|  |  |
| --- | --- |
| Міністерство освіти і науки України  НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  “ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ” | |
| Кафедра управління проєктами в інформаційних технологіях | |
| Звіт з лабораторної № 1  «Основи розробки додатків у .NET»  з дисципліни «Стек технологій***»*** | |
|  | *Виконали*:  студенти  групи КН-322а  Данилова Ангеліна Вадимівна  Непоменко Дмитро Григорович  Лобач Даніїл Артемович |
| Харків, НТУ “ХПІ”  2024 | |

Зміст

[Мета: 2](#_Toc178712414)

[Теоретичні відомості 2](#_Toc178712415)

[Завдання 2](#_Toc178712416)

[Хід роботи 4](#_Toc178712417)

[Висновки 12](#_Toc178712418)

# Мета:

Навчитися створювати консольні та віконні додатки, ознайомитись з використання делегатів та подій у мові програмування C#.

# Теоретичні відомості

У платформі .NET можна виділити наступні основні компоненти:

Common Language Specification (CLS) — набір правил, які мають підтримуватись мовами програмування для сумісності.

Microsoft Intermediate Language (MSIL) — проміжний код, в який компілюється код на C# перед виконанням на платформі.

Common Language Runtime (CLR) — віртуальна машина, яка виконує MSIL, забезпечуючи управління пам'яттю, винятками та безпекою.

.NET Class Library (.NET FCL) — бібліотека класів, яка надає інтерфейси для стандартних операцій, таких як робота з файлами, мережею, базами даних тощо.

1. У C# делегати є типізовані посилання на методи, що дозволяє передавати методи як параметри, повертати їх з методів, а також використовувати їх для визначення подій та зворотних викликів.
2. Для оголошення делегата використовується ключове слово delegate, після якого йде тип, назва і параметри, що повертається.
3. Основна різниця між подією та делегатом полягає в тому, що подія представляє дію, яка може статися, і має відповідний обробник, який виконується при виникненні події, в той час як делегат – це об'єкт, що представляє дію, яка може бути викликана явно і може бути передана від однієї частини програми до інший.
4. Делегати, які включають більше одного методу, називають мультикаст-делегатами.
5. Найбільш використовуваними вбудованими делегатами є Action, Predicate та Func.
6. Делегати дозволяють передавати методи як параметри, реалізуючи тим самим зворотні виклики.
7. Анонімні методи використовують для створення екземплярів делегатів.
8. Події - це механізм сповіщення про зміни стану об'єкта. Вони реалізуються за допомогою делегатів.
9. Підписка на подію відбувається додаванням обробника подій до делегата, а відписка - його видаленням.
10. Події оголошуються за допомогою делегатів, а їхнє викликування відбувається через цей делегат.
11. Лямбда-вирази представляють спрощений запис анонімних методів.
12. Делегат C# дозволяє розглядати метод як значення, призначаючи метод змінній, передаючи його іншим методам як параметри, додаючи його до колекції, і так далі.
13. Action — делегат, який не повертає значення, але приймає параметри. Він використовується, коли потрібно викликати метод, який не повертає жодного значення, але може виконувати певні дії. Func — делегат, який приймає один або більше параметрів і повертає певне значення. Він використовується, коли потрібно викликати метод, який повертає результат після виконання.
14. Винятки, викинуті в обробнику подій, можуть бути перехоплені за допомогою блоку try-catch.

# Завдання

1. Опрацювати теорію (див. додаток А)

2. Написати програму для моделювання роботи банкомату.

Рішення має складатися з трьох проектів:

* бібліотека класів;
* віконний проект;
* консольний проект.

У бібліотеці класів реалізуйте загальні типи даних, які будуть використані у віконному та консольному проектах. У віконному додатку розробіть інтерфейс, який наближений до інтерфейсу традиційного банкомату. Консольний додаток повинен забезпечити ті ж самі можливості, що має віконний додаток, але за допомогою консольного меню.

Слід передбачити наступні можливості:

* автентифікація (перевірка введення номеру картки та пін-коду);
* перегляд балансу на картці;
* перегляд історії операцій з фільтруванням по проміжку часу (за поточний день, за поточний тиждень, за поточний місяць)
* зняття коштів;
* зарахування коштів на картку;
* перерахування коштів на картку із заданим номером;
* перегляд найближчих банкоматів;

Повинна бути передбачена можливість додавання власних обробників подій на виконання операцій автентифікації, перегляду балансу, зняття коштів та перерахування коштів на іншу картку.

