Л. І. Лозовська, Л. М. Бандоріна,
Р. В. Савчук, Т. О. Климкович

ПРИКЛАДНЕ ПРОГРАМУВАННЯ В БІЗНЕСІ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ

Л. І. Лозовська, Л. М. Бандоріна, Р. В. Савчук, Т. О. Климкович

Прикладне програмування в бізнесі

НАВЧАЛЬНИЙ ПОСІБНИК

ДНІПРО
2025

УДК 004.9:658.114(075.8)
П 75

Авторський колектив:
Лозовська Л. І., Бандоріна Л. М., Савчук Р. В., Климкович Т. О.

Рекомендовано Радою якості освітньої діяльності УДУНТ
Протокол № 6 від «18» лютого 2025 р.

Електронний аналог друкованого видання

П 75 Прикладне програмування в бізнесі : навч. посіб. / Л. І. Лозовська, Л. М. Бандоріна, Р. В. Савчук, Т. О. Климкович ; Укр. держ. ун-т науки і технологій. – Дніпро : УДУНТ, 2025. – 144 с.

Навчальний посібник призначений для використання студентами денної та заочної форм навчання спеціальності 126 «Інформаційні системи та технології» (бакалаврський рівень) під час вивчення дисципліни «Прикладне програмування в бізнесі».

Містить методичні матеріали для вивчення основ програмування мовою високого рівня C#. Одночасне ознайомлення з можливостями системи програмування і конструкцій мови програмування дозволяє студентам з перших занять одержати уявлення про технологію розробки Windows-додатків.

Іл. 43, табл. 5, бібліогр. 11 назв.

УДК 004.43:658.114(075.8)

DOI 10.15802/lib.001OER.25

ЗМІСТ

ВСТУП

Прикладне програмування передбачає створення програм за допомогою наглядних засобів, тобто шляхом оперування графічними об'єктами, а не написання програмного коду в текстовому вигляді. Основною задачею є побудова розв'язку поставленої задачі за допомогою візуальних заготовок, які вставляються у форму, присвоєння значень їхнім атрибутам і створення чи застосування потрібних для розв'язання даної задачі методів.

В основу розробки прикладних програм у бізнесі покладено візуальне програмування, як засіб автоматизації його процесів. Тепер для складання програми користувачу необхідно маніпулювати наданими графічними засобами – компонентами. Компоненти мають певні атрибути (властивості). Властивості можуть набувати значення зі заздалегідь фіксованого значення чи набору, або можуть бути придумані користувачем. Для опрацювання числових та інших даних розроблюються відповідні методи об'єктів.

Метою даного видання є формування теоретичних знань і практичних навичок створення програмних додатків для бізнесу з використанням .NET технологій. В даному навчальному посібнику паралельно розглядаються можливості системи програмування і конструкції мови програмування, що дозволяє студентам з перших занять отримати уявлення про технології розробки Windows-додатків. Після вивчення дисципліни студент може набути всі необхідні навички і уміння для проектування програм достатньо високої складності, які передбачені програмою навчальної дисципліни:

ОРН1. Розуміти принципи роботи програм WINDOWS, їх різновиди. Вміти розробляти консольні додатки в середовищі Microsoft Visual.NET.

ОРН2. Вміти розробляти додатки в середовищі Microsoft Visual.NET з використанням форм. Вміти проводити настройку зовнішнього вигляду і поведінки форми, додавання та використовувати базові елементів для діалогу з користувачем додатку. Вміти користуватися механізмом обробки виключенью.

ОРН3. Знати характеристики, властивості модальних і немодальних діалогових вікон. Вміти розробляти додатки з використанням модальних і немодальних діалогових вікон.

ОРН4. Знати основні елементи управління, їх властивості, призначення. Вміти програмувати з використанням елементів управління, функцій, класів. Вміти застосовувати стандарні вікна відкриття і збереження файлівю

ОРН5. Знати індикаторні елементи управління, їх властивості, призначення. Вміти програмувати з використанням індикаторних елементів управління, додавати меню, застосовувати файли довільного доступую

ОРН6. Вміти розробляти багатовіконні додатки з використанням файлів довільного доступу.

В якості мови програмування використовується мова C#, що входить в комплект мов середовища програмування Visual Studio 2019. В додатках наведені відомості про компоненти, які використовуються під час викладення матеріалу, а також відомості про властивості та застосування стандартних вікон операційній системі Windows. Список літератури містить підручники, які дозволять студенту самостійно розширити об’єм знань в області програмування в операційній системі Windows.

Матеріал посібника може бути повністю або частково використано під час виконання лабораторних робіт.

ТЕМА 1 СЕРЕДОВИЩЕ VISUAL STUDIO. КОНСОЛЬНІ ДОДАТКИ

Призначення середовища програмування

Середовище програмування ***Visual Studio*** призначене для розробки програм мовами високого рівня. Використовуючи це середовище, можна розробляти програми такими мовами як ***Basic, C#, C++, F#.***

Запуск середовища

Запустити середовище можна через кнопку ***«ПУСК»*** в меню ***«Программи»***. Порядок дій: ***Пуск – Всі програми – Microsoft Visual Studio***.

Початок роботи

Після запуску на екрані з’явиться стартове вікно середовища Visual Studio, яке має наступний вигляд (рис.1.1) [2, 6]:

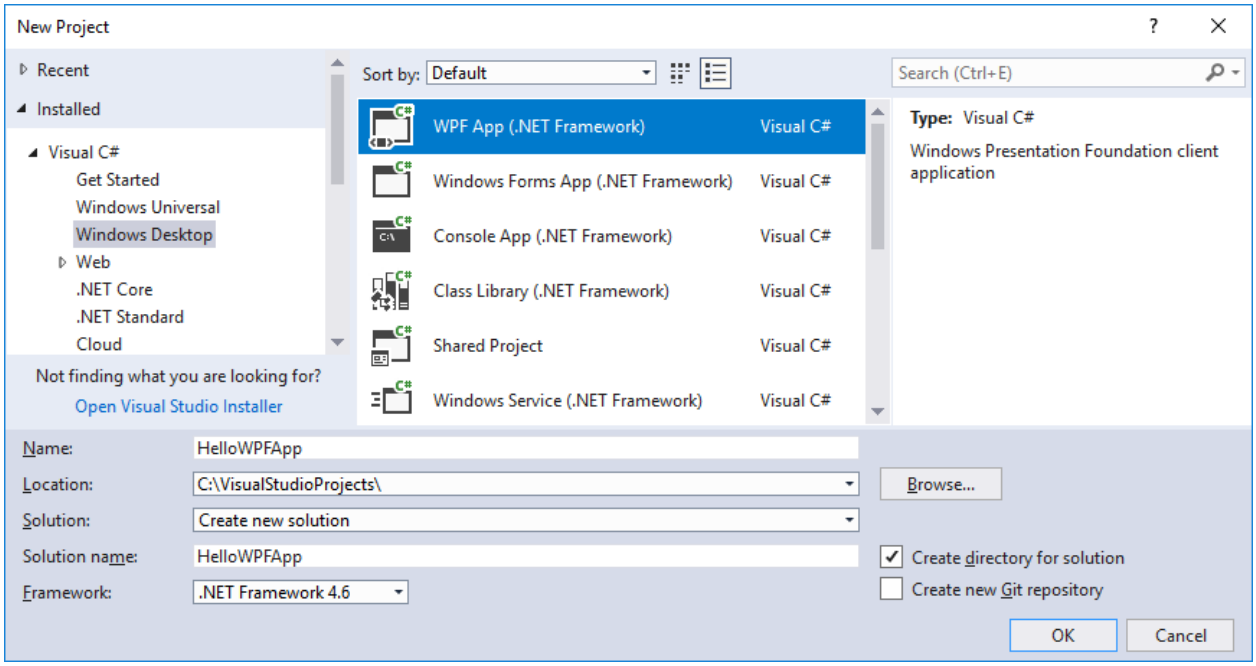

Рисунок 1.1 – Стартове вікно середовища

Стартове вікно містить дві частини: стартова сторінка і оглядач рішень. На стартовій сторінці є корисні можливості. Використовуючи посилання

Відкрити проєкт, програміст може продовжити роботу з одним із додатків, який він розробив раніше. Якщо необхідно створити новий додаток, то можна скористатися посиланням ***Створити проєкт***. Закрийте стартову сторінку. Зверніть увагу на головне меню програми. Лінійка меню містить можливості, з якими будемо познайомимося пізніше.

Вікно справа називається ***Оглядач рішень***. Тут буде відображатися структура додатка. Поки вікно пусте. Воно буде наповнюватися під час включення в додаток різноманітних компонентів. Це вікно являється службовим. Оберемо в головному меню закладку ***Вид***. Випадаючий список містить різні назви службових вікон, серед яких є і вікно стартової сторінки. Відобразимо вікно ***Вывод***. Закріпимо його в нижній частині екрана. Тут будуть відображатися результати роботи програміста: протокол створення програми, повідомлення про синтаксичні помилки, попередження та інше.

Вибір варіанта власного проекту

Кожний додаток розробляється в середовищі ***Visual Studio*** як проект. В одному додатку може бути кілька проектів. Створимо проект, в якому будуть міститися всі компоненти, необхідні для роботи програми. Це можна зробити, використовуючи стартову сторінку, а можна скористатися головним меню. Виконаємо дії: ***Файл – Створити – Проєкт***. На екрані появиться вікно. Ліва секція в цьому вікні (рис. 1.2) містить список можливих проєктів. Обираємо мову C#, зазначаючи при цьому, що розробка буде проводитися для роботи під Windows [6].

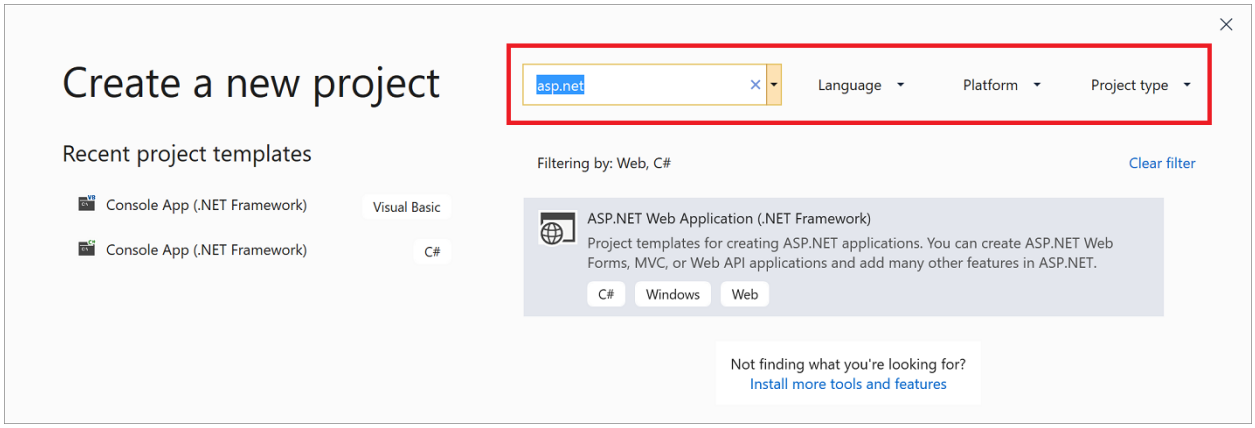

Рисунок 1.2 – Список проєктів

В центральній секції вікна (рис. 1.3) перераховані типи шаблонів проектів [2].

Мовою C# можна розробляти додатки різних призначень. Наприклад:

- ***Додаток Windows Forms.*** Це програма, яка використовує всі можливості ОС Windows, такі як: багатозадачність, графічний багатовіконий режим, управління подіями.
- ***Консольний додаток.*** Це додаток, який працює в середовищі Windows в консольному режимі, коли для введення і виведення даних використовується текстовий режим одного вікна. Робота такої програми схожа на роботу в середовищі DOS.
- ***Бібліотека класів.*** Це не додаток в звичайному сенсі, а динамічна бібліотека (dll).
- ***Служба Windows.*** Це проект для створення служб Windows.[8]

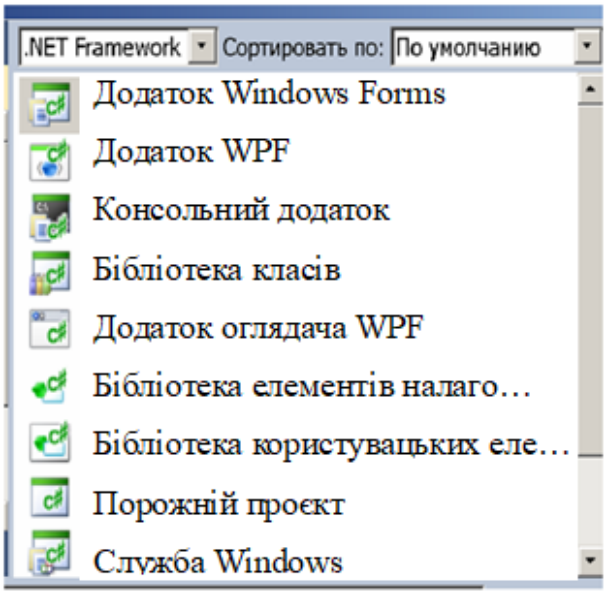

Рисунок 1.3 – Призначення додатка

Ім’я та місце розташування додатка

В нижній секції вікна є три поля для введення даних [2]:

Ім’я. Це поле імені проекту. Щоб задати ім’я проекту, можна використовувати літери (кирилицю і латиницю) і цифри.

Ім’я рішення. Це ім’я додатка, який створюється. Якщо в додатку один проект, то імена співпадають.

Розміщення. В цьому полі указується місце розміщення додатка. Указується повний шлях. Замість набору можна використати кнопку ***«ОГЛЯД»*** і обрати потрібне місце. Можна рекомендувати вибір місця розміщення з точністю до деякої папки, а потім врахувати, що для додатка в обраній папці буде створена нова папка (галочка біля прапорця ***«Створити каталог для рішення»***).

Створення консольного додатка

Оберемо тип проекту ***Консольний додаток***. Залишимо те ім'я, яке запропонує програмне середовище: ***ConsoleApplication1***. В якості місця розміщення обираємо загальну папку студентів ***D:\USERS*** або особисту папку студента. Натискаємо кнопку ***«ОК»***. Програмне середовище відкриє вікно с заготовкою консольного додатка. Тепер у вікні ***Оглядач решень*** видно склад нашого проекту. Майстер проекту розмістив в них такі компоненти, яких достатньо для організації консольного додатка. Кожне рішення в C# може містити кілька проектів. В нашому випадку – це один проект під назвою ***ConsoleApplication1***. В складі проекту вже є один програмний файл – ***Program.cs***. Тип файлу (***cs***, сі-шарп) визначає його як файл, що містить код мовою C#. Вміст цього файлу відображено в лівій секції вікна.[7]

Закриємо програмне середовище. Знайдемо, де саме на жорсткому диску розмістилося наше рішення. Через провідник або ***Мій комп'ютер*** відкриємо папку ***USERS*** і знайдемо папку ***ConsoleApplication1***. Розкриємо цю папку. В ній побачимо ще одну папку ***ConsoleApplication1*** і один файл с розширенням ***sln***. Саме цей файл містить інформацію про рішення. Його можна використовувати як файл, що запускає рішення. Двічі клацнемо лівою кнопкою мишки: рішення знову розкриється.

Знову закриємо вікно рішення.

Звернемо увагу на інший файл: ***Program.cs***. Це файл, який містить код мовою C#. Спробуємо використати його в якості стартового – двічі клацнемо його мишкою. Відкриється тільки сам файл, але не рішення. Закрийте файл і знову відкрийте рішення. Перейдемо до розгляду заготовки в файлі ***Program.cs.***[10]

Вміст заготовки файлу

Зверніть увагу на структуру інформації в програмному вікні. Виділено кілька блоків. Кожен блок можна відобразити в розгорнутому (–) або зжатому (+) вигляді. Це зручно, коли маємо великий програмний текст. Частину блоків можна згорнути, щоб не мішали, а розкрити тільки той блок, що потрібний. Розгорнемо всі блоки і розглянемо кожний з них докладніше.

