using System;

namespace AccountClass

{

    public class Program

    {

        private static Bank bank = new Bank();

        private static MenuOption ReadUserOption()

        {

            int option;

            do

            {

                Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

                Console.WriteLine("\* 1. Withdraw the money   \*");

                Console.WriteLine("\* 2. Deposit the money    \*");

                Console.WriteLine("\* 3. Transfer between accounts \*");

                Console.WriteLine("\* 4. Print account details \*");

                Console.WriteLine("\* 5. New Account           \*");

                Console.WriteLine("\* 6. Print transaction history \*");

                Console.WriteLine("\* 7. Quit                  \*");

                Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

                Console.WriteLine("Choose an option [1-7]");

                try

                {

                    option = Convert.ToInt32(Console.ReadLine());

                }

                catch

                {

                    option = -1;

                }

            } while (option < 1 || option > 7);

            return (MenuOption)(option - 1);

        }

        private static void DoTransfer(Bank bank)

        {

            Account fromAccount = FindAccount(bank);

            if (fromAccount == null) return;

            Account toAccount = FindAccount(bank);

            if (toAccount == null) return;

            decimal amount;

            do

            {

                Console.WriteLine("Enter the amount to transfer:");

                try

                {

                    amount = Convert.ToDecimal(Console.ReadLine());

                }

                catch

                {

                    amount = 0;

                }

            } while (amount <= 0);

            var transferTransaction = new TransferTransaction(fromAccount, toAccount, amount);

            bank.ExecuteTransaction(transferTransaction);

            transferTransaction.Print();

        }

        private static void DoDeposit(Bank bank)

        {

            Account toAccount = FindAccount(bank);

            if (toAccount == null) return;

            decimal money;

            do

            {

                try

                {

                    Console.WriteLine("How much would you like to deposit:");

                    money = Convert.ToDecimal(Console.ReadLine());

                }

                catch

                {

                    money = 0;

                }

            } while (money <= 0);

            var depositTransaction = new DepositTransaction(toAccount, money);

            bank.ExecuteTransaction(depositTransaction);

            depositTransaction.Print();

        }

        private static void DoWithdraw(Bank bank)

        {

            Account fromAccount = FindAccount(bank);

            if (fromAccount == null) return;

            decimal money;

            do

            {

                try

                {

                    Console.WriteLine("How much would you like to withdraw:");

                    money = Convert.ToDecimal(Console.ReadLine());

                }

                catch

                {

                    money = 0;

                }

            } while (money <= 0);

            var withdrawTransaction = new WithdrawTransaction(fromAccount, money);

            bank.ExecuteTransaction(withdrawTransaction);

            withdrawTransaction.Print();

        }

        private static void DoPrint(Bank bank)

        {

            Account account = FindAccount(bank);

            if (account == null) return;

            account.Print();

        }

        private static Account FindAccount(Bank fromBank)

        {

            Console.Write("Enter account name: ");

            string name = Console.ReadLine();

            Account result = fromBank.GetAccount(name);

            if (result == null)

            {

                Console.WriteLine($"No account found with name {name}");

            }

            return result;

        }

        public static void Main()

        {

            bank.AddAccount(new Account("Saeed's Account", 200000));

            bank.AddAccount(new Account("Jake's Account", 100000));

            MenuOption userOption;

            do

            {

                userOption = ReadUserOption();

                switch (userOption)

                {

                    case MenuOption.Withdraw:

                        DoWithdraw(bank);

                        break;

                    case MenuOption.Deposit:

                        DoDeposit(bank);

                        break;

                    case MenuOption.Transfer:

                        DoTransfer(bank);

                        break;

                    case MenuOption.Print:

                        DoPrint(bank);

                        break;

                    case MenuOption.NewAccount:

                        Console.Write("Enter account name: ");

                        string name = Console.ReadLine();

                        decimal balance;

                        do

                        {

                            Console.Write("Enter starting balance: ");

                        } while (!decimal.TryParse(Console.ReadLine(), out balance) || balance < 0);

                        bank.AddAccount(new Account(name, balance));

                        Console.WriteLine("Account created successfully.");

                        break;

                    case MenuOption.PrintTransactionHistory:

                        bank.PrintTransactionHistory();

                        break;

                    case MenuOption.Quit:

                        Console.WriteLine("You have quit the scenario.");

                        break;

                }

            } while (userOption != MenuOption.Quit);

        }

        public enum MenuOption

        {

            Withdraw,

            Deposit,

            Transfer,

            Print,

            NewAccount,

            PrintTransactionHistory,

            Quit

        }

    }

}

using System;

namespace AccountClass

{

    public class WithdrawTransaction : Transaction

    {

        private Account \_account;

        public WithdrawTransaction(Account account, decimal amount)

            : base(amount)

        {

            \_account = account;

        }

        public override void Execute()

        {

            base.Execute();

            if (\_account.Withdraw(\_amount))

            {

                Console.WriteLine("Withdrawal successful.");

            }

        }

        public override void Rollback()

        {

            base.Rollback();