Додайте власні обробники подій, які будуть виводити повідомлення:

− для віконного додатку - за допомогою виклику:

MessageBox.Show("Повідомлення");

− для консольного додатку - за допомогою виклику

Console.WriteLine("Повідомлення");

При виконанні завдання створіть класи, які забезпечать роботу додатку:

* **Account** - клас, який є поданням карткового рахунку клієнта (властивості: номер картки, прізвище та ім’я власника, баланс на рахунку, тощо);
* **AutomatedTellerMachine** - клас, який моделює роботу банкомата (містить кількість грошей, які містяться у банкоматі, ідентифікатор банкомату, його адресу розташування, тощо);
* **Bank** - клас, що представляє банк (назва банку, перелік банкоматів, тощо).

Це не виключний перелік класів, за необхідності створюйте інші класи, які Вам будуть необхідні при реалізації завдання.

# Хід роботи

Програма для моделювання роботи банкомату.

Рішення складається з трьох проектів:

* бібліотека класів;
* віконний проект;
* консольний проект.

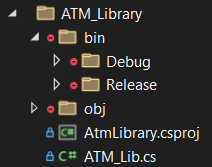


Рисунок 1 – Solution для проекту бібліотека класів.

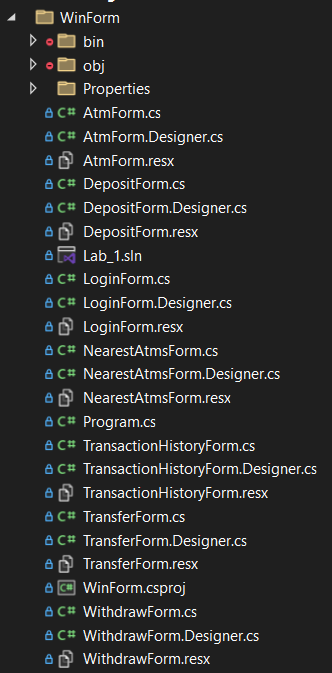


Рисунок 2 – Solution для проекту віконний проект.

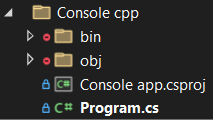


Рисунок 3 – Solution для проекту консольний проект.

У бібліотеці класів реалізовано загальні типи даних, які були використані у віконному та консольному проектах, а саме: Account, Transaction, TransactionFilters, AutomatedTellerMachine, Bank,

У віконному додатку розроблено інтерфейс, який наближений до інтерфейсу традиційного банкомату. Також розроблено консольний додаток, який забезпечує ті ж самі можливості, що має віконний додаток, але за допомогою консольного меню.

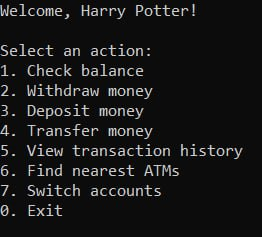


Рисунок 4 – Інтерфейс консольного додатка

Передбачено наступні можливості:

* автентифікація (перевірка введення номеру картки та пін-коду);

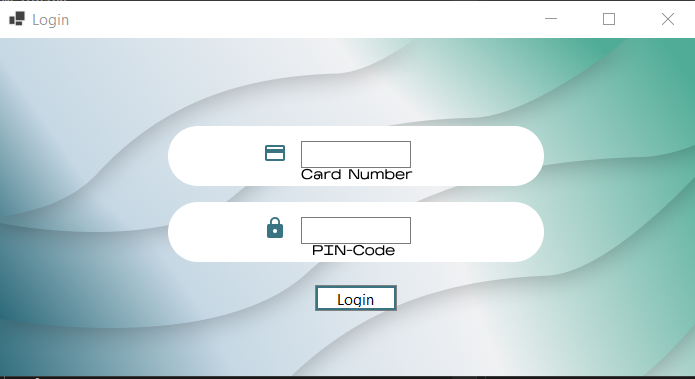


Рисунок 5 – Інтерфейс віконного додатка для функції автентифікації.

