ATM Program Code

CheckingAccounts

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class CheckingAccount : Account

{

public int OverdraftLimit { get; private set; }

public CheckingAccount(int accountNumber, int password, int overdraftLimit)

: base(accountNumber, password)

{

OverdraftLimit = overdraftLimit;

}

public override string Withdraw(int amount)

{

if (amount <= Balance + OverdraftLimit)

{

base.Withdraw(amount);

return $"You have withdrawn ${amount}. Your new balance is ${Balance}";

}

else

{

return null;

}

}

public override string Deposit(int amount)

{

base.Deposit(amount);

return $"You have deposited ${amount}. Your new balance is ${Balance}";

}

}

}

SavingAccount

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class SavingsAccount : Account

{

public int InterestRate { get; private set; }

public SavingsAccount(int accountNumber, int password, int interestRate)

: base(accountNumber, password)

{

InterestRate = interestRate;

}

public override string Deposit(int amount)

{

base.Deposit(amount);

Balance += Balance \* (InterestRate / 100);

return ($"Interest applied at {InterestRate}% for deposit of ${amount}. Now your balance is ${Balance}");

}

public override string Withdraw(int amount)

{

if (amount <= Balance)

{

base.Withdraw(amount);

return $"You have withdrawn ${amount}. Your new balance is ${Balance}";

}

else

{

return null;

}

}

}

}

Account

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class Account

{

int \_accountNumber;

int \_password;

int \_balance = 0;

public Account(int accountNumber, int password)

{

\_accountNumber = accountNumber;

\_password = password;

}

public int AccountNumber

{

get { return \_accountNumber; }

set { \_accountNumber = value; }

}

public int Password

{

get { return \_password; }

set { \_password = value; }

}

public int Balance

{

get { return \_balance; }

set { \_balance = value; }

}

public virtual string Deposit(int amount)

{

\_balance += amount;

return $"You have deposited ${amount}. Your new balance is ${\_balance}";

}

public virtual string Withdraw(int amount)

{

if (amount > \_balance)

{

return null;

}

\_balance -= amount;

return $"You have withdrawn ${amount}. Your new balance is ${\_balance}";

}

public string Transfer(int amount, Account destinationAccount)

{

if (amount > \_balance)

{

return null;

}

else

{

\_balance -= amount;

destinationAccount.Deposit(amount);

return $"You have transferred ${amount} to account number {destinationAccount.AccountNumber}. Your new balance is ${\_balance}";

}

}

public Account? AreYou(int InputAccountNum, int InputPassword)

{

if (InputAccountNum == \_accountNumber && InputPassword == \_password)

{

return this;

}

else

{

return null;

}

}

}

}

Account Options

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class AccountOptions

{

public AccountOptions()

{

}

public bool AccountOptionsMenu(Account account, List<Bank> banksDatabase)

{

bool isSignedOut = false;

Console.WriteLine("\nPlease select an option: \n1. Deposit \n2. Withdraw \n3. Transfer \n4. Check balance \n5. Sign out\n");

int InputOption = Convert.ToInt32(Console.ReadLine().Trim());

switch (InputOption)

{

case 1:

Console.WriteLine("Please enter the amount you would like to deposit: ");

int DepositAmount;

while (true)

{

string input = Console.ReadLine().Trim();

if (int.TryParse(input, out DepositAmount) && DepositAmount > 0)

{

break;

}

else

{

Utility.PrintColoredMessage("Invalid input. Please enter a valid positive number.", ConsoleColor.Red);

input = Console.ReadLine().Trim();

}

}

Console.Clear();

Console.WriteLine(account.Deposit(DepositAmount));

AccountOptionsMenu(account, banksDatabase);

return isSignedOut;

case 2:

Console.WriteLine("Please enter the amount you would like to withdraw: ");

int WithdrawAmount;

while (true)

{

string input = Console.ReadLine().Trim();

if (int.TryParse(input, out WithdrawAmount) && WithdrawAmount > 0)

{

break;

}

else

{

Utility.PrintColoredMessage("Invalid input. Please enter a valid positive number.", ConsoleColor.Red);

input = Console.ReadLine().Trim();

}

}

if (account.Withdraw(WithdrawAmount) == null)

{

Utility.PrintColoredMessage("Insufficient funds. Please try again.", ConsoleColor.Red);

AccountOptionsMenu(account, banksDatabase);

return isSignedOut;

}

else

{

Console.Clear();

Console.WriteLine(account.Withdraw(WithdrawAmount));

AccountOptionsMenu(account, banksDatabase);

return isSignedOut;

}

case 3:

Console.WriteLine("Please enter the amount you would like to transfer: ");

int TransferAmount;

while (true)