В першому блоці описані чотири рядки:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

Слово ***using*** використовується для опису системних можливостей, які надані програмісту. Всі мови програмування мають системні можливості, що оформлені в вигляді системних бібліотек. В мові C# використовуються особлива термінологія для опису системних можливостей. В цьому випадку мова йде про простір імен. Цей термін має своє інтуїтивне пояснення. Кожна вбудована можливість має деяке ім’я, а ряд близьких за сенсом можливостей об’єднуються в групи. Ряд імен можливостей існують в деякому просторі (***namespace***). Прикладом такого простору є простір ***System***. Це основний простір імен. Поставте в першому рядку після слова ***System*** крапку і подивіться, що буде. У вікні, що випадає, є ще багато уточнень, які, в свою чергу, можуть бути також іншими просторами імен або іменами класів, які також містять у собі вбудовані засоби мови. [7]

Три інші рядки цього блока використовують підпростори імен простору ***System***. Описання просторів дозволяють викликати для виконання вбудованих можливостей за їх іменами, тому що самі простори вже описані. Наприклад, всередині простору ***System*** є простори імен ***Linq***, ***Text*** і ***Collection***, а в останньому – простір імен ***Generic***. Які саме можливості будуть потрібні, визначає сам програміст. Він підключає в своїй програмі потрібні йому простори імен. Наша заготовка створена середовищем програмування, яке у відповідності до обраного типу проекту (***консольний додаток***) підключило саме ці простори імен. [7]

Перейдемо до розгляду другого блока. Він називається так, як і наш проект – ***ConsoleApplication1***. Перед ним розміщено слово ***namespace***. Це

значить, що середовище автоматично створило в нашій програмі новий простір імен. Тепер всі імена (ідентифікатори), які ми будемо використовувати в тексті, будуть автоматично включені в простір імен ***ConsoleApplication1***. Таким чином, в процесі роботи будуть використовуватися імена і системних просторів, і простір імен ***ConsoleApplication1***. Середовище програмування буде «підказувати» нам імена з усіх просторів імен, що використовуються. Перевіримо це експериментально. В просторі імен ***ConsoleApplication1*** є опис класу з іменем ***Program***.

Будь-який клас – це структурна одиниця програми, її окремий модуль, де описуються алгоритми.

Опис алгоритмів оформлено у вигляді функцій. Кожна функція має ім'я, яке використовується підчас виклику функції для виконання. Власне, саме імена класів і функцій і визначають функціональність простору імен. Зазначимо, що замість терміна ***«ім'я функції»***, зазвичай використовується термін «метод».

Наприклад, в складі класу ***Program*** є метод ***Main***. Це особливий метод. Він є в будь-якій програмі, причому зустрічається один раз. Це головний метод програми. Виконання будь-якої програми починається саме з нього. Крім нього в програмі ми можемо описати інші методи, в різних класах. Переконаємося, що система «бачить» простір імен ***ConsoleApplication1***. Для цього установимо курсор всередині методу ***Main*** (між фігурними дужками) і наберемо фразу ***ConsoleA***. Зверніть увагу, що в процесі набору система намагається підказувати різні продовження набору. Коли ми наберемо фразу ***ConsoleA***, то побачимо підказку. Оберемо її мишкою і натиснемо кнопку ***«ENTER»***, щоб слово ***ConsoleApplication1*** з'явилося в тексті програми. Затим натиснемо клавішу ***«КРАПКА»***. Система підказує, що в просторі ***ConsoleApplication1*** є клас ***Program***. Повторимо ще раз: мишкою оберемо ім'я класу, натиснемо кнопку ***«ENTER»***, а потім знову клавішу ***«КРАПКА»***. І знову побачимо підказку: можна обрати метод ***Main*** і ще пару методів. Висновок очевидний: при розробці програмного коду програміст постійно отримує дружню підтримку системи. [7]

Набір тексту програми

Текст програмного коду в консольному додатку повинен розміщуватися всередині методу ***Main*** між фігурними дужками. Наберемо у вікні файлу ***Program.cs*** текст. Не рекомендується копіювати наведений текст – економія часу може зробити ведмежу послугу, тому що при копіюванні ми не побачимо, як середовище розуміє текст. При наборі різні слова будуть зафарбовуватися різним кольором, а щось середовище буде нам підказувати. В середині методу ***Main*** набираємо наступний текст:

```
int a,b,c1; // оголошення змінних цілого типу
a=Console.ReadLine(); // введення значення з клавіатури
b=3; // привласнення
c1=a+b; // обчислення
// виведення результатів на екран
Console.WriteLine("Сума={0}",c);
```

Різні фрагменти тексту будуть відображуватися різним кольором. Зелений колір використовується середовищем програмування для відображення коментарів, синій – для службових слів. Наприклад, слово ***int*** вказує, що три імені ***(a, b, c1)*** – це імена змінних, які при виконанні програми будуть містити значення цілого типа. Голубим кольором відображаються імена класів. Червоний колір – це колір для відображення рядків.[7]

Суть програми, яку зараз набрали, достатньо проста. Оголошується, що буде використано три змінних цілого типа. Дві з них отримують числові значення: значення першої вводиться з клавіатури, а друга отримує значення під час виконання програми. Значення третьої змінної обчислюється як сума двох інших. Отриманий результат виводиться на екран, після чого програма завершає роботу.

Підготовка програми до виконання

Написати текст програми – ще не означає, що програма уже створена. Важливо написати її так, щоб в ній не було помилок. Середовище програмування частину програмного коду підкреслює хвилястою лінією – це свідчить про наявність помилок. В другому рядку підкреслено майже весь оператор, в п’ятому – змінна ***c***. Найпростіший спосіб перевірити нормальність

написаного коду – це спробувати виконати програму. Середовище програмування перед запуском програми обов'язково перевірить програму на наявність помилок і повідомить про о них. Але можна спробувати розібратися з помилками до запуску. Для цього розглянемо підкреслені елементи.

Встановимо курсор мишки на змінну c в п'ятому рядку тексту. На екрані з'явиться підказка: ***«Елемент 'c' не існує в поточному контексті»***. Як це розуміти? Справа в тому, що п'ятий оператор видає на екран обчислене значення змінної ***c***. Але, якщо уважно проаналізувати текст програми, то можна побачити, що в першому рядку оголошені для використання змінні ***a***, ***b*** і ***c1***. Змінної ***c*** серед них немає – її взагалі ніде ніхто не оголошував. Це порушення відзначене середовищем. Після аналізу можна зрозуміти як виправити помилку. В програмі є змінна ***c1***, яка оголошена і використовується в четвертому рядку для підрахунку суми. Очевидно, що цю змінну і потрібно використовувати в п'ятому рядку. Замінимо ***c*** на ***c1***.

Помилка в другому рядку ідентифікується більш складно. При наведенні курсору на цей рядок ми отримаємо значно більший об'єм інформації, ніж в першому випадку. Система прокоментує нам, що собою представляє указана мовна конструкція і перерахує деякі особливості її використання. Але в кінці цієї інформації також буде описана суть помилки: ***«неявне перетворення muny string в int неможливе»***. В чому тут справа? Вираз ***Console.ReadLine()*** викликає для виконання метод ***ReadLine*** з класу ***Console***. Цей метод вводить з клавіатури всі символи, які набере користувач до того, як буде натиснута клавіша ***«ENTER»***. Всі набрані символи будуть сформовані в один рядок, тобто значення типу ***string***. Це значення в другому операторі привласнюється змінній ***a***. Але змінна ***a*** оголошена в програмі як змінна типу ***int***, тобто як цілочислова. Таке привласнення заборонено: неможна цілочисловій змінній привласнити рядкове значення. Перед привласненням необхідно виконати перетворення. Значить, цю помилку потрібно також виправляти програмісту. Це зробити нескладно, якщо знати, що в мові є відповідні можливості. В просторі імен ***System*** є клас ***Convert***, в якому зберігаються різноманітні методи перетворення рядкових значень в інші типи даних. Існує метод ***ToInt32***, який як раз і виконує перетворення рядка в ціле число. Скористаємося цим і замінимо другий рядок тексту на наступний:

```
a = Convert.ToInt32(Console.ReadLine());
```

Зазначимо, що текст більше не підкреслено. Схоже, що помилок немає. Можна спробувати виконати програму.

Виконання програми

Для виконання програми скористаємося комбінацією клавіш ***«CTRL-F5»***. Отримаємо чорне пусте вікно. Це вікно, в якому відображаються всі результати виконання нашої програми. Так як наша програма розроблена як консольний додаток, то ми бачимо на екрані саме таке вікно. Але, воно ще пусте: програма тільки почала виконання і ще не завершилась. Вона чекає. Згадаємо, що другий рядок програми – це введення значення з клавіатури. Значить, потрібно це значення ввести. Введемо, наприклад, число – 8 і натиснемо клавішу ***«ENTER»***. Відобразиться наступне вікно, в якому буде видно результат виконання оператора введення, виведення і рядок, який пропонує для завершення роботи програми натиснути будь-яку клавішу.[8]

Якщо натиснути яку-небудь клавішу, то чорне вікно закриється. Програмне середовище по завершені виконання програми штучно затримала вікно, щоб користувач встиг побачити результати своєї роботи. Рядок завершення видається тому, що ми запустили програму з середовища програмування. Якщо б ми запустили готовий програмний файл, то цього рядка не було б. Перевіримо це.

Відкриємо папку ***D:\USERS\ ConsoleApplication1\ ConsoleApplication1 \bin\debug.*** В цій теці знаходиться готова програма ***ConsoleApplication1*** (***exe***-файл). Спробуємо її виконати. Після запуску ми зможемо ввести дані, але зразу після цього вікно закривається – так швидко вона завершує свою роботу (в тексті програми немає ніяких затримок). А ось при запуску з середовища вікно на екрані затримується до натискання якої-небудь клавіші.

Завдання для самостійної роботи

1. При запуску розглянутого прикладу робота програми починається з відображення пустого вікна. Користувач повинен сам здогадатися, що можна вводить ціле значення. Доопрацюйте текст програми. Видайте на

екран пояснюючі повідомлення. Наприклад, запропонуйте користувачу ввести значення.

2. Розробіть власний простий проект, збережіть його і продемонструйте роботу з ним.

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Призначення середовища програмування ***Visual Studio.***
2. Як здійснюється процес запуску середовища програмування?
3. Перелічіть основні дії на стартовому вікні.
4. Опишіть процес створення консольного додатка.
5. Опишіть структуру консольного додатка.
6. Опишіть вміст заготовки файлу ***Program.cs.***
7. Як підготувати програму до виконання?
8. Назвіть особливості процесу набору тексту програми.
9. Опишіть процес виконання програми.
10. Прихначення методу ***main().***
11. Який метод використовується для введення даних?
12. Наведіть приклади використання методу ***Console.ReadLine()***.
13. Наведіть приклади використання методу ***Console.Read().***
14. В чому поляє різниця між методами ***Console.Read()*** та ***Console.ReadLine()***?
15. Який метод використовується для виведення даних?
16. Наведіть приклади використання методу ***Console.WriteLine()***.
17. Наведіть приклади використання методу ***Console.Write().***
18. В чому поляє різниця між методами ***Console.Write()*** та ***Console.WriteLine()***?
19. Опишіть призначення методів ***Convert.***

ТЕМА 2 СЕРЕДОВИЩЕ VISUAL STUDIO. WINDOWS-ДОДАТОК

2.1 Створення Windows-додатка

Запускаємо середовище ***Visual Studio*** і на стартовій сторінці обираємо посилання ***Створити проєкт***. В розділі шаблонів обираємо ***Додаток Windows Forms***. ***Ім'я проєкта*** і ***додатка*** – ***Проба***. Місце розміщення: ***D:\USERS***. Буде створена заготовка (рис. 2.1).

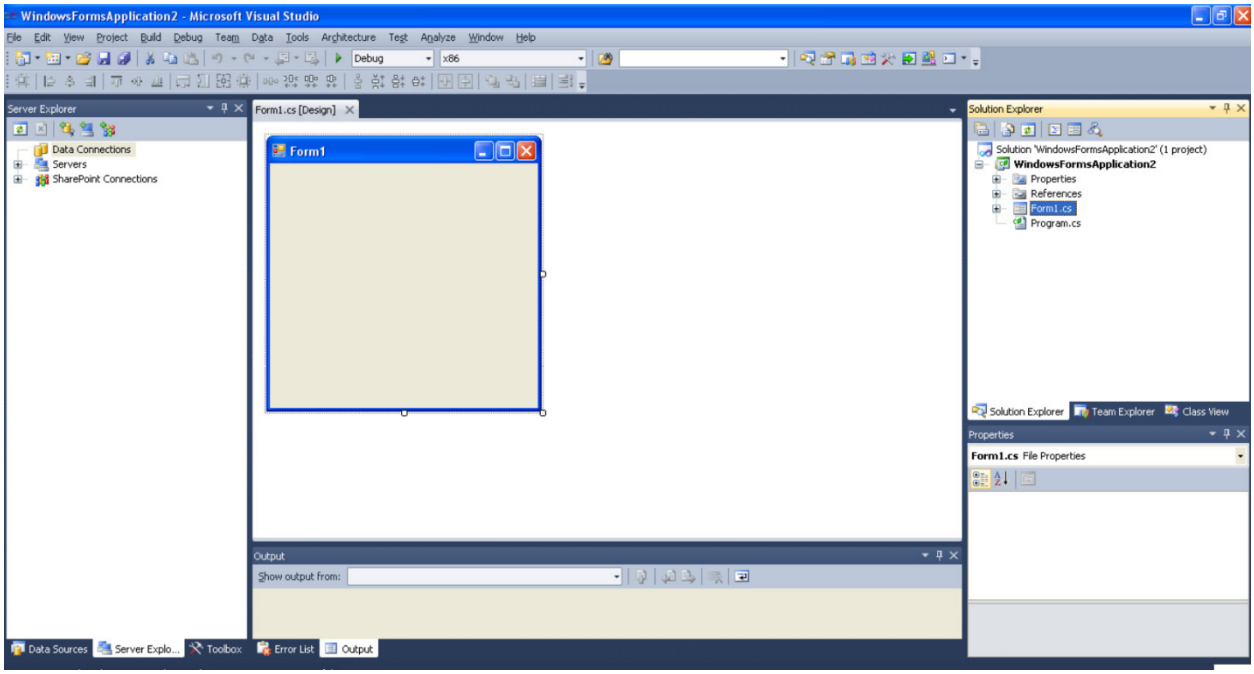

Рисунок 2.1 – Заготовка Windows-додатка

В вікні ***Оглядач рішень*** видно склад нашого проекту. Він не пустий – в папках є компоненти, необхідні для організації ***Windows***-додатка. Кожне рішення в C# може містити кілька проектів. Зараз існує один проект під назвою ***Проба,*** у складі якого є два програмних файла – ***Program.cs*** і ***Form1.cs.***[7]

2.2 Склад заготовки

На екрані зліва ми бачимо відображення складу файлу ***Form1.cs***.

Зверніть увагу на закладку, що відповідає цьому відображенню. Закладка позначена словом ***[Конструктор]***. Файл відображається візуально. Показано, які об'єкти в ньому використовуються. Зараз в ньому немає ніяких

об’єктів, крім самого вікна. Установимо мишку на вікно і натиснемо праву кнопку. Відобразиться відповідне контекстне меню. [7]

Оберемо режим ***Перейти до коду***. З’явиться нова закладка з іменем ***Form1***, але без позначки ***Конструктор*** (рис. 2.2). Відображено склад файлу ***Form1.cs***, але уже не у вигляді об’єктів, а у вигляді програмного коду мовою C#. Операторів, що підключають до програми системні простори імен, тут значно більше, ніж в консольному додатку. Є вже відомий нам простір імен ***System***. Інші рядки підключають підпростори імен простору ***System***. Є серед них і простір ***System.Windows.Forms***, який містить все необхідне для розробки вікон ***Windows***-додатка.[7]

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace PROBA
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Рисунок 2.2 – Програмний код Form1

Є простір імен нашої програми – ***PROBA***. В ньому є опис класу з іменем ***Form1***. Описи алгоритмів оформлені у вигляді функцій (методів). [1]

Наприклад, у складі класу ***Form1*** уже є метод ***Form1()***. Це особливий метод: його називають конструктором об’єктів класу. В даному випадку – це конструктор вікна ***Form1***. Що він містить і як саме він будує об’єкти – буде вивчено пізніше.

Крім конструктора в складі класу будуть у інші методи, які напише програміст. Всі ці методи будуть мати відношення саме до вікна ***Form1***.