* перегляд балансу на картці;

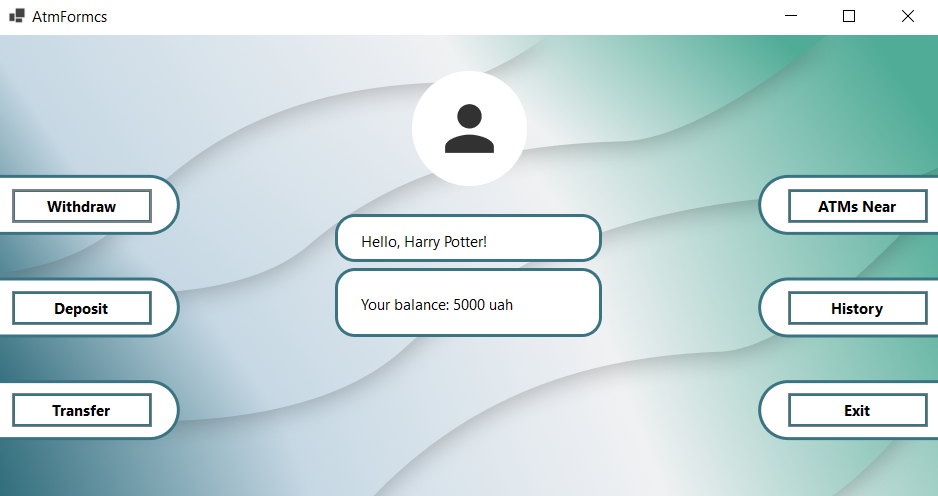


Рисунок 6 – Інтерфейс віконного додатка для функції перегляду балансу на картці

* перегляд історії операцій з фільтруванням по проміжку часу (за поточний день, за поточний тиждень, за поточний місяць)

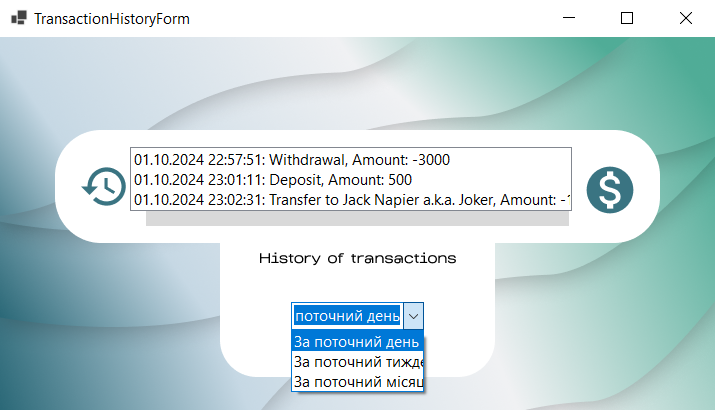


Рисунок 7 – Інтерфейс віконного додатка для функції перегляду історії операцій

* зняття коштів;



Рисунок 8 – Інтерфейс віконного додатка для функції зняття коштів

* зарахування коштів на картку;



Рисунок 9 – Інтерфейс віконного додатка для функції зарахування коштів на картку

* перерахування коштів на картку із заданим номером;

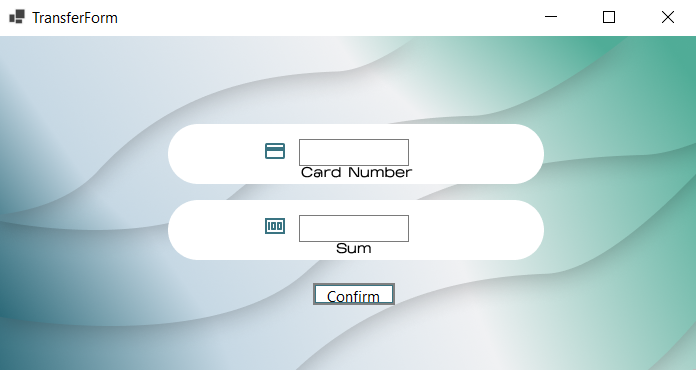


Рисунок 10 – Інтерфейс віконного додатка для функції перерахування коштів на картку

* перегляд найближчих банкоматів;

