ДАНЯ

За завданням, ми створили програму, яка моделює роботу банкомату. Зрештою, усе рішення складається з трьох проектів: бібліотеки класів, вона ось тут, уверху. Віконного проекту, він найнижчий. І консольного проекту, який повністю повторює функціонал віконного, він у експлоері по серединці.

Почнемо, власне, з бібліотеки. За завданням ми створили класи **Account**, який є поданням карткового рахунку клієнта, **AutomatedTellerMachine,** який моделює роботу банкомата і **Bank**, що представляє банк. Окрім цього нами були додані декілька додаткових класів. **Transaction**, який відповідає за усі операції з грошима у банку. **TransactionFilters** – відповідає за фільтрування транзакцій за різними періодами часу. І **BankInitializer** - клас, що ініціалізує виконання усіх інших класів.

ЛИНА

Далі перейдемо до віконного додатка.

\*запустити вінформу лаби\*

При запуску одразу ж виконується перша функція, яка була описана в завданні – функція автентифікації. Зараз в нас створено два акаунти, ми зайдемо на перший.

\*ввести логін 0123456789 і пароль 1234\*

Відкривається основне вікно з акаунтом користувача. Тут ми дивимось на баланс. Зараз ви можете його запам’ятати, щоб мати змогу відслідкувати, що наступні операції справді впливають на нього. Усі подальші операції обираємо приблизно так само, як і на справжньому банкоматі, просто не за допомогою кнопок, а за допомогою кліків миші.

Почнемо з опції «Withdraw», тобто зняття грошей з рахунку. Ось тут у поле вводимо суму, натискаємо «Confirm» ака «Підтвердити» і повертаємось на головний екран. Як бачите, баланс став трохи меншим.

Далі на черзі «Deposit», бо давайте підемо зверху вниз. Приблизно той самий інтерфейс, але іконка змінилась, як і мета вікна. Тут ми можемо ввести суму, яка *зачислиться* на наш рахунок. Ми також передбачили варіант введення від’ємного числа. Наш банк не зламається, а просто скаже Вам трохи подумати головою. Все передбачено, все досить логічно. Повертаємось на головний екран і о чудо, грошики надійшли.

Наступним давайте подивимось на «Transfer», себто переказ з картки на картку за номером іншої картки. Давайте переведемо грошиків на інший акаунт. Але запам’ятайте цю суму, бо ми ще з іншого облікового запису перевіримо, чи відбулась транзакція. Повертаємось на головний екран, принаймні з цієї сторони. Ось, все вийшло, баланс став меншим.

ДИМА

Банкоматів нашої компанії загалом 4, якщо враховувати той, на якому відбуваються операції зара. Їх адреси ми можемо подивитись після натискання на кнопочку справа зверху «ATMs Near». Для кожного вказані координати, за якими їх можна вбити у гугл мепс, щоб знайти і відстань до них усих, щоб у разі чого розуміти, до куди ближче їхати.

Далі кнопочка «History». Тут ми можемо побачити усі ті операції, які виконували до цього. Усі циферки збігаються, все добре.

І остання кнопочка, для виходу з облікового запису – «Exit». Як бачите, ми вийшли з акаунту. І загалом можемо зайти на другий, тому давайте так і зробимо, щоб перевірити правильність операції трансферу.

\*ввести логін 9876543210 і пароль 4321\*

Робимо і тут депозит, щоб одній операції на рахунку не було сумно самій. Заходимо в історію і бачимо тут трансфер.

ДАНЯ

І перед тим, як ми приступимо до перевірки консольного додатку, давайте спочатку пробіжимось по коду.

\*Відкрити Program.cs\*

Тут, як можна побачити, як на самому початку починається ініціалізація, після чого ми переходимо вже до класів, які вже бачили раніше. Не думаю, що є сенс дивитись на файли кожної сторінки, бо там просто ініціалізуються необхідні класи і підвантажуються бібліотеки.

ЛИНА

А ось тепер ми можемо перейти до консольного додатку

При запуску знов одразу функція автентифікації. Давайте цього разу будемо від імені другого користувача банку.

