Task 10: Has A Relation / Association
1. Create a `Customer` class with the following attributes:
• Customer ID
• First Name
• Last Name
• Email Address (validate with valid email address)
• Phone Number (Validate 10-digit phone number)
• Address
• Methods and Constructor:
o Implement default constructors and overload the constructor with Account attributes, generate getter, setter, print all information of attribute) methods for the attributes.

```
using System;
using System.Net.Mail;

namespace task10.entity
{
    public class Customer
    {
        public long CustomerID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        private string emailAddress;
        public string EmailAddress
        {
            get => emailAddress;
            set
            {
                try
                {
                    var addr = new MailAddress(value);
                    emailAddress = value;
                }
                catch
                {
                    throw new ArgumentException("Invalid email address.");
                }
            }
        }

        private string phoneNumber;
        public string PhoneNumber
        {
            get => phoneNumber;
            set
            {
                if (value.Length == 10 && long.TryParse(value, out _))
                    phoneNumber = value;
                else
                    throw new ArgumentException("Phone number must be 10
digits.");
            }
        }

        public string Address { get; set; }

        public Customer() { }

        public Customer(long id, string fname, string lname, string email,
string phone, string address)
```

```
        {
            CustomerID = id;
            FirstName = fname;
            LastName = lname;
            EmailAddress = email;
            PhoneNumber = phone;
            Address = address;
        }

        public void PrintCustomerInfo()
        {
            Console.WriteLine($"Customer ID: {CustomerID}, Name:
{FirstName} {LastName}");
            Console.WriteLine($"Email: {EmailAddress}, Phone:
{PhoneNumber}");
            Console.WriteLine($"Address: {Address}");
        }
    }
}
```

2. Create an `Account` class with the following attributes:
• Account Number (a unique identifier).
• Account Type (e.g., Savings, Current)
• Account Balance
• Customer (the customer who owns the account)
• Methods and Constructor:
o Implement default constructors and overload the constructor with Account attributes, generate getter, setter, (print all information of attribute) methods for the attributes.

Create a Bank Class and must have following requirements:
1. Create a Bank class to represent the banking system. It should have the following methods:
• create_account(Customer customer, long accNo, String accType, float balance): Create a new bank account for the given customer with the initial balance.
• get_account_balance(account_number: long): Retrieve the balance of an account given its account number. should return the current balance of account.
• deposit(account_number: long, amount: float): Deposit the specified amount into the account. Should return the current balance of account.
• withdraw(account_number: long, amount: float): Withdraw the specified amount from the account. Should return the current balance of account.
• transfer(from_account_number: long, to_account_number: int, amount: float): Transfer money from one account to another.
• getAccountDetails(account_number: long): Should return the account and customer details.

2. Ensure that account numbers are automatically generated when an account is created, starting from 1001 and incrementing for each new account.

```
using System;
using task10.entity;

namespace task10.entity
{
    public class Account
    {
        public long AccountNumber { get; set; }
        public string AccountType { get; set; }
        public float Balance { get; set; }
        public Customer Customer { get; set; }

        public Account() { }
```

```
        public Account(long accNo, string accType, float balance, Customer
customer)
        {
            AccountNumber = accNo;
            AccountType = accType;
            Balance = balance;
            Customer = customer;
        }

        public void PrintAccountInfo()
        {
            Console.WriteLine($"\nAccount No: {AccountNumber}, Type:
{AccountType}, Balance: {Balance:C}");
            Customer.PrintCustomerInfo();
        }
    }
}
```