{

string input = Console.ReadLine().Trim();

if (int.TryParse(input, out TransferAmount) && TransferAmount > 0)

{

break;

}

else

{

Utility.PrintColoredMessage("Invalid input. Please enter a valid positive number.", ConsoleColor.Red);

input = Console.ReadLine().Trim();

}

}

BankSelector bankSelector = new BankSelector(banksDatabase);

Console.WriteLine("Please enter the bank of the account you want to transfer to:\n" + bankSelector.BankList);

string DestinationBank = Console.ReadLine().Trim();

Console.WriteLine("\nPlease enter the account number you would like to transfer to (6 digits): ");

int DestinationAccountNum;

while (true)

{

string input = Console.ReadLine().Trim();

if (int.TryParse(input, out DestinationAccountNum) && input.Length == 6)

{

break;

}

else

{

Utility.PrintColoredMessage("Invalid input. Please enter a valid 6-digit account number.", ConsoleColor.Red);

}

}

foreach (Bank bank in banksDatabase)

{

if (bank.AreYou(DestinationBank))

{

Account? destinationAccount = bank.GetTransferAccount(DestinationAccountNum);

if (destinationAccount == null)

{

Utility.PrintColoredMessage("Account not found. Please try again.", ConsoleColor.Red);

AccountOptionsMenu(account, banksDatabase);

}

else if (account.Withdraw(TransferAmount) == null)

{

Utility.PrintColoredMessage("Insufficient funds. Please try again.", ConsoleColor.Red);

AccountOptionsMenu(account, banksDatabase);

}

else

{

Console.Clear();

Console.WriteLine(account.Transfer(TransferAmount, destinationAccount));

AccountOptionsMenu(account, banksDatabase);

}

}

}

Utility.PrintColoredMessage("Bank not found. Please try again.", ConsoleColor.Red);

AccountOptionsMenu(account, banksDatabase);

return isSignedOut;

case 4:

Console.Clear();

Console.WriteLine("Your current balance is $" + account.Balance);

AccountOptionsMenu(account, banksDatabase);

return isSignedOut;

case 5:

Console.Clear();

Console.WriteLine("You have signed out successfully.");

isSignedOut = true;

return isSignedOut;

default:

Console.WriteLine("Invalid option. Please try again.");

AccountOptionsMenu(account, banksDatabase);

return isSignedOut;

}

}

}

}

Utility

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class Utility

{

public static string GetSecretInput(string prompt)

{

Console.Write(prompt + "\n");

StringBuilder input = new StringBuilder();

while (true)

{

ConsoleKeyInfo inputKey = Console.ReadKey(true);

if (inputKey.Key == ConsoleKey.Enter)

{

break;

}

if (inputKey.Key == ConsoleKey.Backspace)

{

if (input.Length > 0)

{

input.Remove(input.Length - 1, 1);

// Move the cursor back, overwrite the last asterisk with a space, and move the cursor back again

Console.Write("\b \b");

}

}

else

{

input.Append(inputKey.KeyChar);

Console.Write("\*");

}

}

Console.WriteLine(); // Move to the next line after input is done

return input.ToString();

}

public static void PrintColoredMessage(string message, ConsoleColor color)

{

Console.ForegroundColor = color;

Console.WriteLine(message);

Console.ResetColor();

}

public static void PressEnterToContinue()

{

Console.WriteLine("\nPress Enter to continue...");

while (Console.ReadKey(true).Key != ConsoleKey.Enter)

{

}

}

}

}

Validator

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class Validator

{

public Validator()

{

}

public Account? SignInOptions(Bank bank)

{

Console.WriteLine("Please select an option: \n1. Sign in \n2. Create a normal account\n3. Create a saving account\n4. Create a checking account\n5. Back");

int InputOption = Convert.ToInt32(Console.ReadLine().Trim());

if (InputOption == 5)

{

return null;

}

switch (InputOption)

{

case 1:

Console.WriteLine("Please enter your account number (6 digits): ");

int InputAccountNum = Convert.ToInt32(Console.ReadLine().Trim());

int InputPassword = Convert.ToInt32(Utility.GetSecretInput("Please enter your password (4 digits): "));

if (InputAccountNum.ToString().Length != 6 || InputPassword.ToString().Length != 4)

{

Utility.PrintColoredMessage("Invalid account number or password. Please try again.", ConsoleColor.Red);

return SignInOptions(bank);

}

else if (bank.GetAccount(InputAccountNum, InputPassword) == null)

{

Utility.PrintColoredMessage("Account not found. Please try again.", ConsoleColor.Red);

return SignInOptions(bank);

}

else

{

Console.Clear();