Розглянемо модуль ***Program.cs***. Оберемо його мишкою у вікні ***Оглядач рішення***. В цьому модулі є клас – ***Program***. Він містить метод ***Main()***. Але, на відмінність від консольного додатка тут цей метод не пустий – в ньому вже є три оператори. Якщо послідовно навести курсор мишки на кожний оператор, то можна отримати відомості про призначення кожного оператора. [1]

В цілому сенс операторів такий. Два перших оператори дозволяють використовувати візуальні стилі в процесі виконання програми і задають деякі стандартні режими управління програмою. Третій оператор забезпечує запуск процесу, який пов'язаний з вікном ***Form1***. Для цього використовується метод ***Run()*** з класу ***Application***. В круглих дужках указано ім'я того об'єкта, з яким пов'язаний процес. А тепер за допомогою закладки ***Form1.cs [Конструктор]*** переключимося в вікно візуального відображення форми ***Form1***. [7]

Установимо курсор мишки на вікні, викличемо контекстне меню і оберемо ***Свойства***. Відобразиться додаткове вікно властивостей форми. Властивостей багато, вони різні, знайомитися с ними треба поступово, по мірі необхідності.[7]

Розглянемо властивість ***Name***, яка визначає ідентифікатор вікна. Фактично це ім'я змінної: ***Form1***. Саме це ім'я буде використовуватися в програмному коді при зверненні до об'єкту вікно ***Form1***.

Властивість ***Text*** містить заголовок вікна. Її можна змінити, це приведе до зміни назви вікна.

Властивість ***MaximizeBox*** дозволяє заборонити розгорнути вікно на повний екран, якщо установити значення ***False***.

Завдання для самостійної роботи

*Проекспериментуйте з установкою різних значень у властивостях форми. Спробуйте експериментально зрозуміти сенс різних значень. Щось можна побачити зразу при змінені значення властивості, а щось – тільки після запуску програми. Спробуйте змінити значення деяких властивостей, запустивши програму (**Ctrl-F5**), подивіться, як різні значення властивостей проявляються при виконані програми.*

Дослідіть:

*1. **AutoSizeMode** (переконайтеся, що при одному із значень вікно неможливо розтягти).*

*2. **BackColor** (установіть фоновий колір).*

*3. **BackgroundImage** (встановлює фонову картинку).*

*4. **ControlBox** (при установці значення false у вікна відсутні кнопки управління; завершити програму тепер можна тільки через диспетчер задач Windows).*

*5. **Cursor** (змінюється зовнішній вид курсору мишки).*

*6. **Enabled** (забороняє доступ до елементів вікна – якщо вони там є).*

*7. **FormBorderStyle** (різноманітні види оформлення границь вікна).*

*8. **Icon** (зміна значка-іконки в заготовочному рядку вікна).*

*9. **ShowIcon** (показати або приховати іконки).*

*10. **MaximizeBox** (активізація або блокування кнопки розгортання вікна на повний екран).*

*11. **MinimizeBox** (активізація або блокування кнопки згортання вікна).*

*12. **Opacity** (ступінь прозорості вікна).*

*13. **Size** (установка розмірів вікна).*

*14. **Text** (установка надпису в заголовок вікна).*

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Назвіть різновиди програм C#.
2. Опишіть процес створення ***Windows***-додатку.
3. Перелічіть основні властивості форми.
4. Як змінюється робота форми при зміні цих властивостей.
5. Які компонети має проєкт ***Windows***-додатку.
6. Вміст та призначення компонета ***Form1.cs [Конструктор]***.
7. Вміст та призначення компонета ***Program.cs.***
8. Вміст та призначення компонета ***Form1.cs.***
9. Назвіть освні події, пов’язані з формою.

ТЕМА 3 КОМПОНЕНТИ ВІКНА. СТРУКТУРА КЛАСУ FORM1

3.1 Організація класу Form1

Створимо нове рішення з іменем ***Задача1.*** Заготовка має вікно ***Form1***. Відобразимо програмний код, що відповідає цьому вікну. Зазначимо, що у відповідності з назвою задачі простір імен також називається ***Задача1***, а клас, що відповідає вікну, називається ***Form1***. Клас містить лише один метод з іменем ***Form1()*** – це конструктор об'єктів класу. Всередині методу в фігурних дужках написано вираз:

```
InitializeComponent();
```

Це вираз являється оператором мови C#. Ім'я ***InitializeComponent()*** – це ім'я методу [7]. Даний оператор викликає метод для виконання. Те, що це саме метод, можна встановити за круглими дужками – для виклику методу використовується саме така форма: ім'я з круглими дужками. В круглих дужках при виклику методу можуть бути записані якісь значення або імена змінних, але в нашому випадку не вказано нічого – тому що просто не потрібно. Метод, що викликається, повинен бути десь описаний – адже якщо метод викликається для виконання, то повинно бути відомо, що буде зроблено, який саме алгоритм або алгоритми будуть виконані.

Оголошення може бути вбудованим в систему або наперед описаним програмістом в його програмі. В нашому випадку алгоритм методу ***InitializeComponent()*** не являється системним, його оголошення знаходиться в нашій програмі. Питання: де саме? Звернемо увагу на заголовок оголошення класу:

```
public partial class Form1: Form
```

В заголовку присутнє слово ***partial***, воно означає ***«частковий»***. Це значить, що опис класу в цьому модулі наведено не повністю. Наведена тільки частина опису класу, а десь є іще одна частина. Спробуємо відшукати другу частину опису класу. В вікні ***Обозреватель решений*** розкриємо плюс біля імені файлу ***Form1.cs***. Ми побачимо, що є ще один модуль, який має відношення до вікна ***Form1***: ***Form1.Designer.cs***. Подвійним кліком мишки

розкриємо програмний модуль ***Form1.Designer.cs.*** Саме тут і міститься друга частина опису класу ***Form1*** [7]. Зверніть увагу на рядок:

```
partial class Form1
```

Це означає, що ми бачимо ще одну частину класу ***Form1***. Давайте ще раз перерахуємо, що ми встигли зробити:

- замовили програмному середовищу створення проекту ***Задача1***;
- вказали, що проект повинен бути ***Windows-додатком***;
- вказали, що місцем його розташування буде папка ***USERS***.

Після цього отримали проект з одним вікном. Вікно, правда, доки є пустим. Але в склад нашої програми середовище програмування уже включило два модулі: ***Form1.cs*** і ***Form1.Designer.cs***. Обидва імені ми бачимо в вікні ***Оглядач рішень.*** При автоматичній побудові нашої програми середовище створило опис об'єкта Form1 у вигляді класу з таким же ім'ям. При цьому опис класу розділено на два модулі. Чому? Необхідно це зрозуміти і запам'ятати. Середовище програмування «прийняло мудре рішення», розділивши клас на два модулі.

1. Перший модуль ***Form1.Designer.cs*** використовується самим середовищем для автоматичного розміщення програмного коду, який пов'язаний з об'єктом. Зокрема, при створені додатка середовище створило вікно, включивши в цей модуль всі оператори, що необхідні для відображення вікна на екрані. [7]

2. Другий модуль ***Form1.cs*** призначений для програміста, тобто для нас. Тут ми будемо розміщувати програмний код, який напишемо самі. [7]

А тепер поглянемо, що саме добавила система. Відкриємо закладку ***Form1.Designer.cs***. Знайдемо сірий рядок:

Код, що автоматично створений конструктором форм Windows.

Розкриємо плюс проти нього. Відкриється секція програми, обмежена фразами ***#region*** і ***#endregion.*** Фрази, що починаються знаком #, називаються директивами. Ці директиви обмежують область програмного коду, який створено автоматично самим середовищем програмування. Зверніть увагу на наступний фрагмент коду:

```
private void InitializeComponent()
{                    this.components=                    new
System.ComponentModel.Container();
this.AutoScaleMode=
System.Windows.Forms.AutoScaleMode.Font;
this.Text = "Form1";
}
```

Це і є опис методу, ім'я якого ***InitializeComponent()***. Всередині опису методу між фігурними дужками записані оператори, які «рисують» вікно на екрані. Спробуємо зрозуміти сенс цих операторів.

Перший оператор оголошує поточне вікно контейнером. Це означає, що вікно може містити в собі інші об'єкти. Другий оператор забезпечує автоматичне масштабування. Наприклад, зміна розмірів тексту при зміні розмірів вікна. Третій оператор задає назву вікна, яка відображається в заголовку. Назва задається через властивість ***Text***. Цю властивість можна установить і вручну через вікно властивостей. [7]

Таким чином, в модулі ***Form1.cs*** в методі-конструкторі викликається метод ***InitializeComponent()***. При його виклику виконуються ті самі три оператори, які знаходяться в оголошені методу в модулі ***Form1.Designer.cs.*** При виконанні операторів на екрані з'являється програмне вікно.

Тепер спробуємо наповнити наше вікно іншими об'єктами. Для цього скористаємося панеллю елементів.

3.2 Панель елементів

Відкриємо закладку з візуальним відображенням вікна ***Form1.cs [Design]***. Оберемо мишкою вікно, а потім використаємо головне меню програмного середовища: ***Вид – Панель елементів***. На екрані з'явиться ще одне вікно – ***Панель елементів*** або, як її називали в більш ранніх версіях середовища, ***Панель інструментів***. Інструменти, представлені на цій панелі, являються вбудованими компонентами середовища програмування. За допомогою цих інструментів можна включити в вікно різні об'єкти: поля введення даних, поля надписів, кнопки управління, прапорці, перемикачі та інші компоненти. [7]

Принцип використання будь-якого інструмента простий. Достатньо захватити його мишкою і перенести на вікно. На вікні з'явиться зображення об'єкта, зазвичай в тому вигляді, в якому його буде видно користувачу. Програміст переносить ті об'єкти, які, як він вважає, потрібні для розв'язання задачі. Він розміщує їх у вікні так, щоб користувачу було зручно з ними працювати, і щоб дизайн вікна був на рівні. Перенесемо на вікно компонент ***Label***. Після переносу цей стандартний компонент відобразився як об'єкт з іменем ***label1***. Це ім'я ми будемо використовувати для звернення до об'єкта в програмному коді.

Об'єкт зазвичай використовується для зберігання різноманітних надписів, що пояснюють хід роботи. оберемо мишкою цей об'єкт. Потім викличемо контекстне меню і оберемо ***Властивості***. Справа з'явиться вікно властивостей об'єкта ***label1***. Властивість ***Text*** уже має значення ***label1***. Саме цей текст і записано в полі об'єкта на вікні. Якщо значення цієї властивості змінити на інший текст, то в полі об'єкта буде відображатися інший текст. Це означає, що об'єкт можна використати для відображення вихідних даних і результатів обчислень. [7]

Завдання для самостійної роботи.

*1. Заповніть вікно **Form1** так, щоб в ньому розмістилися поля для завдання і відображення характеристик геометричного тіла – циліндра. В програмному середовищі це вікно повинне відображатися так, як показано на рис. 3.1.*

*Всі надписи повинні бути реалізовані як об'єкти **label**. Очевидно, що їх 11 штук. Вони мають імена от **label1** до **label11**. Що якому імені відповідають, легко встановити за рисунком:*

- заголовок – це **label1**.
- заголовки зліва (радіус, площа і т.д.) – це **label2, …, label6**.
- тексти справа – це **label7, … label11**.

*Властивість **Text** кожного об'єкта мають значення у відповідності з текстом на екрані. **«Вихідні дані і результати»** – це об'єкт **groupBox**. Він використовується для того, щоб об'єднати в один блок близькі за сенсом об'єкти заради зручності користувача.*

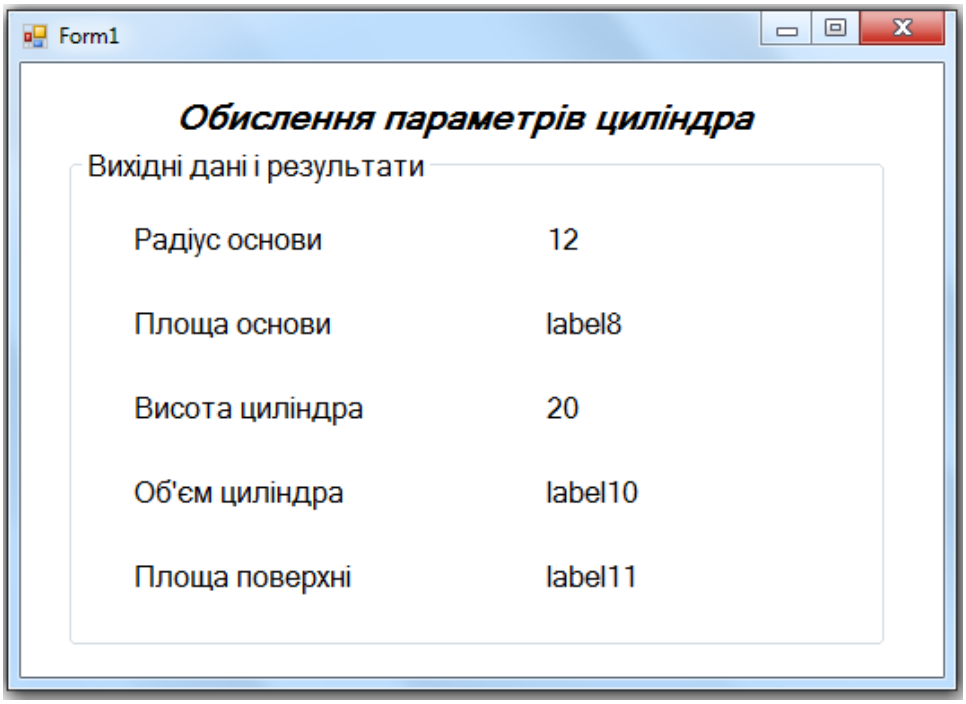

Рисунок 3.1 – Вікно задачі

*2. Зверніть увагу, як змінився склад модуля **Form1.Designer.cs** після завершення оформлення вікна. В нього додались секції створення і оформлення кожного об'єкта, який добавлено в вікно.*

*3. Проаналізуйте секцію коду, що відноситься, наприклад, до **label7**. Кожний з операторів починається зі слова **this** (цей). Це означає, що оператор відноситься до об'єкта поточного класу. Поточний клас – **Form1**. Значить, мова йде про склад класу **Form1**. Далі, через точку записано ім'я об'єкта, для якого записано оператор – це **label7**. Далі через точку записано ім'я властивості, якій привласнюється значення. Так програмним шляхом можна змінити властивість об'єкта. Таким чином, якщо властивість **AutoSize** приймає значення **true**, то розмір об'єкта буде змінюватися автоматично в залежності від довжини тексту в надписі.*

*Далі, встановлюється шрифт і колір. Властивість **Location** містить координати точки лівого верхнього кута об'єкта. Властивість **Name** визначає ім'я змінної, яка використовується при зверненні до об'єкта (не змінюйте її, якщо не бажаєте зайвих помилок в програмі!). Поточний розмір об'єкта визначає властивість **Size**.*

*Властивість **Text** містить те значення, яке ми бачимо на екрані. З властивістю **TabIndex** познайомимося пізніше.*

4. Запустіть програму і подивіться, як це виглядає з точки зору користувача. Зараз забезпечено тільки відображення вихідних даних (радіус і висота), які задані в ручному режимі через властивість ***Text****. Результатів обчислень немає.*

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Призначення методу ***InitializeComponent()***.
2. Призначення та вміст модулю ***Form1.Designer.cs.***
3. Призначення та вміст модулю ***Form1.cs.***
4. Призначення ***Панелі елементів (інструментів)***.
5. Назвіть групи, на які ділиться ***Панелі елементів***, призначення кожної з них.
6. Які інструменти з ***Панелі елементів*** найчастіше використовуються? Дайте характеристику їх основних властивостей.
7. Пизначення та основні властивості об’єкту ***groupBox.***
8. Пизначення та основні властивості об’єкту ***label.***
9. Назвіть принцип використання будь-якого інструмента.
10. Призначення та використання директив ***#region*** і ***#endregion.***

ТЕМА 4 ПОДІЇ, ПОВ'ЯЗАНІ З ВІКНОМ. МЕХАНІЗМ ОБРОБКИ ПОДІЙ

4.1 Завантаження форми

Будемо використовувати рішення, що отримали під час вивчення попередньої теми. Зробіть копію папки з програмою (оригінал ще знадобиться під час вивчення Теми 5). Дайте їй ім'я ***Задача1_Т4.*** Відкрийте скопійований проект. Упевніться, що відкрилося таке ж саме вікно, як і фінальне вікно попереднього проекту.

Запустимо програму (***Ctrl-F5***). У вікні містяться тільки вихідні дані: радіус основи циліндра і його висота. А результатів немає. Це і зрозуміло: вихідні дані ми задали вручну, установивши властивість ***Text*** для об'єктів ***label7*** і ***label9***. Тепер хотілося б, щоб обчислення по заданим даним були виконані програмно.

