Task 8: Inheritance and polymorphism

1. Overload the deposit and withdraw methods in Account class as mentioned below.

• 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.

• deposit(amount: int): Deposit the specified amount into the account.

• withdraw(amount: int): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.

• deposit(amount: double): Deposit the specified amount into the account.

• withdraw(amount: double): Withdraw the specified amount from the account. withdraw amount only if there is sufficient fund else display insufficient balance.

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

namespace task8.entity.model
{
    public class Account
    {
        public string AccountNumber { get; set; }
        public string AccountType { get; set; }
        public double Balance { get; set; }

        public Account() { }

        public Account(string accountNumber, string accountType, double
balance)
        {
            AccountNumber = accountNumber;
            AccountType = accountType;
            Balance = balance;
        }

        // Method Overloading
        public virtual void Deposit(float amount) => Balance += amount;
        public virtual void Deposit(int amount) => Balance += amount;
        public virtual void Deposit(double amount) => Balance += amount;

        public virtual void Withdraw(float amount)
        {
            if (Balance >= amount)
                Balance -= amount;
            else
                Console.WriteLine("Insufficient balance.");
        }

        public virtual void Withdraw(int amount) =>
Withdraw((float)amount);
        public virtual void Withdraw(double amount) =>
Withdraw((float)amount);

        public virtual void CalculateInterest()
        {
            double interest = Balance * 0.045;
            Balance += interest;
            Console.WriteLine($"Interest added: {interest}, New Balance:
{Balance}");
        }
```

```
        public virtual void PrintInfo()
        {
            Console.WriteLine($"\nAccount No: {AccountNumber}, Type:
{AccountType}, Balance: {Balance}");
        }
    }
}
```

2. Create Subclasses for Specific Account Types

• Create subclasses for specific account types (e.g., `SavingsAccount`, `CurrentAccount`) that inherit from the `Account` class.
o SavingsAccount: A savings account that includes an additional attribute for interest rate. override the calculate_interest() from Account class method to calculate interest based on the balance and interest rate.
o CurrentAccount: A current account that includes an additional attribute overdraftLimit. A current account with no interest. Implement the withdraw() method to allow overdraft up to a certain limit (configure a constant for the overdraft limit).

--for savings account

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

namespace task8.entity.model
{
    public class SavingsAccount : Account
    {
        private double interestRate;

        public SavingsAccount(string accNo, double balance, double
interestRate = 4.5)
            : base(accNo, "Savings", balance)
        {
            this.interestRate = interestRate;
        }

        public override void CalculateInterest()
        {
            double interest = Balance * (interestRate / 100);
            Balance += interest;
            Console.WriteLine($"[Savings] Interest added: {interest}, New
Balance: {Balance}");
        }
    }
}
```

-------currentaccount

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

namespace task8.entity.model
{
    public class CurrentAccount : Account
```

```
    {
        private const double overdraftLimit = 5000;

        public CurrentAccount(string accNo, double balance)
            : base(accNo, "Current", balance) { }

        public override void Withdraw(float amount)
        {
            if (Balance + overdraftLimit >= amount)
            {
                Balance -= amount;
                Console.WriteLine($"[Current] Withdrawn: {amount}, New
Balance: {Balance}");
            }
            else
            {
                Console.WriteLine($"[Current] Cannot exceed overdraft
limit of {overdraftLimit}. Withdrawal denied.");
            }
        }

        public override void CalculateInterest()
        {
            Console.WriteLine("[Current] No interest applied on Current
Accounts.");
        }
    }
}
```

3. Create a Bank class to represent the banking system. Perform the following operation in main method:

• Display menu for user to create object for account class by calling parameter constructor. Menu should display options `SavingsAccount` and `CurrentAccount`. user can choose any one option to create account. use switch case for implementation.

• deposit(amount: float): Deposit the specified amount into the account.

• withdraw(amount: float): Withdraw the specified amount from the account. For saving account withdraw amount only if there is sufficient fund else display insufficient balance.

For Current Account withdraw limit can exceed the available balance and should not exceed the overdraft limit.

• calculate_interest(): Calculate and add interest to the account balance for savings accounts.

```
using System;
using task8.entity.model;

namespace task8
{
    class MainModule
    {
        static void Main(string[] args)
        {
            bool running = true;
            Account account = null;

            while (running)
            {
                Console.WriteLine("\n====== HM BANK MENU ======");
                Console.WriteLine("1. Create Account");
                Console.WriteLine("2. Deposit");
                Console.WriteLine("3. Withdraw");
```

```
                Console.WriteLine("4. Calculate Interest");
                Console.WriteLine("5. Exit");
                Console.Write("Choose an option: ");
                string choice = Console.ReadLine();

                switch (choice)
                {
                    case "1":
                        Console.WriteLine("\nChoose Account Type:");
                        Console.WriteLine("1. Savings Account");
                        Console.WriteLine("2. Current Account");
                        Console.Write("Your choice: ");
                        string accType = Console.ReadLine();

                        Console.Write("Enter Account Number: ");
                        long accNo = Convert.ToInt64(Console.ReadLine());

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

                        if (accType == "1")
                        {
                            account = new SavingsAccount("accNo",balance);
                            Console.WriteLine("Savings Account Created
Successfully!");
                        }
                        else if (accType == "2")
                        {
                            account = new CurrentAccount("accNo", balance);
                            Console.WriteLine("Current Account Created
Successfully!");
                        }
                        else
                        {
                            Console.WriteLine("Invalid account type
selected.");
                        }
                        break;

                    case "2":
                        if (account == null)
                        {
                            Console.WriteLine("Create an account first.");
                            break;
                        }
                        Console.Write("Enter amount to deposit: ");
                        float depositAmount =
float.Parse(Console.ReadLine());
                        account.Deposit(depositAmount);
                        break;

                    case "3":
                        if (account == null)
                        {
                            Console.WriteLine("Create an account first.");
                            break;
                        }
                        Console.Write("Enter amount to withdraw: ");
                        float withdrawAmount =
float.Parse(Console.ReadLine());
                        account.Withdraw(withdrawAmount);
                        break;
```

```
                    case "4":
                        if (account == null)
                        {
                            Console.WriteLine("Create an account first.");
                            break;
                        }
                        account.CalculateInterest();
                        break;

                    case "5":
                        running = false;
                        Console.WriteLine("Thank you for using HM Bank!");
                        break;

                    default:
                        Console.WriteLine("Invalid choice. Please try
again.");
                        break;
                }
            }
        }
    }
}
```