Utility.PrintColoredMessage("You have logged in successfully. Welcome back " + (bank.GetAccount(InputAccountNum, InputPassword).AccountNumber), ConsoleColor.Green);

return bank.GetAccount(InputAccountNum, InputPassword);

}

break;

case 2:

int NewAccountNum;

Console.WriteLine("\nPlease enter your new account number (6 digits): ");

string accountNumInput = Console.ReadLine();

if (!int.TryParse(accountNumInput, out NewAccountNum) || accountNumInput.Length != 6)

{

Utility.PrintColoredMessage("Invalid account number format. Please enter a 6-digit number.", ConsoleColor.Red);

return SignInOptions(bank);

}

int NewPassword;

string passwordInput = Utility.GetSecretInput("Please enter your preferred password (4 digits): ");

if (!int.TryParse(passwordInput, out NewPassword) || passwordInput.Length != 4)

{

Utility.PrintColoredMessage("Invalid password format. Please enter a 4-digit number.", ConsoleColor.Red);

return SignInOptions(bank);

}

Account newAccount = new Account(NewAccountNum, NewPassword);

bank.AddAccount(newAccount);

Console.Clear();

Utility.PrintColoredMessage("Account created successfully!", ConsoleColor.Green);

return SignInOptions(bank);

break;

case 3:

int NewAccountNum2;

Console.WriteLine("\nPlease enter your new account number (6 digits): ");

string accountNumInput2 = Console.ReadLine();

if (!int.TryParse(accountNumInput2, out NewAccountNum2) || accountNumInput2.Length != 6)

{

Utility.PrintColoredMessage("Invalid account number format. Please enter a 6-digit number.", ConsoleColor.Red);

return SignInOptions(bank);

}

int NewPassword2;

string passwordInput2 = Utility.GetSecretInput("Please enter your preferred password (4 digits): ");

if (!int.TryParse(passwordInput2, out NewPassword2) || passwordInput2.Length != 4)

{

Utility.PrintColoredMessage("Invalid password format. Please enter a 4-digit number.", ConsoleColor.Red);

return SignInOptions(bank);

}

int InputInterestRate;

Console.WriteLine("Please select your desired interest rate: \n1. 3%\n2. 5%");

string interestRateInput = Console.ReadLine().Trim();

if (!int.TryParse(interestRateInput, out InputInterestRate) || (InputInterestRate != 1 && InputInterestRate != 2))

{

Utility.PrintColoredMessage("Invalid interest rate choice. Please enter either 1 or 2.", ConsoleColor.Red);

return SignInOptions(bank);

}

if (InputInterestRate == 1)

{

SavingsAccount Account = new SavingsAccount(NewAccountNum2, NewPassword2, 3);

bank.AddAccount(Account);

}

else

{

SavingsAccount Account = new SavingsAccount(NewAccountNum2, NewPassword2, 5);

bank.AddAccount(Account);

}

Utility.PrintColoredMessage("Account created successfully!", ConsoleColor.Green);

return SignInOptions(bank);

break;

case 4:

int NewAccountNum3;

Console.WriteLine("\nPlease enter your new account number (6 digits): ");

string accountNumInput3 = Console.ReadLine();

if (!int.TryParse(accountNumInput3, out NewAccountNum3) || accountNumInput3.Length != 6)

{

Utility.PrintColoredMessage("Invalid account number format. Please enter a 6-digit number.", ConsoleColor.Red);

return SignInOptions(bank);

}

int NewPassword3;

string passwordInput3 = Utility.GetSecretInput("Please enter your preferred password (4 digits): ");

if (!int.TryParse(passwordInput3, out NewPassword3) || passwordInput3.Length != 4)

{

Utility.PrintColoredMessage("Invalid password format. Please enter a 4-digit number.", ConsoleColor.Red);

return SignInOptions(bank);

}

int InputOverdraftLimit;

Console.WriteLine("Please select your desired overdraft limit: \n1. $100\n2. $200");

string overdraftLimitInput = Console.ReadLine().Trim();

if (!int.TryParse(overdraftLimitInput, out InputOverdraftLimit) || (InputOverdraftLimit != 1 && InputOverdraftLimit != 2))

{

Utility.PrintColoredMessage("Invalid overdraft limit choice. Please enter either 1 or 2.", ConsoleColor.Red);

return SignInOptions(bank);

}

CheckingAccount accountToCreate;

if (InputOverdraftLimit == 1)

{

accountToCreate = new CheckingAccount(NewAccountNum3, NewPassword3, 100);

}

else

{

accountToCreate = new CheckingAccount(NewAccountNum3, NewPassword3, 200);

}

bank.AddAccount(accountToCreate);

Utility.PrintColoredMessage("Account created successfully!", ConsoleColor.Green);

return SignInOptions(bank);

break;

default:

Utility.PrintColoredMessage("Invalid option. Please try again.", ConsoleColor.Red);

return SignInOptions(bank);

}

}

}

}

BankSelector

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class BankSelector

{

internal List<Bank> \_banks = new List<Bank>();

public BankSelector(List<Bank> banks)

{

\_banks = banks;

foreach (Bank bank in \_banks)

{

int num = \_banks.IndexOf(bank) + 1;

bank.AddId(num.ToString());

}

}

public Bank? SelectBank()

{

Console.WriteLine("Please select a bank: \n" + BankList);

string InputBank = Console.ReadLine().ToLower().Trim();

foreach (Bank bank in \_banks)

{

if (bank.AreYou(InputBank))

{

Console.Clear();

Utility.PrintColoredMessage("Welcome to " + bank.Name + ".", ConsoleColor.Green);

return bank;

}

}

return null;

}

public string BankList

{

get

{

int i = 1;

string bankList = "";

foreach (Bank bank in \_banks)

{

bankList += i + ". " + bank.Name + " (" + bank.Id + ")\n";

i++;

}

return bankList;

}

}

}

}

Bank

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class Bank

{

private List<Account> \_accounts = new List<Account>();

private List<string> \_ids = new List<string>();

private string \_name;

public Bank(List<string> ids, string name)

{

foreach (string id in ids)

{

\_ids.Add(id.ToLower());

\_name = name;

}

}

public string Id

{

get { return \_ids[0].ToUpper(); }

}

public void AddId(string id)

{

\_ids.Add(id.ToLower());

}

public bool AreYou(string id)

{

if (\_ids.Contains(id.ToLower()))

{

return true;

}

else

{

return false;

}

}

public string Name

{

get { return \_name; }

}

public void AddAccount(Account account)

{

\_accounts.Add(account);

}

public void DeleteAccount(Account account)

{

\_accounts.Remove(account);

}

public Account? GetAccount(int accountNumber, int password)

{

foreach (Account account in \_accounts)

{

if (account.AccountNumber == accountNumber && account.Password == password)

{

return account;

}

}

return null;

}

public Account? GetTransferAccount(int accountNumber)

{

foreach (Account account in \_accounts)

{

if (account.AccountNumber == accountNumber)

{

return account;

}

}

return null;

}

}

}

Program

using System;

using System.Collections.Generic;

using System.Diagnostics.Eventing.Reader;

using System.Linq;

using System.Net;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace ATM

{

public class Program

{

static void Main(string[] args)

{

//Set up banks and accounts

Bank cba = new Bank(new List<string> {"CBA"}, "Commonwealth");

Account account1 = new Account(123456, 1234);

cba.AddAccount(account1);

Bank westpac = new Bank(new List<string> {"WBC"}, "Westpac");

Account account2 = new Account(654321, 4321);

westpac.AddAccount(account2);

Bank nab = new Bank(new List<string> { "NAB" }, "National Australia Bank");

Bank anz = new Bank(new List<string> { "ANZ" }, "Australia and New Zealand Banking Group");

List<Bank> banksDatabase = new List<Bank> { cba, westpac, nab, anz };

// Set the console title and text color

Console.Title = "My ATM Application";

Console.ForegroundColor = ConsoleColor.White;

// Display the welcome message

Utility.PrintColoredMessage("\n\n-----------------------------------Welcome to the ATM-----------------------------------\n", ConsoleColor.Green);

Utility.PressEnterToContinue();

//Sign in or create an account to sign in

BankSelector bankSelector = new BankSelector(banksDatabase);

Bank? selectedBank = bankSelector.SelectBank();

while (selectedBank == null)

{

Utility.PrintColoredMessage("Please select a valid bank by either abbreviated name or number", ConsoleColor.Red);

selectedBank = bankSelector.SelectBank();

}

Validator validator = new Validator();

Account signedInAccount = validator.SignInOptions(selectedBank);

//Make sure the user is signed in first before proceeding

while (signedInAccount == null)

{

Console.Clear();

selectedBank = bankSelector.SelectBank();

signedInAccount = validator.SignInOptions(selectedBank);

}

Utility.PressEnterToContinue();

//Display account options

AccountOptions accountOptions = new AccountOptions();

bool isSignedOut = accountOptions.AccountOptionsMenu(signedInAccount, banksDatabase);

//For the sign out option

while (isSignedOut)

{

Console.Clear();

signedInAccount = validator.SignInOptions(selectedBank);

while (signedInAccount == null)

{

selectedBank = bankSelector.SelectBank();

signedInAccount = validator.SignInOptions(selectedBank);

}

isSignedOut = accountOptions.AccountOptionsMenu(signedInAccount, banksDatabase);

}

}

}

}