Ми запускаємо програму і хочемо бачити результати. Очевидно, що коли вікно з'являється на екрані, результати в ньому уже повинні бути. Постає питання: коли, в який момент часу повинні виконуватися розрахунки? Розв'язок може бути знайдено, якщо розуміти, як здійснюється виконання програми. Виконання програми – це процес. Під час цього процесу проходить багато подій. Наприклад, таких [7]:

1. Операційна система (ОС) отримала замовлення на запуск програми.
2. ОС розмістила програму в оперативній пам'яті.
3. Програма отримала від ОС команду ***«Працюй»***.
4. Програма почала виконуватися.
5. Програма «зобразила» на екрані вікно разом з усіма об'єктами.
6. Програма чекає реакції користувача.
7. Користувач натиснув на кнопку закриття вікна, тому що невдоволений відсутністю результатів.
8. Програма завершила роботу і повідомила про це ОС. ОС вивільнила пам'ять, яку займала програма під час виконання.

Чого не вистачає серед цих подій? Очевидно, що відсутня подія ***«обчислення результатів»***. Коли ця подія повинна відбутися? Мабуть перед

подією № 5. Адже «зображувати» вікно на екрані має сенс тільки тоді, коли вже є що зображувати! Постає питання: «куди і як» потрібно вставити в програму команди-оператори, що виконують обчислення? В нашій програмі є два модулі для програмного тексту. Один з них (***Form1.cs***) призначений для написаного програмістом програмного коду. А питання «як» легко вирішується за допомогою програмного середовища. [3]

Виконаємо подвійний клік мишкою на вікні, але не де попало, а на вільному місці: не всередині об'єкта ***GroupBox1*** і не на будь-якому об'єкті ***Label*** – в цьому випадку система «вважатиме», що наш подвійний клік відноситься до цих об'єктів.

Наприклад, на сірому фоні трохи вище об'єктуа ***label1***. Після подвійного кліку відобразиться програмний текст (рис. 4.1).

```
namespace Задача1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs
e)
        {

        }
    }
}
```

Рисунок 4.1 – Вікно Form1.cs

Зверніть увагу: середовище саме розкрило модуль ***Form1.cs*** і додало в програму код [7]:

```
private void Form1_Load(object sender, EventArgs e)
{
 }
```

Цей код – заготовка методу під назвою ***Form1_Load()***. Чому така назва і для чого зроблена заготовка? Справа в тому, що з об'єктом ***Form1*** пов'язана вбудована подія, яка має ім'я ***Load*** («завантаження»), а це означає, що метод с іменем ***Form1_Load()*** буде виконано в той момент виконання програми, коли вікно вже готове до відображення, але ще не відображене на екрані. Тобто, ті

команди-оператори, які будуть записані в цьому методі (між фігурними дужками), виконаються до появи вікна на екрані.

Таким чином, ми отримали відповідь на питання «куди». Саме в цей метод потрібно записати команди-оператори, що обчислюють площу основи, об'єм і площу повної поверхні циліндра. Зверніть увагу: курсор установлено системою саме в те місце, куди потрібно записати оператори. Згадаємо формули.

Площа основи: $S = \pi R^2$.

Об'єм циліндра: $V = S \cdot h$.

Повна поверхня: $P = 2\pi Rh + 2S$.

Всі значення, що використовуються – дійсні числа. Оголошуємо п'ять змінних: для радіуса, висоти, площі, об'єму і поверхні [3]:

```
double R, h, S, V, P;
```

Радіус і висота задані в об'єктах ***label7*** і ***label9*** через властивість ***Text***, а це рядкове значення – тип ***string***. Таким чином, ці значення потрібно перетворити в дійсне число. Вбудований клас ***Convert*** містить потрібні методи перетворення, для даного випадку метод ***ToDouble()***:

```
R=Convert.ToDouble(label7.Text);
h=Convert.ToDouble(label9.Text);
```

Тепер можна записати формули. Використаємо число π – константу з вбудованого класу ***Math***:

```
S=Math.PI * R * R;
V=S * h;
P=2 * Math.PI * R * h + 2 * S;
```

Щоб результати відобразилися на екрані, обчисленні значення потрібно розмістити в відповідні об'єкти ***Label***. Привласнимо обчислені значення властивості ***Text*** кожного об'єкта, скориставшись методом ***Format()*** з класу ***string***:

```
label8.Text = string.Format("{0,10:#.##}",S);
label10.Text = string.Format("{0,10:#.##}",V);
label11.Text = string.Format("{0,10:#.##}",P);
```

Запустимо програму (рис. 4.2).

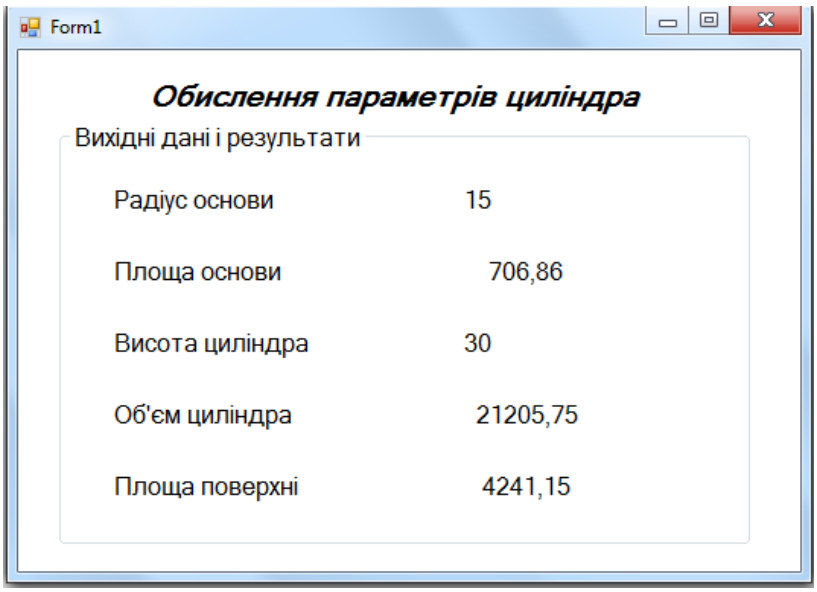

Рисунок 4.2 – Виконання програми

4.2 Обробка подій, пов’язаних з формою

Ми забезпечили розв’язання задачі, вставивши алгоритм обчислення результату в метод, який підключився точно в момент завантаження вікна. Але чому він підключився саме в цей момент? В його імені є слово ***Load***, але це не означає, що із-за цього слова все і виконалося. І як система «узнала», що коли відбулася подія ***Load***, цей метод потрібно виконати? Дійсно, слово ***Load*** тут ні до чого. Відкриємо модуль ***Form1.Designer.cs***, знайдемо секцію ***Form1***, а в ній оператор [6]:

```
this.Load += new System.EventHandler(this.Form1_Load);
```

Фраза ***this.Form1_Load()*** визначає ім’я методу, в який ми вставили алгоритм обчислення. Фраза ***this.Load()*** – це посилання на подію ***Load***. Зарезервоване слово ***this*** означає посилання на об’єкт ***«вікно Form1»*** – бо слово ***this***, що використовується всередині класу ***Form1***. А через крапку указано ім’я події, пов’язаної з вікном. Будь-яка подія містить список, в якому зберігаються відомості про методи (будемо умовно вважати, що зберігаються імена методів).

Спеціалізована операція += забезпечує запис імені методу в список подій. Таким чином, саме цей оператор записав наш метод в список подій. Інакше кажучи, це і є оператор підключення нашого методу до події. А наш метод з повним правом можна назвати обробником подій. Він буде працювати кожного разу, коли наступає подія ***Load***.

І останнє питання: хто вставив даний оператор в текст секції ***Form1***? Не програміст. В той момент, коли ми двічі клікнули по об'єкту ***Form1***, середовище програмування зробило не одну, а дві вставки [6] :

1) в модуль ***Form1.cs*** вставила заготовку обробника;

2) в модуль ***Form1.Designer.cs*** в секцію ***Form1*** було вставлено оператор підключення до події.

Ім'я методу також було автоматично сформоване середовищем програмування по імені події. Постає питання: чи завжди працює така автоматика? Чи завжди середовище формує і заготовку обробника, і оператор підключення? Ні, не завжди. Для події ***Load*** – так, автоматично формує і те, і інше. Цю подію можна вважати головною подією для вікна. Але є і інші події. Крім події ***Load***, з об'єктом ***Form1*** пов'язано ще кілька подій. Наприклад:

- ***FormClosing*** – наступає при закритті вікна;
- ***FormClosed*** – наступає після закриття вікна;
- ***Click*** – наступає, якщо виконується клік по вікну, тобто яке вікно обирається для роботи.

Обробку цих подій доводиться робити вручну: і заготовку обробника писати, і оператор підключення. Продемонструємо цю подію ***FormClosing***.

Припустимо, що ми хочемо при закритті вікна (натискання користувачем кнопки з хрестиком) видати на екран результати обчислення в такому вигляді:

```
Об'єм циліндра = … (значення)
Поверхня = … (значення)
```

Потрібно написати обробник і підключити його до події. Краще за все це робити в зворотному порядку: спочатку виконати підключення, а потім описувати метод-обробник – в цьому випадку можна максимально використовувати підказку середовища програмування. Відкриємо модуль ***Form1.Designer.cs***, знайдемо секцію ***Form1***. В кінці цієї секції почнемо набирати: слово ***this***, потім знак ***«.»***. В списку, що випаде, оберемо подію ***FormClosing*** (подія в списку позначена значком ***«блискавка»***).

Тепер наберемо операцію += і побачимо підказку середовища. Середовище підказує, що воно рекомендує натиснути клавішу «***Tab»***. Натиснемо, і запропонований текст буде включено в програму. Це і буде оператор підключення [11]:

```
this.FormClosing+=new
System.Windows.Forms.FormClosingEventHandler
(Form1_FormClosing);
```

Але це ще не все. З'являється друга підказка – і знову натискання клавіші ***«Tab»***. Знову натискаємо, і в текст модуля вставиться заготовка обробника. Всередині обробника уже буде один оператор, але он нам не потрібний: видалимо його. А потім перенесемо обробник в модуль ***Form1.cs***.

Після цього запустимо програму і упевнимося, що вона працює правильно. Але при закритті вікна нічого не відбувається – метод обробник є, він підключений, але в ньому немає жодного оператора. Запишемо всередину методу потрібний алгоритм. Нам потрібно видати результат на екран. Для цього скористаємося методом ***Show()*** з класу ***MessageBox*** [7]:

```
MessageBox.Show(string.Format( "Об'єм циліндра = {0}
Площа поверхні = {1}",label10.Text, label11.Text));
```

При закритті вікна на екрані з'явиться вікно повідомлення. Нас не влаштовує видача тексту в один рядок. Вставимо в шаблон форматування сполучення символів ***«\n»*** перед словом ***«Площа поверхні»***. Це сполучення забезпечує «переведення рядка» – наступний текст друкується з нового рядка. Додайте і перевірте, запустивши програму (рис. 4.3).

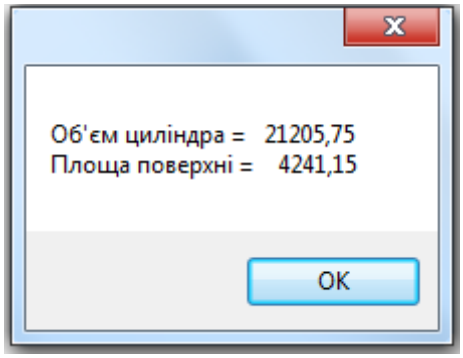

Рисунок 4.3 – Результати виконання

Завдання для самостійної роботи

1. Реалізуйте алгоритм: після закриття вікна на екран видається повідомлення ***«Кінець роботи програми»****.*

2. Реалізуйте алгоритм: при виборі вікна програми на екран видається текст ***«Ви знову повернулися в свою програму»****.*

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Перерахуйте події, що відбувають в процесі виконання програми.
2. Призначення методу ***Form1_Load()***. *Я*к створити для нього заготовку?
3. *Я*ким чином описуються події в модулі ***Form1.Designer.cs***?
4. Опишіть способи та механізм створення обробників подій.
5. Опишіть призначення та застосування обробника події ***FormClosing***? Коли вона наступає?
6. Опишіть призначення та застосування обробника події ***FormClosed***? Коли вона наступає?
7. Опишіть призначення та застосування обробника події ***Click***? Коли вона наступає?
8. Опишіть призначення та спосіб використання методу ***Show()*** з класу ***MessageBox.***

ТЕМА 5 ОБРОБКА ПОМИЛОК ВВЕДЕННЯ ЗА ДОПОМОГОЮ ВИКЛЮЧЕНЬ. МЕХАНІЗМИ ВВЕДЕННЯ І КОНТРОЛЮ ДАНИХ

Ми розробили програму для розв'язання задачі. Обчислення характеристик циліндра користувач повинен здійснювати в наступному порядку:

- використовуючи середовище програмування, візуально задати значення радіуса і висоти через властивість ***Text*** об'єктів ***label7*** і ***label9;***
- запустити програму і отримати результат.

Такий порядок роботи має суттєві недоліки:

- користувач повинен уміти працювати в візуальному середовищі, щоб змінювати вихідні дані, а він не повинен це уміти – він же не програміст;
- після зміни даних за допомогою середовища програмування програму кожний раз потрібно запускати заново.

Очевидно, було б краще, якщо б кожний займався своїми справами. Програміст нехай розробляє програму, а користувач за її допомогою розв'язує свою задачу. Програміст повинен розробити програму так, щоб користувач, запустивши її один раз, міг розв'язувати свою задачу багато разів: сам задавати вихідні дані і отримувати результати. Необхідно переробити програму так, щоб користувач міг запустить її один раз, а вихідні дані зміг вводити уже під час її виконання. І не один раз, а стільки, скільки потрібно [4].

5.1 Розв'язання проблеми введення даних

Візьмемо за основу розв'язання задачі з теми 3. Там є заготовлене вікно, але алгоритми ще не вставлені. Скопіюємо його в доступну робочу область і перейменуємо скопійоване рішення в ***Задача1_Т5*** . Відкриємо проект. Перед нами поставлена задача: надати можливість користувачу набирати вихідні дані уже після запуску програми. Але в нашому проекті вихідні дані – це об'єкти ***Label***. Їх значення неможна змінювати вручну під час роботи програми. Значить, необхідно підібрати для збереження вихідних даних інший об'єкт.

Відкрийте панель інструментів. Знайдіть там компонент ***textBox***. Цей компонент створює об’єкт, який можна використовувати для введення значень під час виконання програми. Видаліть з вікна ***Form1*** об’єкти ***label7*** і ***label9***. Замість кожного з них установіть компонент ***TextBox*** – в вікні з’являться об’єкти ***textBox1*** і ***textBox2***. Зверніть увагу: ці об’єкти відображаються не як надписи, а як поля введення. Імена об’єктів: ***textBox1*** і ***textBox2***. Запустіть програму і спробуйте в цих полях набрати які-небудь числові значення (наприклад, радіус 10 і висота 28). Введення виконується, але ніякі розрахунки не виконуються (рис. 5.1).

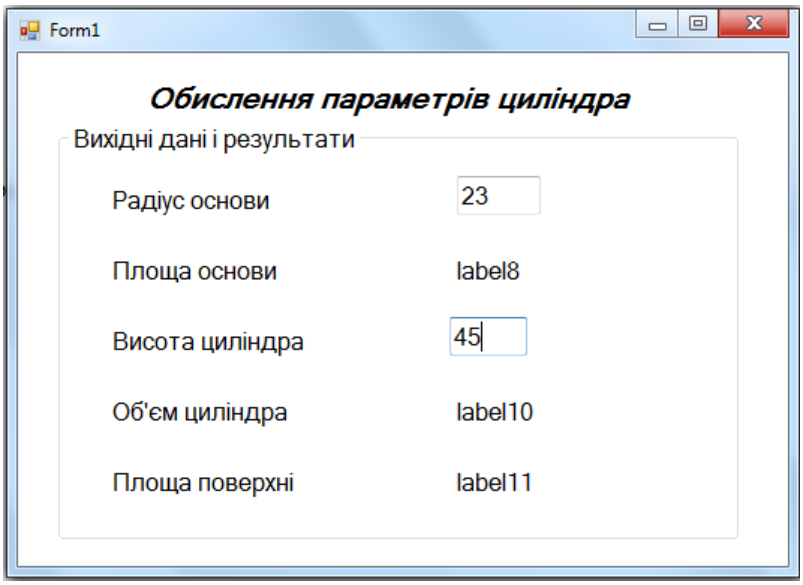

Рисунок 5.1 – Розрахунки відсутні

Перед нами знову проблема: куди вставити алгоритм обчислення результатів? Минулого разу ми вставили алгоритм в метод-обробник події ***Load***. Цим самим ми забезпечили обчислення безпосередньо перед відображенням вікна. Але зараз це не проходить, тому що при завантаженні вікна ще немає вихідних даних – вони з’являться після відображення вікна, коли користувач набере їх в полях введення. Це означає, обчислення повинні виконуватися зразу після того, як користувач закінчить набір значення. Таким чином, подією, яка повинна запускати алгоритм обчислення, являється завершення введення значення. [6]

В процесі введення значення може вводитися багатозначне число. Більш того, в процесі введення можуть виправлятися помилки введення – користувач

може помилково ввести не ту цифру. Значить, потрібно чітко визначити момент, коли саме введення завершено.