            \_account.Deposit(\_amount);

        }

        public override void Print()

        {

            Console.WriteLine($"Withdraw Transaction: Amount {\_amount:C} on {DateStamp}");

        }

    }

}

using System;

namespace AccountClass

{

    public class TransferTransaction : Transaction

    {

        private Account \_toAccount;

        private Account \_fromAccount;

        public TransferTransaction(Account fromAccount, Account toAccount, decimal amount)

            : base(amount)

        {

            \_fromAccount = fromAccount;

            \_toAccount = toAccount;

        }

        public override void Execute()

        {

            base.Execute();

            var withdraw = new WithdrawTransaction(\_fromAccount, \_amount);

            var deposit = new DepositTransaction(\_toAccount, \_amount);

            withdraw.Execute();

            if (!withdraw.Executed)

            {

                Console.WriteLine("Withdraw failed.");

                return;

            }

            deposit.Execute();

            if (!deposit.Executed)

            {

                Console.WriteLine("Deposit failed. Rolling back withdraw.");

                withdraw.Rollback();

                return;

            }

        }

        public override void Rollback()

        {

            base.Rollback();

            var withdraw = new WithdrawTransaction(\_fromAccount, \_amount);

            var deposit = new DepositTransaction(\_toAccount, \_amount);

            deposit.Rollback();

            withdraw.Rollback();

        }

        public override void Print()

        {

            Console.WriteLine($"Transfer Transaction: {\_amount:C} from {\_fromAccount.Name} to {\_toAccount.Name} on {DateStamp}");

        }

    }

}

using System;

namespace AccountClass

{

    public abstract class Transaction

    {

        protected decimal \_amount;

        private bool \_executed;

        private bool \_reversed;

        private DateTime \_dateStamp;

        public Transaction(decimal amount)

        {

            \_amount = amount;

            \_executed = false;

            \_reversed = false;

        }

        public bool Executed

        {

            get { return \_executed; }

        }

        public virtual bool Success { get; }

        public bool Reversed

        {

            get { return \_reversed; }

        }

        public DateTime DateStamp

        {

            get { return \_dateStamp; }

        }

        public virtual void Execute()

        {

            if (\_executed)

            {

                throw new InvalidOperationException("Transaction has already been executed.");

            }

            \_executed = true;

            \_dateStamp = DateTime.Now;

        }

        public virtual void Rollback()

        {

            if (!\_executed)

            {

                throw new InvalidOperationException("Transaction has not been executed.");

            }

            if (\_reversed)

            {

                throw new InvalidOperationException("Transaction has already been reversed.");

            }

            \_reversed = true;

        }

        public abstract void Print();

    }

}

using System;

namespace AccountClass

{

    public class DepositTransaction : Transaction

    {

        private Account \_account;

        public DepositTransaction(Account account, decimal amount)

            : base(amount)

        {

            \_account = account;

        }

        public override void Execute()

        {

            base.Execute();

            if (\_account.Deposit(\_amount))

            {

                Console.WriteLine("Deposit successful.");

            }

        }

        public override void Rollback()

        {

            base.Rollback();

            \_account.Withdraw(\_amount);

        }

        public override void Print()

        {

            Console.WriteLine($"Deposit Transaction: Amount {\_amount:C} on {DateStamp}");

        }

    }

}

using System;

using System.Collections.Generic;

namespace AccountClass

{

    public class Bank

    {

        private List<Transaction> \_transactions;

        private List<Account> \_accounts;  // <-- Define the \_accounts list

        public Bank()

        {

            \_transactions = new List<Transaction>();

            \_accounts = new List<Account>();  // <-- Initialize the \_accounts list

        }

        public void AddAccount(Account account)

        {

            \_accounts.Add(account);

        }

        public Account GetAccount(string name)

        {

            return \_accounts.Find(account => account.Name == name);

        }

        public void ExecuteTransaction(Transaction transaction)

        {

            \_transactions.Add(transaction);

            transaction.Execute();

        }

        public void PrintTransactionHistory()

        {

            Console.WriteLine("Transaction History:");

            foreach (var transaction in \_transactions)

            {

                transaction.Print();

            }

        }

    }

}

using System;

using SplashKitSDK;

namespace AccountClass

{

    public class Account

    {

        private decimal \_balance;

        private string \_name;

        public Account(string name, decimal balance)

        {

            \_name = name;

            \_balance = balance;

        }

        public bool Withdraw(decimal amountWithdraw)

        {

            if (amountWithdraw <= \_balance && amountWithdraw>0)

            {

                \_balance -= amountWithdraw;

                return true;

            }

            else

            {

                Console.WriteLine("Insufficient balance.");

                return false;

            }

        }

        public bool Deposit(decimal amountToDeposit)

        {

            if (amountToDeposit > 0)

            {

                \_balance += amountToDeposit;

                return true;

            }

            return false;

        }

        public void Print()

        {

            Console.WriteLine("Name: " + \_name + "   Balance " + \_balance);

        }

        public string Name

        {

            get { return \_name; }

        }

    }

}