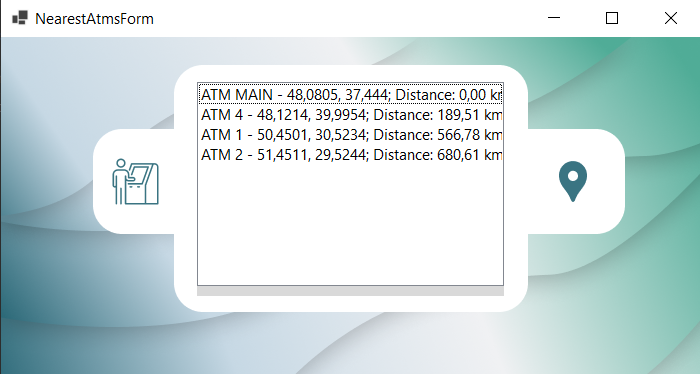


Рисунок 11 – Інтерфейс віконного додатка для функції перегляду найближчих банкоматів

Передбачена можливість додавання власних обробників подій на виконання операцій автентифікації, перегляду балансу, зняття коштів та перерахування коштів на іншу картку.

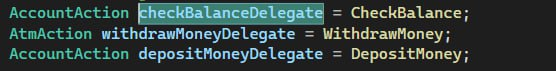


Рисунок 12 – Код для додавання власних обробників подій

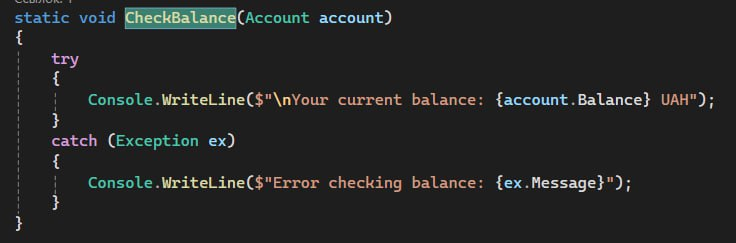


Рисунок 13 – Код для обробника події Перевірка Балансу

Додайте власні обробники подій, які будуть виводити повідомлення:

− для віконного додатку - за допомогою виклику:

MessageBox.Show("Повідомлення");

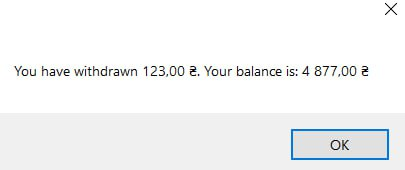
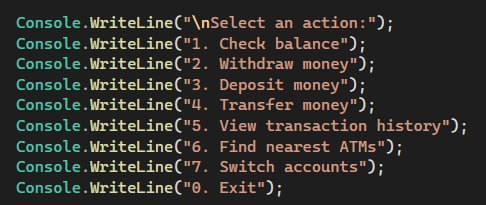


Рисунок 14 – MessageBox для виведення балансу

− для консольного додатку - за допомогою виклику

Console.WriteLine("Повідомлення");



При виконанні завдання було створено наступні класи, які забезпечують роботу додатку:

* **Account** - клас, який є поданням карткового рахунку клієнта (властивості: номер картки, прізвище та ім’я власника, баланс на рахунку, тощо);

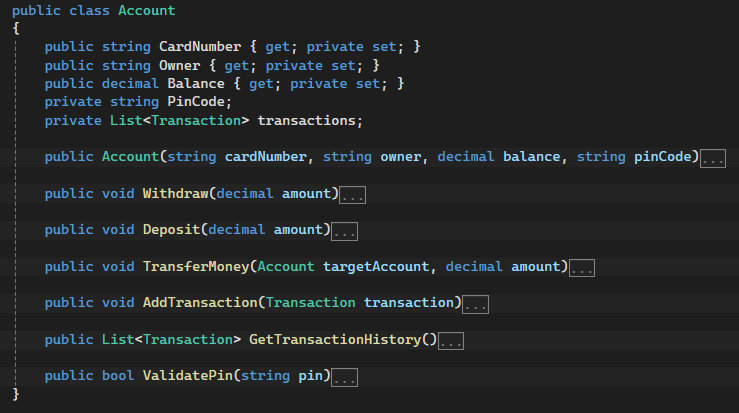


Рисунок 15 – Код для класу Account

* **Transaction** – клас, який відповідає за усі операції з грошима у банку;

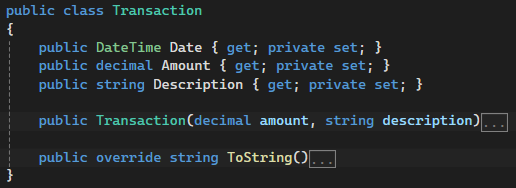
****

Рисунок 16 – Код для класу Transaction

* **TransactionFilters** - клас, відповідає за фільтрування транзакцій за різними періодами часу;