----------**BANK.CS**

```
using System;
using System.Collections.Generic;
using task10.entity;

namespace task10.service
{
    public class Bank
    {
        private List<Account> accounts = new List<Account>();
        private long nextAccountNumber = 1001;

        public void CreateAccount(Customer customer, string accType, float
initialBalance)
        {
            Account acc = new Account(nextAccountNumber++, accType,
initialBalance, customer);
            accounts.Add(acc);
            Console.WriteLine("Account created successfully!\n");
            acc.PrintAccountInfo();
        }

        public float GetAccountBalance(long accNo)
        {
            var acc = accounts.Find(a => a.AccountNumber == accNo);
            return acc != null ? acc.Balance : throw new
Exception("Account not found.");
        }

        public float Deposit(long accNo, float amount)
        {
            var acc = accounts.Find(a => a.AccountNumber == accNo);
            if (acc == null) throw new Exception("Account not found.");
            acc.Balance += amount;
            return acc.Balance;
        }

        public float Withdraw(long accNo, float amount)
        {
            var acc = accounts.Find(a => a.AccountNumber == accNo);
            if (acc == null) throw new Exception("Account not found.");
            if (acc.Balance >= amount)
                acc.Balance -= amount;
            else
                throw new Exception("Insufficient balance.");
```

```
            return acc.Balance;
        }

        public void Transfer(long fromAcc, long toAcc, float amount)
        {
            var from = accounts.Find(a => a.AccountNumber == fromAcc);
            var to = accounts.Find(a => a.AccountNumber == toAcc);

            if (from == null || to == null) throw new Exception("Account
not found.");
            if (from.Balance < amount) throw new Exception("Insufficient
balance.");

            from.Balance -= amount;
            to.Balance += amount;
            Console.WriteLine("Transfer successful!");
        }

        public void GetAccountDetails(long accNo)
        {
            var acc = accounts.Find(a => a.AccountNumber == accNo);
            if (acc == null)
                Console.WriteLine("Account not found.");
            else
                acc.PrintAccountInfo();
        }
    }
}
```

3. Create a BankApp class with a main method to simulate the banking system. Allow the user to interact with the system by entering commands such as "create_account", "deposit", "withdraw", "get_balance", "transfer", "getAccountDetails" and "exit." create_account should display sub menu to choose type of accounts and repeat this operation until user exit.

```
using System;
using task10.entity;
using task10.service;

class MainModule
{
    static void Main()
    {
        Bank bank = new Bank();
        bool running = true;

        while (running)
        {
            Console.WriteLine("\n====== HM BANK MENU ======");
            Console.WriteLine("1. Create Account");
            Console.WriteLine("2. Deposit");
            Console.WriteLine("3. Withdraw");
            Console.WriteLine("4. Get Balance");
            Console.WriteLine("5. Transfer");
            Console.WriteLine("6. Get Account Details");
            Console.WriteLine("7. Exit");
            Console.Write("Choose an option: ");
            string choice = Console.ReadLine();

            try
            {
                switch (choice)
                {
                    case "1":
                        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: ");
                        string phone = Console.ReadLine();
                        Console.Write("Address: ");
                        string address = Console.ReadLine();
                        Console.Write("Account Type (Savings/Current): ");
                        string accType = Console.ReadLine();
                        Console.Write("Initial Balance: ");
                        float balance = float.Parse(Console.ReadLine());

                        Customer cust = new Customer(DateTime.Now.Ticks,
fname, lname, email, phone, address);
                        bank.CreateAccount(cust, accType, balance);
                        break;

                    case "2":
                        Console.Write("Account Number: ");
                        long depAcc = long.Parse(Console.ReadLine());
                        Console.Write("Amount to Deposit: ");
                        float depAmt = float.Parse(Console.ReadLine());
                        float newBalDep = bank.Deposit(depAcc, depAmt);
                        Console.WriteLine($"New Balance: {newBalDep:C}");
                        break;

                    case "3":
                        Console.Write("Account Number: ");
                        long withAcc = long.Parse(Console.ReadLine());
                        Console.Write("Amount to Withdraw: ");
                        float withAmt = float.Parse(Console.ReadLine());
                        float newBalWith = bank.Withdraw(withAcc, withAmt);
                        Console.WriteLine($"New Balance: {newBalWith:C}");
                        break;

                    case "4":
                        Console.Write("Account Number: ");
                        long balAcc = long.Parse(Console.ReadLine());
                        Console.WriteLine($"Current Balance:
{bank.GetAccountBalance(balAcc):C}");
                        break;

                    case "5":
                        Console.Write("From Account: ");
                        long from = long.Parse(Console.ReadLine());
                        Console.Write("To Account: ");
                        long to = long.Parse(Console.ReadLine());
                        Console.Write("Amount: ");
                        float amt = float.Parse(Console.ReadLine());
                        bank.Transfer(from, to, amt);
                        break;

                    case "6":
                        Console.Write("Account Number: ");
                        long detailAcc = long.Parse(Console.ReadLine());
                        bank.GetAccountDetails(detailAcc);
                        break;

                    case "7":
                        running = false;
                        break;
```

```
                    default:
                        Console.WriteLine("Invalid option.");
                        break;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error: {ex.Message}");
            }
        }
    }
}
```