З об'єктом ***textBox*** пов'язано ряд подій. Одна з них називається ***«втрата фокусу введення»***. Сутність події така. Доки дані набираються користувачем, на об'єкті встановлено фокус введення, тобто об'єкт в стані введення даних. Але як тільки користувач за допомогою мишки (або клавіші ***«Tab»***) переведе фокус введення на інший об'єкт, то зразу наступає подія ***Leave*** – втрата фокусу введення. Це і є завершення введення значення.

Спробуємо скористатися цією подією. В модулі ***Form1.Designer.cs*** в секції ***textBox1*** набираємо оператор підключення обробника [7] :

```
this.textBox1.Leave+=new
System.EventHandler(textBox1_Leave);
```

Зверніть увагу: після ***this*** ми вказали ще й об'єкт, який втрачає фокус. Якщо б ми цього не зробили, то подією була б втрата фокусу вікном, а не полем введення. Створену заготовку обробника переносимо в модуль ***Form1.cs***. В цей обробник нам і потрібно записати алгоритм обчислення. Але не весь, а тільки обчислення площі основи – адже ми оброблюємо втрату фокусу об'єктом ***textBox1***, тобто завершення введення радіусу. Алгоритм такий:

```
double r;
r = Convert.ToDouble(textBox1.Text);
label8.Text =
string.Format("{0,10:####.##}", Math.PI * r * r);
```

Перевірте правильність роботи програми (рис. 5.2).

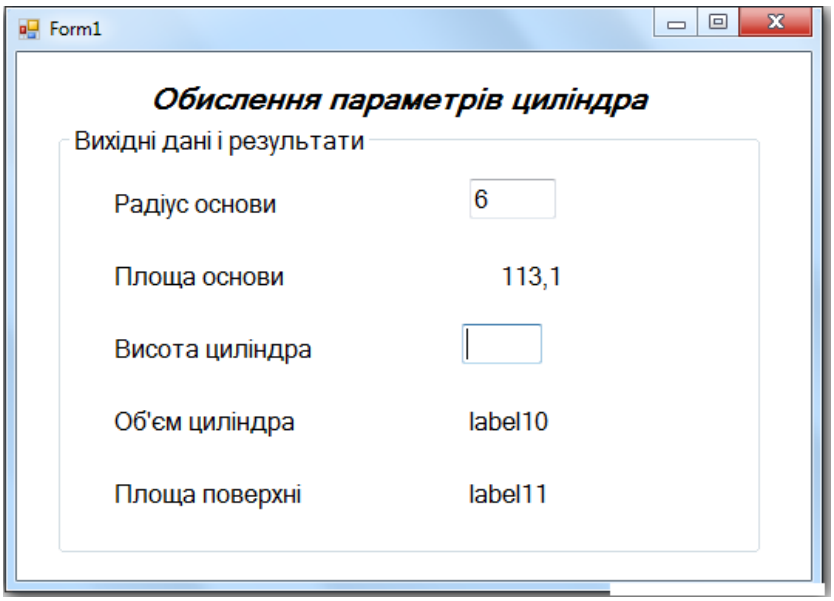

Рисунок 5.2 – Введення радіусу

Завдання для самостійного виконання

Забезпечте обчислення об'єму і поверхні циліндра, використовуючи аналогічну подію для об'єкта при введенні значення в об'єкт ***textBox2****. Після виконання завдання перевірте правильність обчислень. Візьміть у якості вихідних даних для радіусу 6 і висоти значення 88 (рис. 5.3).*

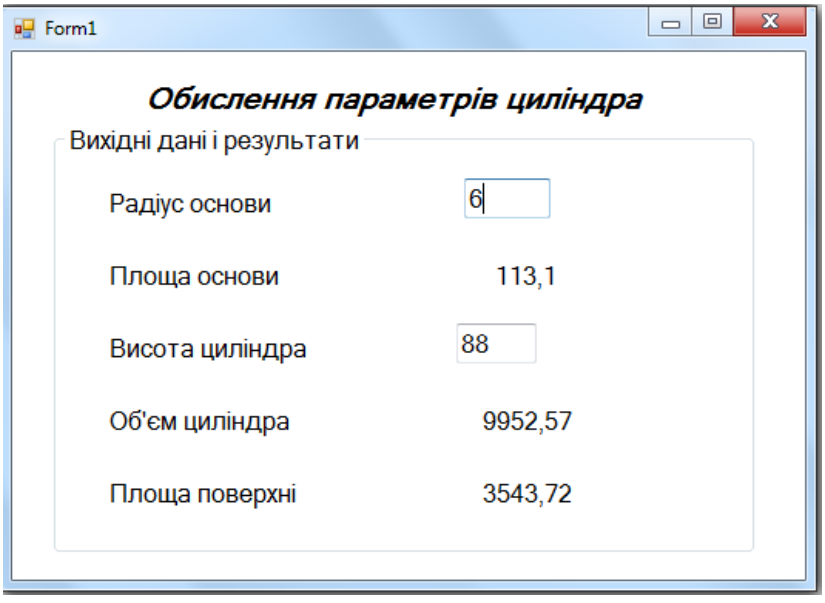

Рисунок 5.3 – Введення висоти

5.2 Доопрацювання і виправлення помилок введення

А тепер змініть значення радіуса на 100 (результат на рис. 5.4).

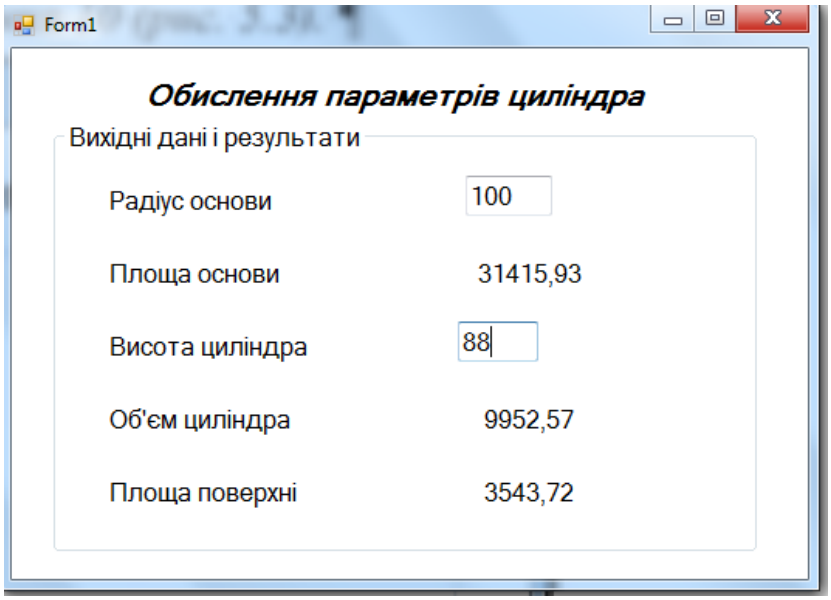

Рисунок 5.4 – Змінений радіус

Але ж це зовсім не вірно! Якщо змінився радіус, то зміниться все: і площа основи, і об’єм, і поверхня. У нас же змінилася тільки площа основи. Бо при задані радіуса в методі-обробнику обчислюється тільки площа основи. Додамо в метод-обробник об’єкта ***textBox1*** оператори обчислення об’єму і поверхні. Для цього видалимо з першого обробника два перших оператори і додамо в нього все, що є в другому обробнику. Отримаємо:

```
double h, r;
h = Convert.ToDouble(textBox2.Text);
r = Convert.ToDouble(textBox1.Text);
label8.Text =
string.Format("{0,10:####.##}", Math.PI * r * r);
label10.Text =
string.Format("{0,10:####.##}", Math.PI * r * r * h);
label11.Text = string.Format
("{0,10:####.##}", 2*Math.PI*r*h + 2*Math.PI*r*r);
```

Запустимо програму. Наберемо значення 10 і перенесемо фокус введення. Отримаємо вікно аварійного завершення програми (рис. 5.5).

Натиснемо кнопку Сведения і за допомогою движків знайдемо адресу помилки. Це 21 рядок в модулі ***Form1.cs***.

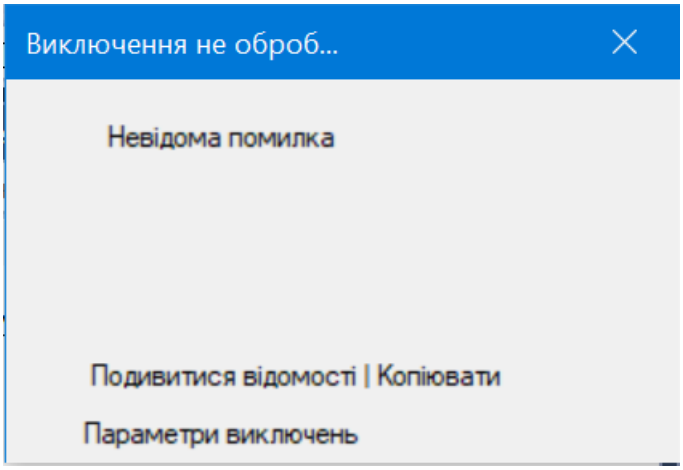

Рисунок 5.5 – Вікно аварійного завершення

Відкриємо модуль, знайдемо рядок:

```
h = Convert.ToDouble(textBox2.Text);
```

В цьому рядку виконується перетворення значення висоти з рядкового значення в чисельне. Але ж ми ще не ввели значення висоти – воно не визначено, і тому виконання оператора завершилось помилкою. Як виправити цю помилку? Спробуємо розібратися, як ця помилка виникла.

При виконанні програми система з'ясувала, що метод ***ToDouble()*** не може перетворити значення (воно не існує) і, як наслідок, програма не може виконуватися далі. В тому випадку, коли виконання програми неможливе, система аварійно завершує виконання програми. Кажуть, що система виробила виключення. Ситуація зовсім не безнадійна. Справа в тому, що програміст може на випадок виникнення виключення передбачити деякий алгоритм. Є механізм, що дозволяє включити в програму алгоритм обробки виключення, який буде виконано при аварійному завершенні. Оформимо оператор перетворення висоти як окремий блок, а перед першою дужкою запишемо слово ***try***. Отримаємо [4]:

```
try
{ h = Convert.ToDouble(textBox2.Text); }
```

Це означає, що даний оператор поставлено під контроль. Зразу після цього створимо ще один блок – блок пастки помилок:

```
catch
{ }
```

Сенс двох блоків такий. Якщо при виконанні будь-якого оператора всередині блока ***try*** буде вироблене виключення, то виконання програми не перерветься – програма продовжить своє виконання операторами блока ***catch***. Залишилось вирішити: що потрібно зробити, якщо буде вироблено виключення? Очевидно, що якесь значення висоти повинно бути сформовано. Наприклад, нульове. Всередині блока ***catch*** запишемо оператор: ***h = 0;*** Перевіримо роботу програми. Після набору радіуса маємо (рис. 5.6). Як видно, об’єм не обчислено, а повна поверхня дорівнює площі двох основ. Якщо набрати значення висоти, то все інше теж буде обчислено. Тепер неначе все добре. Але давайте зробимо помилку в наборі радіуса: наберемо замість цифри букву. Наприклад, так: ***10a***. І ми отримаємо ту ж саму проблему, але уже з радіусом. Нецифрове значення також не може бути перетворене в число. А а значить, також виникає виключення. [8]

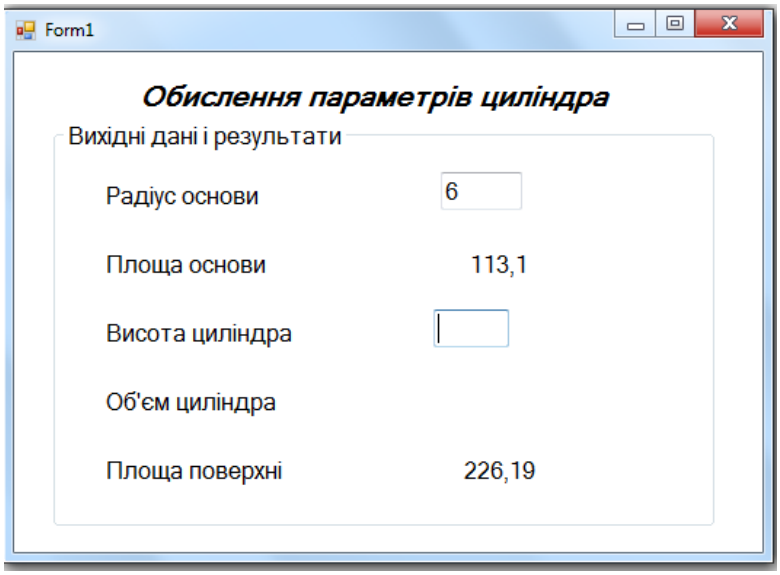

Рисунок 5.6 – Виключення оброблено

Завдання для самостійного виконання

Забезпечте обробку виключень для радіуса. Те ж саме для радіуса і висоти необхідно зробити в другому методі-обробнику.

Зауваження. Після всіх доробок обидва обробники дуже схожі. Відмінність лише в тому, що в другому немає оператора обчислення площі основи. Але якщо б він і був, то все одно все працювало б правильно. Можливо варто зекономити на одному обробнику?

Завдання для самостійної роботи

1. Зробіть так, щоб обидві події (втрата фокуса будь-яким з об'єктів ***textBox****) оброблював один обробник. Наприклад,* ***textBox1_Leave****. Другий обробник видаліть з програми.*

2. Перероблять програму. Використайте замість події ***Leave*** *подію* ***TextChanged****.*

Зауваження. Подія наступає при наборі кожного символу. Це дозволить бачити розрахунки уже в процесі набору без втрати фокуса.

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Призначення події ***Leave*** та використання її обробника.
2. Призначення події ***TextChanged*** та використання її обробника.
3. Опишіть призначення методу ***Format()***, його аргументів.
4. Опишіть механізм алгоритм обробки виключеннь.
5. Обробка помилок введення за допомогою виключень.
6. Опишіть структуру та використання оператору ***try {.} catch { }.***

ТЕМА 6 ТИПОВІ АЛГОРИТМИ ОБРОБКИ МАСИВУ

Створимо новий проект з іменем ***Задача2_Т6***. В ньому єдине вікно наступного вигляду (рис. 6.1). У вікні – 16 об'єктів ***label*** і один об'єкт ***textBox1*** з установленим багаторядковим режимом (властивість ***MultiLine=true***). В об'єктах ***label*** з 10 по 16 будуть відображатися значення у відповідності з написами в об'єктах. В об'єкт ***textBox1*** користувач буде набирати значення елементів масиву. Кнопка забезпечує запуск алгоритму обчислень. Порядок роботи користувача такий. Спочатку він вводить значення в масив, потім натискає кнопку і отримує результати одразу всіх обчислень.

