WriteLine($"Interest of ${interest} added. New
Balance: ${accountBalance}");
            }
            else
            {
                Console.WriteLine("Interest only applies to savings
accounts.");
            }
        }
    }
}
```

Create a Bank class to represent the banking system. Perform the following operation in main method:
o create object for account class by calling parameter constructor.
o deposit(amount: float): Deposit the specified amount into the account.
o withdraw(amount: float): Withdraw the specified amount from the account.
o calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```
using practice.entity;

namespace practice
{
```

```
internal class Program
{
    static List<Customer> customers = new List<Customer>();
    static List<Account> accounts = new List<Account>();

    static void Main(string[] args)
    {
        bool running = true;

        while (running)
        {
            Console.WriteLine("\n====== HM BANK MENU ======");
            Console.WriteLine("1. Add Customer and Account");
            Console.WriteLine("2. View All Customers and Accounts");
            Console.WriteLine("3. Deposit");
            Console.WriteLine("4. Withdraw");
            Console.WriteLine("5. Calculate Interest");
            Console.WriteLine("6. Exit");
            Console.Write("Choose an option: ");

            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    AddCustomerAndAccount();
                    break;
                case "2":
                    ViewAll();
                    break;
                case "3":
                    Deposit();
                    break;
                case "4":
                    Withdraw();
                    break;
                case "5":
                    CalculateInterest();
                    break;
                case "6":
                    running = false;
                    Console.WriteLine("Thank you for using HM Bank!");
                    break;
                default:
                    Console.WriteLine("Invalid choice.");
                    break;
            }
        }
    }

    static void AddCustomerAndAccount()
    {
        try
        {
            Console.WriteLine("\n--- Enter Customer Info ---");
            Console.Write("Customer ID: ");
            int custId = int.Parse(Console.ReadLine());

            Console.Write("First Name: ");
            string fName = Console.ReadLine();

            Console.Write("Last Name: ");
            string lName = Console.ReadLine();
```

```
                Console.Write("Email: ");
                string email = Console.ReadLine();

                Console.Write("Phone Number: ");
                string phone = Console.ReadLine();

                Console.Write("Address: ");
                string address = Console.ReadLine();

                Customer cust = new Customer(custId, fName, lName, email,
phone, address);
                customers.Add(cust);

                Console.WriteLine("\n--- Enter Account Info ---");
                Console.Write("Account Number: ");
                int accNo = int.Parse(Console.ReadLine());

                Console.Write("Account Type (Savings/Current): ");
                string type = Console.ReadLine();

                Console.Write("Initial Balance: ");
                double balance = double.Parse(Console.ReadLine());

                Account acc = new Account(accNo, type, balance);
                accounts.Add(acc);

                Console.WriteLine("Customer and Account created
successfully.");
            }
            catch (Exception)
            {
                Console.WriteLine("Invalid input! Please try again.");
            }
        }

        static void ViewAll()
        {
            Console.WriteLine("\n--- All Customer and Account Info ---");
            for (int i = 0; i < customers.Count; i++)
            {
                Console.WriteLine($"\n--- Customer {i + 1} ---");
                customers[i].PrintCustomerInfo();
                accounts[i].PrintAccountInfo();
            }
        }

        static void Deposit()
        {
            try
            {
                Console.Write("Enter Account Number to Deposit: ");
                int accNo = int.Parse(Console.ReadLine());

                Account acc = accounts.Find(a => a.AccountNumber == accNo);
                if (acc != null)
                {
                    Console.Write("Enter Amount to Deposit: ");
                    double amount = double.Parse(Console.ReadLine());
                    acc.Deposit(amount);
                }
                else
                {
```

```
                    Console.WriteLine("Account not found.");
                }
            }
            catch (Exception)
            {
                Console.WriteLine("Error occurred during deposit.");
            }
        }

        static void Withdraw()
        {
            try
            {
                Console.Write("Enter Account Number to Withdraw: ");
                int accNo = int.Parse(Console.ReadLine());

                Account acc = accounts.Find(a => a.AccountNumber == accNo);
                if (acc != null)
                {
                    Console.Write("Enter Amount to Withdraw: ");
                    double amount = double.Parse(Console.ReadLine());
                    acc.Withdraw(amount);
                }
                else
                {
                    Console.WriteLine("Account not found.");
                }
            }
            catch (Exception)
            {
                Console.WriteLine("Error occurred during withdrawal.");
            }
        }

        static void CalculateInterest()
        {
            try
            {
                Console.Write("Enter Account Number to Calculate Interest:
");
                int accNo = int.Parse(Console.ReadLine());

                Account acc = accounts.Find(a => a.AccountNumber == accNo);
                if (acc != null)
                {
                    acc.CalculateInterest();
                }
                else
                {
                    Console.WriteLine("Account not found.");
                }
            }
            catch (Exception)
            {
                Console.WriteLine("Error calculating interest.");
            }
        }

    }
}
```