Task 9: Abstraction

1. Create an abstract class BankAccount that represents a generic bank account. It should include the following attributes and methods:

• Attributes:
o Account number.
o Customer name.
o Balance.
• Constructors:
o Implement default constructors and overload the constructor with Account attributes, generate getter and setter, print all information of attribute methods for the attributes.
• Abstract methods:
o deposit(amount: float): Deposit the specified amount into the account.
o withdraw(amount: float): Withdraw the specified amount from the account (implement error handling for insufficient funds).
o calculate_interest(): Abstract method for calculating interest.

```
namespace task9.entity.model
{
    public abstract class BankAccount
    {
        public long AccountNumber { get; set; }
        public string CustomerName { get; set; }
        public double Balance { get; set; }

        public BankAccount() { }

        public BankAccount(long accountNumber, string customerName, double
balance)
        {
            AccountNumber = accountNumber;
            CustomerName = customerName;
            Balance = balance;
        }

        public abstract void Deposit(double amount);
        public abstract void Withdraw(double amount);
        public abstract void CalculateInterest();

        public virtual void PrintInfo()
        {
            Console.WriteLine($"Account Number: {AccountNumber}");
            Console.WriteLine($"Customer Name: {CustomerName}");
            Console.WriteLine($"Balance: {Balance:C}");
        }
    }
}
```

2. Create two concrete classes that inherit from BankAccount:

• SavingsAccount: A savings account that includes an additional attribute for interest rate. Implement the calculate_interest() method to calculate interest based on the balance and interest rate.
• CurrentAccount: 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).

```
namespace task9.entity.model
{
    public class SavingsAccount : BankAccount
    {
        public double InterestRate { get; set; } = 4.5;
```

```
        public SavingsAccount(long accNo, string name, double balance)
            : base(accNo, name, balance) { }

        public override void Deposit(double amount)
        {
            Balance += amount;
            Console.WriteLine($"Deposited: {amount:C} | New Balance:
{Balance:C}");
        }

        public override void Withdraw(double amount)
        {
            if (amount <= Balance)
            {
                Balance -= amount;
                Console.WriteLine($"Withdrawn: {amount:C} | New Balance:
{Balance:C}");
            }
            else
            {
                Console.WriteLine("Insufficient balance.");
            }
        }

        public override void CalculateInterest()
        {
            double interest = Balance * (InterestRate / 100);
            Balance += interest;
            Console.WriteLine($"Interest Added: {interest:C} | New Balance:
{Balance:C}");
        }
    }
}
```

---------currentSavings

```
namespace task9.entity.model
{
    public class CurrentAccount : BankAccount
    {
        public const double OverdraftLimit = 1000.0;

        public CurrentAccount(long accNo, string name, double balance)
            : base(accNo, name, balance) { }

        public override void Deposit(double amount)
        {
            Balance += amount;
            Console.WriteLine($"Deposited: {amount:C} | New Balance:
{Balance:C}");
        }

        public override void Withdraw(double amount)
        {
            if (amount <= Balance + OverdraftLimit)
            {
                Balance -= amount;
                Console.WriteLine($"Withdrawn: {amount:C} | New Balance:
{Balance:C}");
            }
            else
            {
```

```
                Console.WriteLine("Overdraft limit exceeded.");
            }
        }

        public override void CalculateInterest()
        {
            Console.WriteLine("No interest for 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. create_account should display sub menu to choose type of accounts.

o Hint: Account acc = new SavingsAccount(); or Account acc = new CurrentAccount();

• 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 System.Collections.Generic;
using task9.entity.model;

class MainModule
{
    static void Main()
    {
        List<BankAccount> accounts = new List<BankAccount>();
        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. Calculate Interest");
            Console.WriteLine("5. Exit");
            Console.Write("Choose an option: ");
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    Console.WriteLine("Choose Account Type: 1. Savings 2.
Current");
                    string type = Console.ReadLine();
                    Console.Write("Enter Account Number: ");
                    long accNo = Convert.ToInt64(Console.ReadLine());
                    Console.Write("Enter Customer Name: ");
                    string name = Console.ReadLine();
                    Console.Write("Enter Initial Balance: ");
                    double balance = Convert.ToDouble(Console.ReadLine());
```

```
                    BankAccount acc = type == "1"
                        ? new SavingsAccount(accNo, name, balance)
                        : new CurrentAccount(accNo, name, balance);

                    accounts.Add(acc);
                    Console.WriteLine("Account created successfully.");
                    break;

                case "2":
                    Console.Write("Enter Account Number: ");
                    long depNo = Convert.ToInt64(Console.ReadLine());
                    var depAcc = accounts.Find(a => a.AccountNumber ==
depNo);
                    if (depAcc != null)
                    {
                        Console.Write("Enter deposit amount: ");
                        double amt = Convert.ToDouble(Console.ReadLine());
                        depAcc.Deposit(amt);
                    }
                    else Console.WriteLine("Account not found.");
                    break;

                case "3":
                    Console.Write("Enter Account Number: ");
                    long withNo = Convert.ToInt64(Console.ReadLine());
                    var withAcc = accounts.Find(a => a.AccountNumber ==
withNo);
                    if (withAcc != null)
                    {
                        Console.Write("Enter withdrawal amount: ");
                        double amt = Convert.ToDouble(Console.ReadLine());
                        withAcc.Withdraw(amt);
                    }
                    else Console.WriteLine("Account not found.");
                    break;

                case "4":
                    foreach (var a in accounts)
                    {
                        Console.WriteLine($"\nAccount {a.AccountNumber}:");
                        a.CalculateInterest();
                    }
                    break;

                case "5":
                    running = false;
                    break;

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