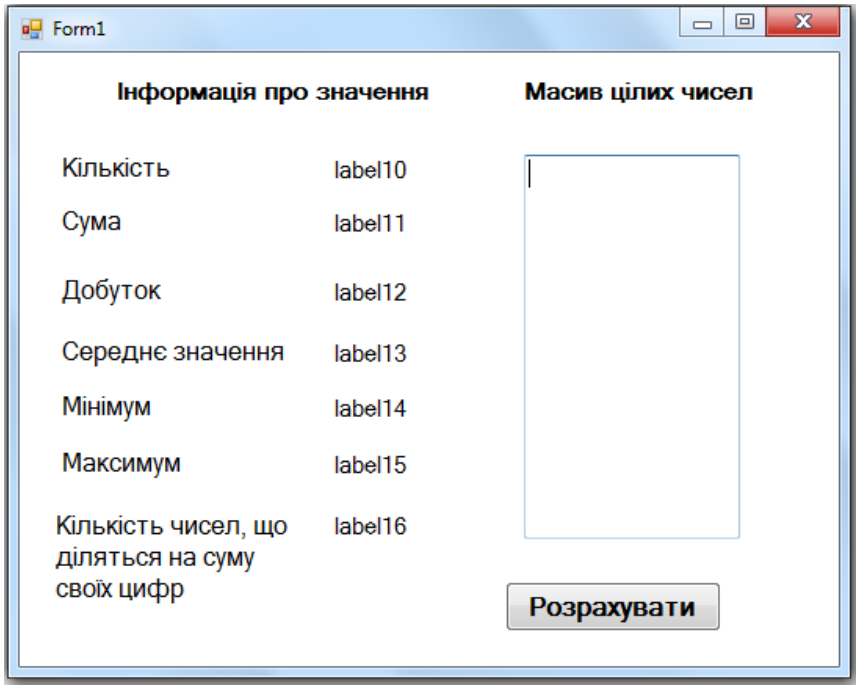

Рисунок 6.1 – Вікно задачі

При натисканні на кнопку виникає подія ***Click***, обробник якої будується подвійним кліком по кнопці (заготовка добавиться в модуль ***Form1.cs***) [7] :

```
private void button1_Click(object sender, EventArgs e)
{ }
```

Підключати цей обробник до події немає необхідності – це зробить сама система. Відкрийте модуль ***Form1.Designer.cs*** і переконайтесь в цьому. Всі характеристики масиву (сума, середнє та інші) обчислюються за власними типовими алгоритмами роботи з масивами. Виключення – кількість чисел, які

діляться на суму своїх цифр. Всі алгоритми повинні бути записані всередині обробника.

До натискання кнопки дані зберігаються в об'єкті ***textBox1***. Для цього об'єкт має колекцію рядків ***Lines*** (звичайний масив типу ***string***).[7] Перетворимо елементи цього масиву з рядкових значень в числові і збережемо їх у цілочисельному масиві, вважаючи, що користувач задає в одному рядку тільки одне число:

```
int[] m = new int[1000];
```

Тисячі елементів достатньо. Потім обчислимо кількість чисел в колекції:

```
int n = textBox1.Lines.Length;
```

Оголосимо змінну циклу – індекс поточного значення:

```
int i;
```

Оголосимо змінну, в якій буде зберігатися кількість значень в масиві:

```
int k = 0;
```

Напишемо цикл, який формує масив **m** шляхом перетворення рядкових значень з колекції:

```
for (i = 0; i < n; i++)
{ m[k] = Convert.ToInt32(textBox1.Lines[i]); k++; }
```

При додаванні значення збільшуємо лічильник чисел на одиницю. Перевіримо роботу програми. Набираємо кілька чисел. При натисканні на кнопку нічого не відбувається, якщо числа набрано вірно – цифровими символами. Але, якщо десь випадково замість цифри попала літера або один з рядків колекції виявився пустим, то програма завершується аварійно. Це означає, що генерується виключення під час виконання методу ***ToInt32()***.

Завдання для самостійної роботи

1. Обробіть виключення. Якщо в рядку некоректне значення (не цифра або пустий рядок), то ніяке значення в масив не включається.

2. Запишіть алгоритми видачі на екран кількості чисел в масиві, суми, добутку елементів масиву, середнього арифметичного елементів масиву, мінімального та максимального елемента масиву, підрахунку кількості чисел, які діляться націло на суму своїх цифр.[5]

*3. Доробіть програму. Виведіть на екран номери рядків, які мають помилки введення і не включені в обчислення (використайте вікно **MessageBox,** збережіть номери рядків в окремому масиві).*

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Як встановити багаторядковий режим для об'єкту ***TextBox***?
2. Дайте характеристику та споаіб використання властивості колекція рядків ***Lines*** для об'єкту ***TextBox.***
3. Опишіть процес створення обробника події ***Click***?
4. Яким чином здійснюється оголошення одновимірного масиву?
5. Опишіть механізм введення елементів одновимірного масиву.
6. Сформулюйте алгоритм видачі на екран кількості чисел в масиві.
7. Сформулюйте алгоритм видачі на екран суми, добутку елементів масиву.
8. Сформулюйте алгоритми видачі на екран середнього арифметичного елементів масиву.
9. Сформулюйте алгоритм видачі на екран мінімального та максимального елемента масиву.
10. Сформулюйте алгоритм підрахунку кількості чисел, які діляться націло на суму своїх цифр.

ТЕМА 7 РОБОТА С МАСИВАМИ ВИПАДКОВИХ ЧИСЕЛ. МЕТОДИ КЛАСІВ RANDOM І MATH

7.1 Інтерфейс і постановка задачі

Створимо новий проект з іменем ***Задача3_Т7.***

Завдання для самостійного виконання

Створіть вікно наступного вигляду (рис 7.1).

1. Два об'єкта ***ListBox***:
   - ***listBox1*** – для аргументів функції;
   - ***listBox2*** – для значень функції.
2. Три об'єкта ***textBox***:
   - ***textBox1*** – поле для джерела повторення;
   - ***textBox2*** – поле для мінімуму;
   - ***textBox3*** – поле для максимуму.
3. Об'єкт ***comboBox1*** – для вибору функції.
4. Два об'єкта ***checkBox***:
   - ***checkBox1*** – для генерації серії чисел, що повторюється;
   - ***checkBox2*** – для генерації цілих чисел.
5. Три об'єкта ***radioButton***:
   - ***radioButton1*** – всі цілі;
   - ***radioButton2*** – цілі в діапазоні [0, max];
   - ***radioButton3*** – цілі в діапазоні [min, max].
6. Об'єкт ***groupBox1*** – групує характеристики генератора випадкових чисел.
7. Кнопка ***button1*** – кнопка с написом ***«Генерація»***.
8. Об'єкт ***label3*** – це надпис ***«Функції»***.
9. Об'єкт **label2** – це надпис ***«Джерело повторення»***.

Всі інші надписи – це об'єкти ***label*** під довільними номерами.

Запустіть програму і уважно прочитайте те, що описано нижче.

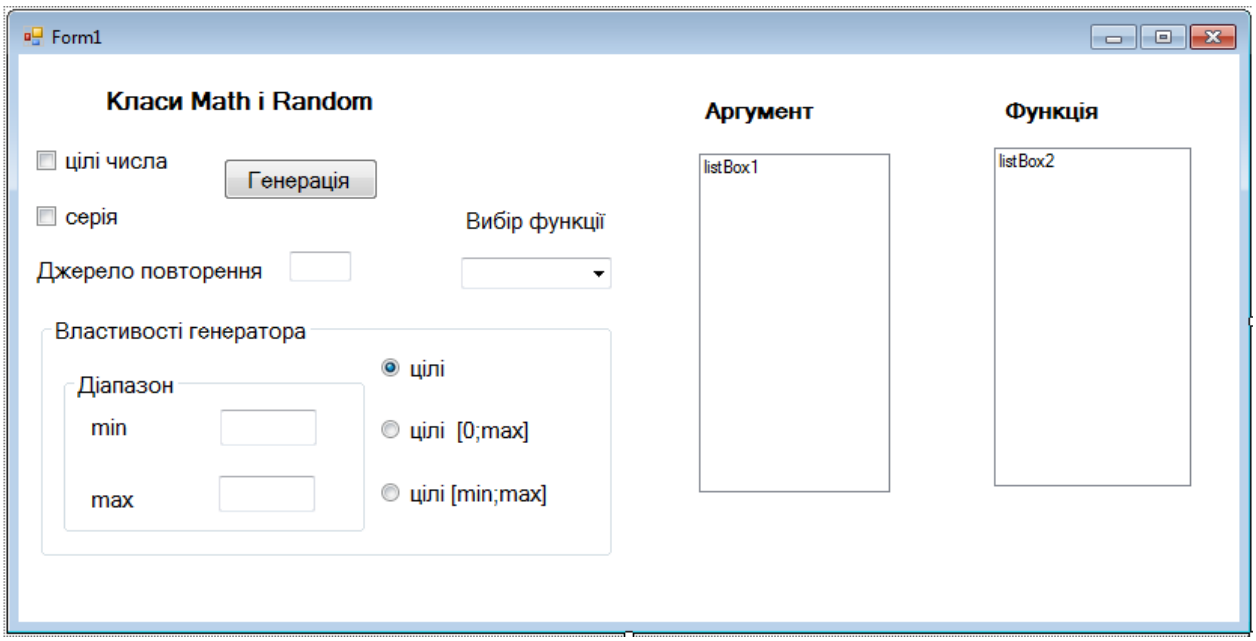

Рисунок 7.1 – Вікно задачі № 3

Пропонується реалізувати наступний сценарій.

Користувачу потрібна програма для вивчення функцій класу ***Math***. Функцію він зможе обрати зі списку об'єктів ***comboBox1***. Переконайтесь, що доки список пустий. При виборі функції надпис ***«Функція»*** повинен замінитися на ім'я обраної функції.

Аргументи для обчислення функції будуть розміщуватися в об'єкті ***listBox1***, а результати обчислення – в об'єкті ***listBox2***. Обчислення повинні виконуватися одразу, як тільки користувач обере функцію.

Аргументи формуються в списку об'єкта ***listBox1*** при натисканні кнопки ***«Генерація»***. Значення, що генеруються – це випадкові числа.

Які значення будуть генеруватися, залежать від налаштувань. Це можуть бути дійсні числа в інтервалі від 0 до 1 (за замовчуванням). Можна установити прапорець – тоді будуть генеруватися цілі числа.

Серії чисел, що генеруються, можуть бути різними, а при установленні прапорця повторення серії – кожного разу однаковими для кожного заданого числа (джерела повторення).

В випадку цілих чисел можлива генерація:

- з усіх можливих додатних цілих чисел;
- з відрізку від 0 до максимального значення;
- з відрізку від мінімального до максимального (в залежності від обраного перемикача).

Значення мінімуму і максимуму повинні задаватися тільки за необхідності. Бажано, щоб в кожний момент часу виконання програми в вікні відображалося тільки те, що необхідно.

Наприклад, якщо не установлено прапорець цілих чисел, то і всі настройки для цілих відображуватися не повинні. Те же саме і з повторенням серії: якщо прапорець не установлено, то і джерело відображатися не повинен. Закрийте запущену програму.

Переходимо до реалізації сценарію.

7.2 Формування вихідних даних

За умовами задачі при запуску програми прапорець «цілі числа» не установлено. Значить, не повинні відображатися ніякі з настройок, тобто всі об'єкти, вміщені в ***groupBox1***, повинні бути невидимі. Знайдемо в властивостях об'єкта ***groupBox1*** властивість ***Visible*** і установимо в ньому значення ***false***. Запустіть програму і переконайтесь, що об'єкти настройки не відображаються.

Завдання для самостійного виконання

При запуску програми прапорець повторення серії також не установлено. Зробіть так, щоб під час запуску програми поле джерела повторень і надпис не відображалися на екрані. Перевірте запуском програми.

При установці прапорця «цілі числа» установлюється режим генерації цілих чисел. При цьому повинні показатися всі режими настройки, вміщені в об'єкті ***groupBox1***. І навпаки, коли прапорець знято, режими настройки знову повинні стать невидимими. Зробимо це. Подвійним кліком по об'єкту ***checkBox2*** відкриємо обробник події ***Click***:

```
private void checkBox2_CheckedChanged
     (object sender, EventArgs e) { }
```

Додамо всередині методу такий алгоритм: якщо прапорець установлено, то зробимо групу об'єктів видимою, а якщо не установлено, зробимо групу невидимою:

```
if (checkBox2.Checked == true)
  { groupBox2.Visible = true; }
     else groupBox2.Visible = false;
```

Підключати обробник до події не потрібно – це зроблено автоматично.[7]

Перевірте, як це працює.

Завдання для самостійного виконання

*При установці прапорця **«повторяющаяся серия»** поле джерела і надпис повинні відображатися. Навпаки, при знятті прапорця обидва об'єкти повинні стати невидимими. Запрограмуйте цю логіку і перевірте роботу.*

Тепер наповнимо список об'єкта ***comboBox1.***

Включимо в цей список імена тих функцій класу ***Math***, які користувач зможе тестувати. Наприклад, такі [10] :

Геометричні: `Atan` (арктангенс), `Cos` (косинус), `Exp` (експонента, число E в ступені), `Log` (натуральний логарифм), `Sin` (синус), `Tan` (тангенс).

Математичні: `Floor` (округлення), `Ceiling` (округлення), `Pow(x,3)` (піднесення числа в третю ступінь), `Sqrt` (квадратний корінь).

Для занесення в список відкриємо властивості об'єкта ***comboBox1*** і знайдемо властивість ***Items***. Ця властивість являється колекцією значень. Занесемо їх вручну. Перевіримо, запустивши програму. [10]

За умовами задачі при виборі імені функції це вибране ім'я повинне замінити собою надпис ***«Функція»***.

Завдання для самостійного виконання

*Замініть надпис **«Функція»** при виборі імені зі списку.*

Вказівки

1. Обране ім'я функції – у властивості ***Text*** об'єкта ***comboBox***. Подія, яка наступає при виборі імені – ***SelectedIndexChanged***.

2. Створіть відповідний метод-обробник – для створення заготовки достатньо подвійного кліку на об'єкті ***comboBox1***. Підключати до події його не потрібно. Тепер можна зайнятися кнопкою. Подвійним кліком по кнопці створимо обробник події. Алгоритм тут достатньо складний. Адже ми повинні

забезпечити різні способи генерації послідовності чисел – їх повинно бути 10 штук.

Нам можуть знадобитися два генератора – два об'єкта класу ***Random***. Один – для випадку послідовності, що не повторюється, інший – для послідовності, що повторюється. Створимо той, який потрібний. А щоб узнати, який саме потрібний, перевіримо прапорець ***checkBox1***. В залежності від того обраний він чи ні, у нас будуть дві гілки алгоритму.

Перша гілка ***if*** відповідає послідовності, що не повторюється. Тут повинен бути створений простий об'єкт класу ***Random***:

```
Random G = new Random();
```

Гілка ***else*** відповідає послідовності, що повторюється. Тут повинен бути створеним складний об'єкт, що враховує задане значення в джерелі повторення ***textBox3.Text***. Але, це значення типу ***string***, його потрібно перетворити в ціле число, а це можливо тільки тоді, коли це цифровий вираз! Таким чином, необхідно ще обробити виключення: якщо значення не задано або задано не цифрою, то взяти в якості джерела будь-яке число. Наприклад, нуль:

```
Random G;
if (checkBox1.Checked == false) G = new Random();
     else
     { int n;
       try { n = Convert.ToInt32(textBox1.Text); }
         catch { n = 0; }
       G = new Random(n);
      }
```

Об'єкт-генератор створено.

Тепер можна доставати з генератора випадкові числа, використовуючи метод ***Next()*** або метод ***NextDouble()***. Яким методом – залежить від стану прапорця ***«цілі числа»***. Якщо він не обраний, то використовуємо ***NextDouble()***, а якщо обрано, то ***Next()***. Якщо послідовність цілочисельна, то потрібно іще врахувати, який інтервал вибору: всі числа, от нуля до максимуму або від мінімуму до максимуму. А це залежить від того, який з перемикачів (***radioButton1***, ***radioButton2***, ***radioButton3***) обрано.

При обранні двох останніх перемикачів повинні бути обрані допустимі (цифрові) значення мінімуму і максимуму! І весь цей вибір – в циклі. Кожне отримане випадкове число потрібно розмістити в ***listBox1*** як окремий рядок!

Створимо цикл і розгалуження в залежності від стану прапорця:

```
for (n = 0; n < 10; n++)
{ if (checkBox2.Checked == false)
    { }
     else
     { }
}
```

В секцію ***«if»*** запишемо два оператори:

```
double d = G.NextDouble(); //1
listBox1.Items.Add(d.ToString()); //2
```

Перший оператор забезпечує отримання випадкового дійсного числа. Другий за допомогою методу ***Add()*** добавляє це число в колекцію об'єкта, з попереднім перетворення його у в рядок.

Завдання для самостійного виконання

1. Запустіть програму. Перевірте роботу без прапорця – генерація виконується. А з прапорцем не виконується – ще не написано алгоритм!

2. Перезапустіть програму. Натисніть на кнопку. Потім натисніть ще раз. Це непорядок! Повинні бути нові значення, але старі теж залишилися. Підправте програму: перед формуванням списку потрібно його очистити.