\*ввести логін 9876543210 і пароль 4321\*

Відкривається основне вікно з акаунтом користувача, де можна вибрати усі подальші дії. Думаю, почнемо ми з перевірки балансу. Видно, що число не «округлено» до тисяч, як було до цього, а, отже, трансфер прийшов.

І зара ще раз швиденько пробіжимось по всім іншим функціям, бо нам тре лише перевірити, що вони працюють, а себто повністю відповідають функціоналу віконного додатку.

Тицяємо на «Withdraw», знімаємо необхідну кількість грошиків. Одразу виводить залишок на картці, тому кожен раз натискати на «чек бАланс» не треба. Та сама історія з «Deposit», працює так само правильно. Одразу бачимо поповнення. Функція «Transfer» також працює. «History» виводить усі транзакції з часом, коли їх здійснили.

Кількість банкоматів нашої компанії не змінилась від додатку до додатку, але ми вирішили, що на консольному додатку, якщо ми будемо розширювати мережу, то тут повне подання буде максимально незручним. Тому він просто виводить найближчі. Якщо захоче побачити більше, ніж у нас є – він просто відкриє усі наявні.

І знову ж вихід з облікового запису – «Exit». З демонстрацією ми наче закінчили.

ДИМА

Переходячи до коду, одразу йдемо в файлик Program.cs.

\*Відкрити Program.cs\*

Отже, переходимо до коду консольного додатку, який також був створений на основі бібліотеки класів. Спочатку підключаємо бібліотеку AtmLibrary, що містить всі необхідні класи для роботи з банком, банкоматом та рахунками.

using AtmLibrary;

Далі ми визначаємо простір імен AtmConsoleApp і створюємо клас Program, в якому відбувається основна логіка програми. У класі визначені два делегати: AccountAction для операцій з акаунтом, і AtmAction для операцій з акаунтом і банкоматом.

namespace AtmConsoleApp

{

class Program

{

public delegate void AccountAction(Account account);

public delegate void AtmAction(Account account, AutomatedTellerMachine atm);

У методі Main програма починає з ініціалізації банку через клас BankInitializer, отримуючи список акаунтів. Після цього користувач проходить автентифікацію, яка відбувається за допомогою введення номера картки і PIN-коду.

static void Main(string[] args)

{

BankInitializer initializer = new BankInitializer();

var accounts = initializer.GetAccounts();

Account? currentAccount = AuthenticateUser(accounts);

Далі створюються делегати для основних операцій: перевірки балансу, зняття грошей і депозиту. Усі ці операції будуть виконуватися через виклик відповідних делегатів.

AccountAction checkBalanceDelegate = CheckBalance;

AtmAction withdrawMoneyDelegate = WithdrawMoney;

AccountAction depositMoneyDelegate = DepositMoney;

Основна частина програми — це цикл while, який відповідає за меню. Користувач обирає опцію з переліку можливих дій: перевірити баланс, зняти гроші, зробити депозит, переказати гроші, переглянути історію транзакцій, знайти найближчі банкомати, переключити акаунти або вийти з програми.

bool exit = false;

while (!exit)

{

Console.WriteLine("\nSelect an action:");

Console.WriteLine("1. Check balance");

Console.WriteLine("2. Withdraw money");

Console.WriteLine("3. Deposit money");

Console.WriteLine("4. Transfer money");

Console.WriteLine("5. View transaction history");

Console.WriteLine("6. Find nearest ATMs");

Console.WriteLine("7. Switch accounts");

Console.WriteLine("0. Exit");

string choice = Console.ReadLine() ?? "";

Для кожної опції є відповідний блок у switch, який викликає потрібну функцію. Наприклад, при виборі опції "1" викликається делегат для перевірки балансу.

switch (choice)

{

case "1":

checkBalanceDelegate(currentAccount);

break;

case "2":

withdrawMoneyDelegate(currentAccount, initializer.atm);

break;

case "3":

depositMoneyDelegate(currentAccount);

break;

case "4":

TransferMoney(currentAccount, accounts);

break;