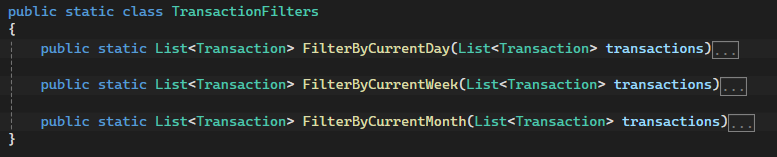


Рисунок 17 – Код для класу TransactionFilters

* **AutomatedTellerMachine** - клас, який моделює роботу банкомата (містить кількість грошей, які містяться у банкоматі, ідентифікатор банкомату, його адресу розташування, тощо);

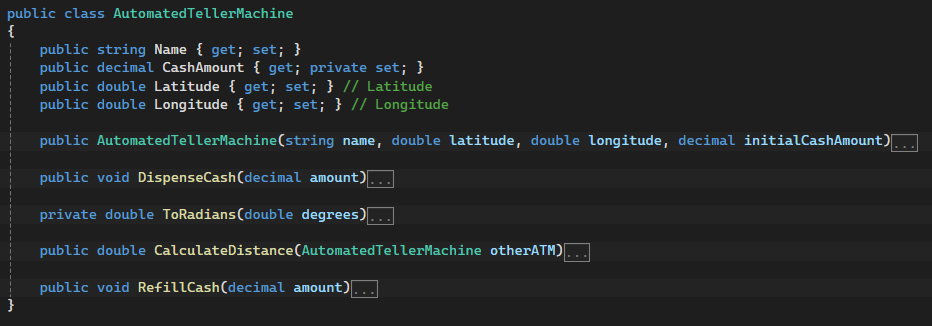


Рисунок 18 – Код для класу AutomatedTellerMachine

* **Bank** - клас, що представляє банк (назва банку, перелік банкоматів, тощо);

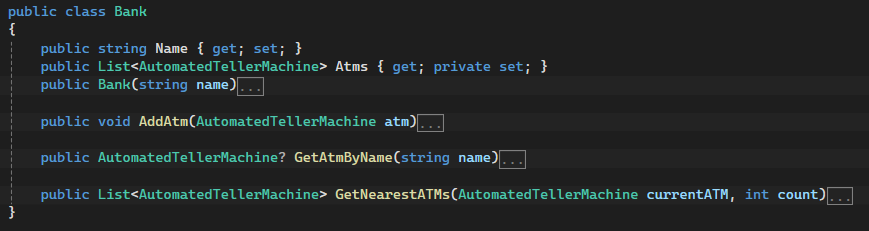


Рисунок 19 – Код для класу Bank

* **BankInitializer** - клас, що ініціалізує усі інші класи .

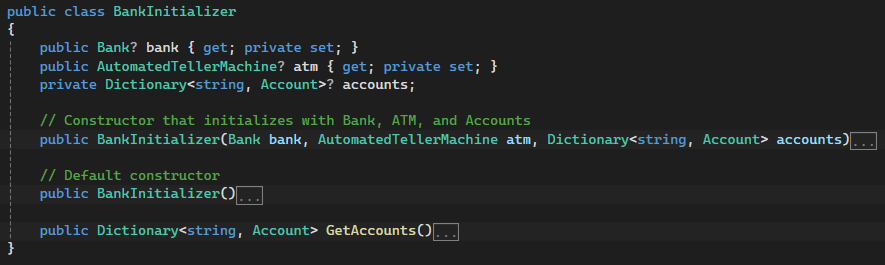


Рисунок 20 – Код для класу BankInitializer

# Висновки

Під час виконання лабораторної роботи було створено програму для моделювання роботи банкомату. Програма реалізована у вигляді трьох проектів: бібліотеки класів, віконного та консольного додатків. Основні функції, які були реалізовані: автентифікація користувача, перегляд балансу, історія операцій, зняття та зарахування коштів. Було розроблено обробники подій, які виводять повідомлення через MessageBox у віконному додатку та Console.WriteLine у консольному. Ця робота дозволила ознайомитись з основами створення додатків на платформі .NET, застосуванням делегатів та подій у мові програмування C#.