Вказівка. Потрібно очистити поле ***Items*** до початку циклу заповнення об'єкта ***listBox1***.[11]

3. Перевірте роботу при установленому прапорці повтору серії, переконайтеся, що при різних значеннях джерела повтору отримуються різні послідовності. Запустіть програму, установіть прапорець ***«цілі числа»****. Відобразяться характеристики генератора для випадку цілих чисел. Зверніть увагу: не обрано жоден з перемикачів. Взагалі це неправильно.*

Завдання для самостійного виконання

Будемо вважати, що при виборі прапорця потрібно установити значення «обрано» в перемикач «всі цілі». Це досягається зміною властивості ***Checked*** *даного перемикача. Внесіть потрібний оператор в програму.*

Тепер можна зайнятися гілкою ***else*** обробника натискання на кнопку. Додамо в цю гілку наступні оператори:

```
int m;
if (radioButton1.Checked == true)
{ }
if (radioButton2.Checked == true)
{ }
if (radioButton3.Checked == true)
{ }
```

Змінна ***m*** призначена для випадкового цілого числа. Кожний блок призначено для свого перемикача: якщо він обраний, то буде обробка. З першим все просто. З генератора береться ціле число з усього діапазону цілих додатних чисел:

```
m = G.Next();
```

В другому блоці потрібно спочатку обчислити максимальне значення. Потрібно врахувати, що воно може бути з помилкою або не задане – в цьому випадку можна взяти якесь значення як максимально можливе, наприклад, 100:

```
int max;
try { max = Convert.ToInt32(textBox3.Text); }
   catch { max = 100; }
m = G.Next(max);
```

Завдання для самостійного виконання

1. Доробіть третій блок. Тут метод ***Next()*** *використовується з двома параметрами – мінімальне і максимальне значення. Не забудьте їх також перевірити на цифрове значення.*

2. Запишіть зразу після третього блока оператор, що добавляє отримане ціле число в список ***listBox1.***

Зауваження. Якщо після додавання компілятор видасть помилку ***«Використання локальної змінної «m», якій не привласнено значення»***, то просто замініть оператор ***«int m;»*** на оператор ***«int m = 0;»***.

7.3 Обчислення значень функції

Необхідно розробити алгоритм обчислення значень функції. Ми обрали 10 функцій, які є в списку об'єкта ***comboBox1***. При виборі функції у нас уже змінюється надпис, тобто подія уже оброблюється. Залишилось в цей же обробник додати блок обчислень. Фактично це 10 різних варіантів обчислень, але потрібно ще розпізнати, який саме варіант обрано – яка функція. Для розпізнавання варіанта не будемо аналізувати ім'я обраної функції. Нам відомо, в якому порядку вони розміщені в списку. Наприклад, функція ***Atan()*** знаходиться в списку під індексом 0, а функція ***Sqrt()*** – під індексом 9. Проще всього скористатися оператором ***switch***, який буде знаходиться всередині циклу, що обчислює функцію для 10 значень з ***listBox1***. Значення відправляються в ***listBox2***.

Додайте в обробник наступний код:

```
int i;
double d=0;
double r=0;
for (i = 0; i < 10; i++)
{ d = Convert.ToDouble(listBox1.Items[i]);
  switch (comboBox1.SelectedIndex)
        { case 0: r= Math.Atan(d); break;
        }
 listBox2.Items.Add(r.ToString());
}
```

Завдання для самостійного виконання

*1. Перевірте роботу програми для функції **Atan()**.*

2. При повторному виборі функції старі значення не зникають. Внесіть виправлення.

3. Після генерації нових аргументів у вікні результатів залишаються старі значення. Забезпечте очистку вікна результатів при генерації нових аргументів.

4. Зараз дійсні аргументи і результати зберігаються з великою кількістю знаків після коми. Відформатуйте виведення значень: аргументи – для дійсних два знака після коми, результати – чотири.

5. Доробіть програму. Забезпечте обчислення інших функцій.

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Опишіть призначення та спосіб використання об’єкт ***ComboBox***.
2. Опишіть механізм та застосування події ***SelectedIndexChanged***.
3. Опишіть призначення та спосіб використання об’єкту ***CheckBox.***
4. Опишіть призначення та спосіб використання об’єкту ***RadioButton.***
5. Опишіть призначення та спосіб використання об’єкту ***GroupBox***.
6. Опишіть призначення класу ***Math.*** Наведіть приклади методів цього класу.
7. Які види об’єктів класу ***Random*** ви знаєте? З якої метою та яким чином вони застосовуються?
8. Опишіть призначення на варіанти виклику методів ***Next()*** та ***NextDouble()*** класу ***Random***.
9. Опишіть способи додавання елементів в колекцію ***Items*** об’єкту ***ListBox.***
10. Яким чином можна видалити окремий елемент колекції ***Items*** об’єкту ***ListBox*** та повністю очистити її?

ТЕМА 8 ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ПРАПОРЦІВ. РОЗРОБКА ФУНКЦІЙ

8.1 Розробка алгоритму програми

Розробити програму знаходження коренів рівняння наступного вигляду: $Ax^2 + Bx + C = 0$. Коефіцієнти рівняння задаються користувачем і відображаються на екрані. Процес розв'язання запускається натисканням кнопки. Розв'язання відображається на екрані і містить:

- виведення про кількість дійсних коренів або повідомлення про їх відсутність;
- значення коренів (показувати тільки знайдені значення).

Варіантів розв'язання даної задачі може бути багато – в залежності від умінь і фантазії програміста. Але в будь-якому випадку потрібно так організувати інтерфейс користувача, щоб йому було максимально зручно і комфортно працювати.

Спробуємо розібратися з умовами задачі. За формою запису – це квадратне рівняння. Але воно буде квадратним тільки в тому випадку, коли A не дорівнює нулю. В противному випадку воно перетворюється в звичайне лінійне: $Bx + C = 0$.

Алгоритм розв'язання квадратного і лінійного рівняння різні. Так як в умові задачі нічого не сказано про можливі значення коефіцієнтів, то значення $A = 0$ необхідно вважати допустимим. Значить, необхідно передбачити алгоритми розв'язання для двох видів рівняння. Для лінійного рівняння маємо наступний алгоритм:

$$x = -C / B.$$

Для квадратного рівняння – трішки складніше:

1. Спочатку обчислюється дискримінант за формулою:

$$D = B^2 - 4AC.$$

2. Далі можливі три випадки:

- якщо D < 0, то дійсних коренів немає;
- якщо D = 0, то рівняння має два однакових кореня, але прийнято говорити, що корінь один, і обчислюється він за формулою:

$$x = \frac{-B}{2A};$$

– якщо D > 0, то рівняння має два різних кореня, які обчислюються за формулами:

$$x_1 = \frac{-B + \sqrt{D}}{2A},\ x_2 = \frac{-B - \sqrt{D}}{2A}.$$

Тепер спробуємо уявити, як буде виглядати вікно програми. Для цього створимо проект з іменем ***Задача4_Т8***.

Створимо заголовок у вигляді об'єкта ***Label*** з текстом ***«Розв'язання рівняння»***. Відобразимо також саме рівняння в вигляді $Ax^2 + Bx + C = 0$ (рис 8.1). Єдиний об'єкт, який нам тут допоможе – це об'єкт ***pictureBox*** (картинка). Порядок дій приблизно такий. Скопіюємо формулу в буфер обміну, відкриємо ***Paint***, вставимо з буфера обміну, збережемо в файл. Перенесемо з ***ToolBox*** компонент ***pictureBox*** на вікно, а потім, використовуючи властивість ***Image*** об'єкта ***pictureBox1***, візуально додамо файл з картинкою в об'єкт. $Ax^2 + Bx + C = 0$

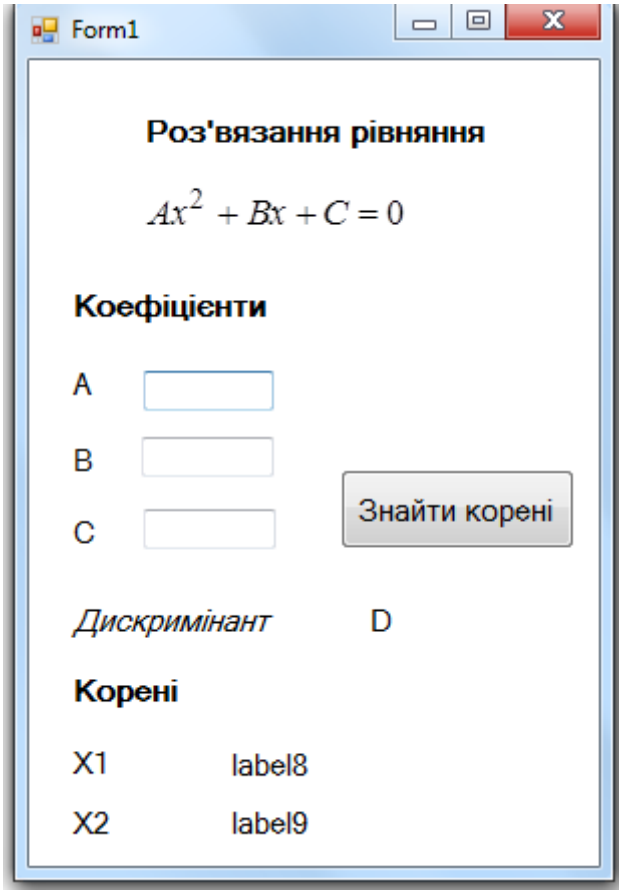

Рисунок 8.1 – Вікно задачі

Для введення коефіцієнтів використаємо компоненти ***TextBox***. Домовимося, що це будуть:

textBox1 – поле введення ***A***; ***textBox2*** – поле ***B***; ***textBox3*** – поле ***C***.

Для виведення розв'язку використовуємо компоненти ***Label***. Також домовимося, що це будуть:

- ***label11*** – значення дискримінанта;
- ***label5*** – повідомлення про кількість коренів або їх відсутність;
- ***label8*** – перший корінь (або єдиний корінь);
- ***label9*** – другий корінь.

Буде в вікні і єдина кнопка с текстом ***«Знайти корені»***. Всі інші пояснюючі надписи оформимо через компонент ***Label***. Приблизний вигляд (з урахуванням наведених вище домовленостей) буде таким (див. рис. 8.1). Запустіть програму. Переконайтесь, що вихідні дані набираються без проблем.

Залишилось «оживити» кнопку. Відкриваємо обробник події «натискання на кнопку». Оголошуємо три дійсних змінних для коефіцієнтів:

```
double a, b, c;
```

І ще три: для дискримінанта і двох коренів:

```
double d, x1, x2;
```

Обчислюємо значення коефіцієнтів за значеннями в об'єктах ***textBox***:

```
a = Convert.ToDouble(textBox1.Text);
b = Convert.ToDouble(textBox2.Text);
c = Convert.ToDouble(textBox3.Text).
```

Визначаємося, яке у нас рівняння – все залежить від значення ***A***:

```
if (a == 0) { }
    else { }
```

В випадку ***if*** у нас лінійне рівняння, інакше – квадратне.

Розв'язуємо лінійне рівняння. Ми можемо скористатися формулою за умови, що знаменник не дорівнює нулю. Якщо же він нуль, то розв'язання немає:

```
if (b==0)
  { label5.Text = "решения нет "; label8.Text=""; }
   else
```

```
{ x1=(-c)/b;
  label8.Text=string.Format("{0,10:##.##}",x1);
  label5.Text = "один корень";
}
```

Перевірте, як це працює. Не забудьте, що ***A = 0.*** І ***B*** також може бути нулем.

Розв'язуємо квадратне рівняння. Обчислюємо дискримінант:

```
d = b * b - 4 * a * c;
```

Розглянемо перший випадок:

```
if (d < 0)
{ label11.Text = "менше нуля";
  label5.Text = "коренів не існує";
}
```

Перевірте, як це працює.

Розглянемо другий випадок:

```
if (d == 0)
{ label11.Text = "0";
  label5.Text = "один корінь ";
  x1 = (-b) / (2 * a);
  label8.Text = string.Format("{0,10:##.##}", x1);
}
```

Перевірте, як це працює.

Завдання для самостійного виконання

Самостійно реалізуйте алгоритм для третього випадку.

Тепер виправимо помилки і недоліки. Очевидно, що все працює нормально, якщо коефіцієнти набрані коректно. А якщо якийсь не набраний або замість цифри – літера?

Завдання для самостійного виконання

А. Забезпечте коректну роботу програми в випадку неправильного набору вихідних даних.

Вказівка 1. Не потрібно перевіряти на коректність кожне значення. Забезпечте контроль всіх зразу, і якщо буде виключення, то виведіть про це

повідомлення на екран «перевірте вихідні дані» з наступним завершенням методу-обробника.

Вказівка 2. Для завершення методу використайте оператор ***return***;

Б. Перевірте роботу програми зразу за кількома варіантами. Переконайтесь, що при виконанні за варіантами 1 і 2, а також при розв'язанні лінійного рівняння на екрані залишаються непотрібні значення і тексти. Приберіть непотрібну інформацію з екрана в кожному варіанті, але не забудьте її поновити для другого варіанта.

8.2 Виділення функцій

Запустіть проект. У вас повинні бути виконані абсолютно всі завдання. Відкрийте програмний код вікна. Проаналізуємо вміст методу-обробника ***button1_Click()***. Це достатньо довгий текст. Але он правильний, забезпечує зручну роботу користувача – і в цьому сенсі (від користувача) ніяких претензій до програміста немає. А ось у програміста до самого себе є. Коли «суцільний» програмний текст достатньо великий, то його стає важко розуміти. Особливо з часом – чим більше проходить часу з моменту його написання, тим менше залишається в пам'яті. Текст не повинен бути дуже великим. Текст можна зменшити, якщо розбити його на окремі підпрограми (маленькі програми), дати цим підпрограмам імена і звертатися до них, коли потрібно, за допомогою всього однієї команди. В C# такі підпрограми, виділені програмістом, називаються функціями (або методами) [11].

Спробуємо перебудувати текст обробника, виділивши з нього окремі фрагменти як функції. Винесемо в окрему функцію наступну групу:

```
try
   { a = Convert.ToDouble(textBox1.Text);
     b = Convert.ToDouble(textBox2.Text);
     c = Convert.ToDouble(textBox3.Text);
   }
  catch
   { MessageBox.Show("проверьте исходные дані"); return;
   }
```

Сенс групи також зрозумілий: перетворення вихідних рядкових даних в числові значення. Назвемо цю функцію ***Перетворення()***. Але, простого

винесення операторів буде недостатньо. Справа в тому, що результат роботи цього блока двоякий. Може бути нормальне завершення, коли всі дані – числові. Але можливий і другий варіант – якщо виробляється виключення. Таким чином, потрібно передбачити обидва варіанти завершення функції. Як?

Наприклад, так. Нехай функція при нормальному завершенні виробляє значення «істина», а при другому варіанті – значення «неістина». В обробнику ми викликаємо цю функцію, а після її завершення перевіримо, що вона виробила (говорять, повернула). Таким чином, функція буде повертати значення типа ***bool***, тобто тип функції буде уже не ***void***, а ***bool***. Запишемо проект нашої функції зразу перед обробником в такому вигляді:

```
private bool Перетворення()
{ try
    { a = Convert.ToDouble(textBox1.Text);
      b = Convert.ToDouble(textBox2.Text);
      c = Convert.ToDouble(textBox3.Text);
    }
   catch
    {MessageBox.Show("проверьте исходные дані ");
return;
    }
}
```

Всі винесені команди видалимо з обробника, а замість них запишемо такий оператор:

```
if (Перетворення() == false) return;
```

Сенс цього оператора такий: якщо в результаті роботи функції повернулось значення ***«неістина»***, то це значить, що в даних є помилка і потрібно завершити роботу обробника оператором ***return*** (повернення). Спробуємо запустити програму. Нажаль, отримаємо кілька помилок (рис. 8.2).

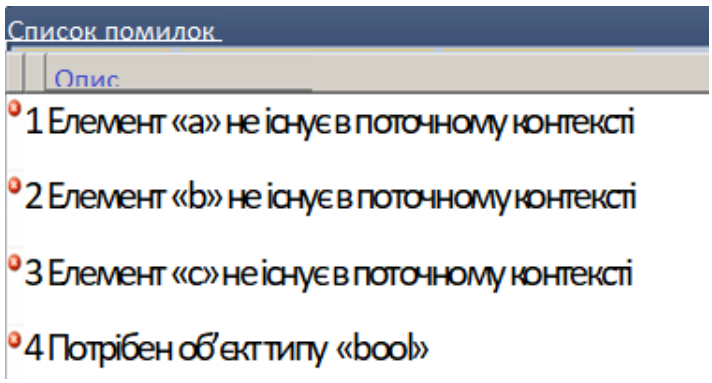

Рисунок 8.2 – Вікно помилок

Три перших помилки мають одну і ту ж природу. Текст пояснення до помилки говорить про те, що змінні не визначені всередині функції ***Перетворення()***.