ДАНЯ

Тепер трохи детальніше про кожну функцію. Для автентифікації ми використовуємо метод AuthenticateUser, де користувач вводить свій номер картки і PIN-код. Якщо все введено правильно, система вітає користувача.

static Account? AuthenticateUser(Dictionary<string, Account> accounts)

{

Account? currentAccount = null;

while (currentAccount == null)

{

Console.WriteLine("Enter card number:");

string cardNumber = Console.ReadLine() ?? "";

Console.WriteLine("Enter PIN:");

string pinCode = Console.ReadLine() ?? "";

if (accounts.ContainsKey(cardNumber) && accounts[cardNumber].ValidatePin(pinCode))

{

currentAccount = accounts[cardNumber];

Console.Clear();

Console.WriteLine($"Welcome, {currentAccount.Owner}!");

}

else

{

Console.WriteLine("Incorrect card number or PIN. Please try again.");

}

}

return currentAccount;

}

Функція CheckBalance виводить поточний баланс користувача.

static void CheckBalance(Account account)

{

Console.WriteLine($"\nYour current balance: {account.Balance} UAH");

}

Функція WithdrawMoney відповідає за зняття грошей з рахунку, перевіряючи правильність введеної суми і достатність коштів на рахунку.

static void WithdrawMoney(Account account, AutomatedTellerMachine atm)

{

Console.WriteLine("\nEnter amount to withdraw:");

if (decimal.TryParse(Console.ReadLine(), out decimal amount))

{

if (amount > 0 && account.Balance >= amount)

{

account.Withdraw(amount);

atm.DispenseCash(amount);

Console.WriteLine($"Successfully withdrew {amount} UAH. New balance: {account.Balance} UAH");

}

else

{

Console.WriteLine("Insufficient funds or invalid amount.");

}

}

}

Аналогічно працює функція DepositMoney, яка дозволяє користувачу внести кошти на свій рахунок.

static void DepositMoney(Account account)

{

Console.WriteLine("\nEnter amount to deposit:");

if (decimal.TryParse(Console.ReadLine(), out decimal amount) && amount > 0)

{

account.Deposit(amount);

Console.WriteLine($"Successfully deposited {amount} UAH. New balance: {account.Balance} UAH");

}

}

Функція TransferMoney дозволяє переказати гроші на інший рахунок за номером картки.

static void TransferMoney(Account currentAccount, Dictionary<string, Account> accounts)

{

Console.WriteLine("Enter card number to transfer money to:");

string targetCardNumber = Console.ReadLine() ?? "";

if (accounts.ContainsKey(targetCardNumber))

{

Account targetAccount = accounts[targetCardNumber];

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

if (decimal.TryParse(Console.ReadLine(), out decimal amount) && amount > 0)

{

currentAccount.TransferMoney(targetAccount, amount);

Console.WriteLine($"Successfully transferred {amount} UAH to {targetAccount.Owner}. New balance: {currentAccount.Balance} UAH");

}

}

}

Також є функція для перегляду історії транзакцій з можливістю фільтрації за певний період часу.

static void ShowTransactionHistory(Account account)

{

var transactions = account.GetTransactionHistory();

Console.WriteLine("\nSelect time period to filter transactions:");

Console.WriteLine("1. Current day");

Console.WriteLine("2. Current week");

Console.WriteLine("3. Current month");

Console.WriteLine("4. All transactions");

string choice = Console.ReadLine() ?? "";

List<Transaction> filteredTransactions = TransactionFilters.FilterByChoice(transactions, choice);

foreach (var transaction in filteredTransactions)

{

Console.WriteLine(transaction);

}

}

Функція FindNearestATMs показує найближчі банкомати на основі введеної кількості.

static void FindNearestATMs(Bank bank, AutomatedTellerMachine currentATM)

{

Console.WriteLine("\nHow many nearest ATMs would you like to see?");

if (int.TryParse(Console.ReadLine(), out int count))

{

var nearestATMs = bank.GetNearestATMs(currentATM, count);

foreach (var atm in nearestATMs)

{

Console.WriteLine($"{atm.Name} - {atm.DistanceFromCurrentATM(currentATM)} km");

}

}

}

Дякую за увагу