Насправді, ці змінні оголошені всередині обробника. Тепер ми так побудували роботу, що значення в ці змінні повинні сформуватися всередині іншої функції ***(Перетворення)***, і ці значення повинні бути передані з неї в обробник. Сформовані значення – це вихід функції ***Перетворення()***. В цьому випадку потрібно використання вихідних параметрів. Змінимо заголовок функції і вкажемо список з трьох вихідних параметрів:

```
private bool Перетворення(out double a, out double b, out double c)
```

Але коли змінився заголовок, то потрібно змінити і оператор виклику:

```
if (Перетворення (out a, out b, out c) == false) return;
```

Тепер ми можемо бути впевненими, що якщо значення сформуються в функції Перетворення, то вони будуть передані в змінні ***a***, ***b*** і ***c*** в обробник.

Знову спробуємо запустити програму. І знову помилка, зате яка цікава! Зверніть увагу: під підозру компілятора попала тепер уже вся функція. В повідомлені про помилку говориться, що не всі варіанти виконання функції завершуються поверненням значення. Є два варіанти. Один варіант, коли перетворення не виконано, ми завершимо оператором ***return false***. А другий варіант, коли все нормально? Додамо і туди оператор повернення, але уже повернемо ***«істину»*** [4]:

```
try
 { a = Convert.ToDouble(textBox1.Text);
   b = Convert.ToDouble(textBox2.Text);
   c = Convert.ToDouble(textBox3.Text); return true;
 }
```

Знову виконаємо компіляцію і отримаємо уже три помилки, що мають однакову природу. Вихідні параметри повинні обов'язково отримувати якісь значення. А якщо буде вироблено виключення? Компілятор це «побачив». Давайте привласнимо змінним якісь значення. Наприклад, нульові, в самому початку функції:

```
private bool Перетворення(out double a, out double b, out
double c)
{ a = 0; b = 0; c = 0;
try
```

Виправимо і знову запустимо програму. Тепер все в порядку! Наступний великий фрагмент тексту (при ***a= 0***):

```
if (b==0)
  { label5.Text = "рішення немає"; label8.Text=""; }
  else
    { x1= (-c) / b;
     label8.Text=string.Format("{0,10:##.##}",x1);
     label5.Text = "один корінь ";
     }
label9.Visible = false; label6.Visible = false;
label11.Visible = false;
label10.Visible = false; label7.Visible = false;
```

Проаналізуємо, що тут робиться?

Обчислюється x1. Це означає, це буде вихідний параметр. Він обчислюється на основі ***b*** і ***c***. Це означає, що ці значення повинні бути передані в нову функцію. Таким чином, буде три параметра: два звичайних (значення) і один вихідний (***out***). Повертати функція нічого не буде, тобто тип функції ***void***. Назвемо функцію ***Лінійне()***:

```
private void Лінійне(double b, double c, out double x1)
{ if (b==0)
   { label5.Text = "рішення немає"; label8.Text=""; }
   else
    { x1 = (-c) / b;
      label8.Text=string.Format("{0,10:##.##}",x1);
      label5.Text = "один корінь ";
    }
label9.Visible = false; label6.Visible = false;
label11.Visible = false; label10.Visible = false;
label7.Visible =false ;
}
```

Замість винесеного тексту – один оператор:

```
Лінійне (b, c, out x1);
```

Запускаємо програму. Знову працює! А тепер винесемо в окрему функцію всю логіку розв'язання квадратного рівняння, тобто весь текст з гілки ***else***. Очевидно, що в даному алгоритмі використовуються для обчислення всі коефіцієнти. Це будуть параметри-значення. Далі, формуються результати ***x1*** і ***x2*** – це будуть вихідні параметри. Функція все обчислює і «розкидає» по об'єктах ***label***, тобто їй просто нічого повертати – значить, тип ***void***. Назвемо її – ***Квадратне()***:

```
private void Квадратне(double a, double b, double c, out
double x1, out double x2)
```

Відповідно, виклик функції:

```
Квадратне(a, b, c, out x1, out x2);
```

Запускаємо! Вісім помилок! Перші сім помилок пов'язані зі змінною ***d***. Правильно, вона ж в нові функції не оголошена. Оголошуємо в самому початку функції:

```
double d;
```

А восьма помилка – це попередження. Клікнемо по тексту помилки двічі і побачимо, що компілятор «придрався» до оголошення цієї змінною в обробнику. Насправді змінна оголошена, але ніде не використовується. Видалимо її.

Запускаємо! І знову бачимо: не за всіма гілками алгоритму вихідні змінні отримують значення (а повинні). Надамо їм в самому початку нульові значення (фрагмент):

```
double d;
x1 = 0; x2 = 0;
d = b * b - 4 * a * c;
```

І знову запускаємо програму. Тепер все. Зверніть увагу на метод-обробник: він став зовсім короткий. І зрозумілий. Правда, функцій стало більше, але знову ж кожна – невелика і зрозуміла.

Завдання для самостійної роботи

Виділіть в окремі функції фрагменти функції ***Квадратне()****:*

1) оператор обчислення дискримінанта;

2) фрагмент операторів всередині фігурних дужок після перевірки ***d < 0****;*

3) аналогічно після перевірки ***d == 0****;*

4) аналогічно після перевірки ***d > 0.***

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Опишіть спосіб оголошення функції.
2. Опишить правила передачі аргументів в фукцію.
3. Які типи формальних аргументів ви знаєте? Коли вони використовуються?
4. Опишіть процес створення функції зі змінною кількістю параметрів.
5. Опишіть механізм виклику та передачі аргументів рекурсивної функції.

ТЕМА 9 ВИКОРИСТАННЯ СПИСКІВ. РОЗРОБКА КЛАСІВ

9.1 Постановка задачі

Маємо відомості про автомобілі та їх власників:

- Державний реєстраційний номер.
- Модель.
- Колір.
- ПІБ власника.

Потрібна програма, яка дозволяє вводити указані дані, зберігати введену інформацію і видаляти відомості про автомобіль і його власника.

9.2 Інтерфейс користувача

Список держномерів зберігається в об’єкті ***comboBox1***. Поля введення використовуються відповідно ***textBox1 – textBox4***, кнопки – відповідно ***button1 – button3***.

Завдання для самостійного виконання

Побудуйте інтерфейс користувача за рис. 9.1.

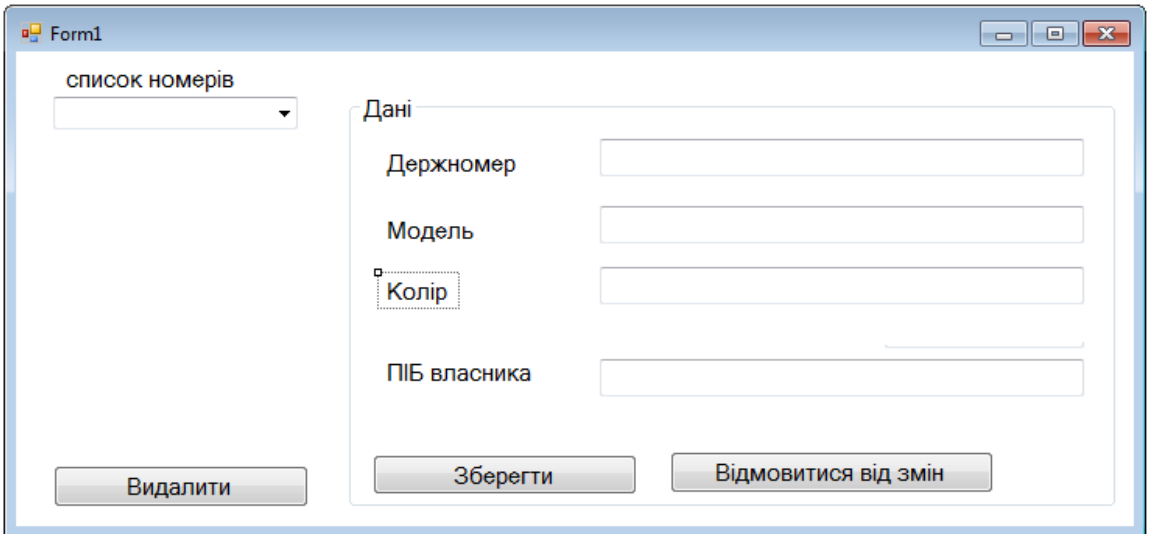

Рисунок 9.1 – Вікно задачі

Сценарій роботи користувача и програми

Користувач запускає програму і бачить на екрані тільки список держномерів і кнопку ***«ВИДАЛИТИ»*** – об'єкти групи даних не показані. При виборі держномера зі списку розкриваються дані. При введені нового держномера також відкриваються поля даних, але вони пусті. Користувач вводить нові значення (або змінює їх) і зберігає зміни (або відмовляється від зберігання зроблених змін). Дані про кожний автомобіль – це окремий об'єкт. Збережені значення записуються в масив об'єктів, а в списку відображаються лише держномери. Держномер можна видалити зі списку – в цьому випадку із масиву видаляються і дані про об'єкт.

9.3 Введення і збереження даних в масиві

Розробка деяких елементів класу даних

Розв'язання будь-якої задачі починається з опису даних, що використовуються. В нашому випадку дані мають складну структуру. Відомості про кожний транспортний засіб містить чотири показника, а самих автомобілів може бути будь-яка кількість. Фактично мова йде про новий тип даних, що містить у собі всі перераховані показники. Оформимо новий тип даних як клас Авто. Помістимо заготовку опису цього класу в модуль ***Form1*** в простір імен ***Задача5_Т9***, але обов'язково після класу `Form1` [9]:

```
public class Авто
{ }
```

В складі класу `Держномер, Модель, Колір` і `ПІБ` визначимо як рядки. Тобто оголошення даних в класі буде таким:

```
public class Авто
{    string Держномер;
     string Модель;
     string Колір;
     string ПІБ; }
```

Після нашої вставки в програмі з'являться помилки, вони пояснюються тим, що ми ще ніяк не використовуємо дані. Продовжуємо розробляти клас.

Тепер необхідно описати конструктор об’єктів. Його призначення: при створенні об’єкта задати вихідні значення його полям. У нас чотири поля. Нехай при створенні об’єкта вони отримують пусті значення. Тіло конструктора буде містити чотири оператора привласнення [9]:

```
string ПІБ;
public Авто()
{ Держномер = ""; Модель = ""; Колір = ""; ПІБ = ""; }
```

Зверніть увагу: як тільки ми додали конструктор, так помилки одразу пропали – з’явились оператори, які задають значення полям об’єкта. Переконайтесь, що програму можна запустити.

Обробка введення держномера

Логічно уявити собі, що користувач при першому запуску програми захоче ввести дані про перший автомобіль. Він спробує набрати в верхньому рядку комбінованого списку держномер. Ця подія – набір з клавіатури. Точніше, натискання символів на клавіатурі, ***KeyPress***. Така подія пов’язана з об’єктом ***ComboBox***, але вона потребує ручного написання і підключення обробника. Створимо обробник, підключимо його до події. Потрібно описати алгоритм, який буде працювати при наборі держномера. У відповідності до сутності події ***KeyPress*** обробник буде виконуватися кожного разу при будь-якому натисканні на будь-яку клавішу. Але держномер буде набрано повністю тільки після натискання клавіші ***«ENTER»***. Це означає, що всі інші натискання метод повинен проігнорувати. Але як визначити, що натиснута саме клавіша ***«ENTER»***? [7]

Відповідь на це питання є очевидною, якщо знати сенс параметрів методу. Параметр `sender` – це джерело події, тобто об’єкт ***comboBox1***. Ми про джерело знаємо, доступ до нього за його ім’ям маємо, а тому `sender` нам не знадобиться. А другий параметр `e` – це контейнер, який містить інформацію про подію. Вираз `e.KeyChar` дозволяє взнати символ натиснутої клавіші. Перевірку можна здійснити так:

```
if (e.KeyChar == (char) Keys.Enter) { }
```

Вставимо цю конструкцію в обробник. Системне перерахування ***Keys*** включає в себе всі символи клавіатури, включаючи ***Enter***. Тільки при порівнянні його потрібно перетворити в символ. Між фігурними дужками

можна записати оператори, які будуть працювати лише при завершенні набору держномера – всі інші клавіші не будуть «помічені» обробником.

Завдання для самостійного виконання

*Переконайтесь, що обробник реагує на натискання клавіші **«ENTER»**. Вставте між фігурними дужками оператор видачі на екран якогось повідомлення (через **messageBox**) і запустіть програму. Наберіть в поле введення що-небудь і натисніть **«ENTER»**. Переконавшись – видаліть оператор видачі повідомлення.*

Тепер визначимо, що робити, коли натиснута клавіша ***«ENTER»***. Очевидно, нам доведеться відобразити невидимі об'єкти вікна – групу даних.

Завдання для самостійного виконання

Додайте оператор, який зробить групу об'єктів видимою.

Введення даних

Тепер можна спробувати набирати дані в поля введення. Взагалі все нормально. Але, ми уже набрали держномер – чому доводиться набирати його знову?

Завдання для самостійного виконання

Додайте ще один оператор. Як тільки група об'єктів стала видимою, перенесіть в перше поле введення значення набраного держномера. Є ще одне значно суттєвіше «але». Колір краще обирати з якогось списку, а не набирати. Користувачу важко помилитися з набором держномера і моделі, а ось з кольором – дуже легко. А це означає, що доведеться заборонити набір, тобто установити у цього поля властивість ***Enable=false***. А для вибору злегка змінимо інтерфейс (рис. 9.2): додамо між рядками кольору і ПІБ об'єкт ***Label*** з текстом ***«Вибір кольору»*** і об'єкт ***comboBox2***, в якому створимо список кольорів. У останнього об'єкт установимо заборону набору кольору – це властивість ***DropDownStyle=DropDownList***.

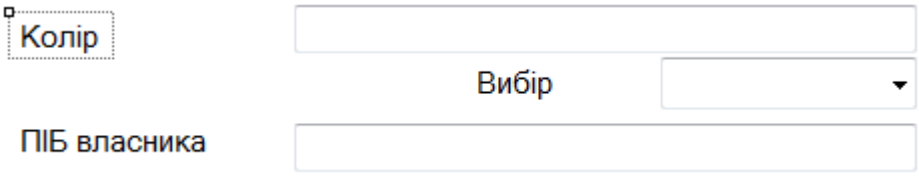

Рисунок 9.2 – Зміна інтерфейсу

Тепер вирішимо питання з заповненням списку кольорів. Створимо в просторі імен Задача5_Т9 ще один тип даних – перерахування ***Color***. Включимо в нього шість значень-констант (можна і більше):

```
public enum Color { невизначений, білий, червоний,
фіолетовий, сірий, зелений };
```

Запишемо цей оператор між класами ***Form1*** і ***Авто***. Потім створимо в ***comboBox2*** список кольорів на основі перерахування ***Color***. Значення цього перерахування перепишемо у колекцію об'єкта ***comboBox2*** при запуску програми – перед відкриттям вікна. Створимо обробник події ***Load*** для вікна ***Form1*** і вставимо в нього код:

```
int i;
for (i = 0; i < 6; i++) comboBox2.Items.Add((Color)i);
```

Запустіть програму і переконайтесь, що список кольорів заповнюється. Вибране значення кольору попадає в властивість ***Text*** об'єкта ***comboBox2***, його потрібно обрати і помістити в текстове поле ***textBox3***. Скористаємося подією ***SelectedIndexChanged***, яка виникає при виборі. Подвійним кліком по об'єкту ***comboBox2*** створимо обробник цієї події [7]. Підключається він автоматично. В середину обробника запишемо оператор:

```
textBox3.Text = comboBox2.Text;
```

Запустіть програму і перевірте роботу.

А тепер зробимо так, щоб при наборі даних користувач міг працювати не лише мишкою. При наборе даних зручно переміщуватися від поля до поля клавішею ***«TAB»***. Відкриємо модуль ***Form1.Designer.cs***. В секціях-описах кожного об'єкта є оператор, що визначає порядок обходу об'єктів вікна за допомогою клавіші ***«TAB»*** [7]. Установимо індекси (***TabIndex***) наступним чином: ***0 – comboBox1, 1 – textBox1, 2 – textBox2, 3 – comboBox2, 4 – textBox4, 5 –*** кнопка ***button2, 6 –*** кнопка ***button3, 7 –*** кнопка ***button1.*** Всі інші не суттєві. Тепер запустимо програму і попрацюємо клавішею ***«TAB»***. Курсор повинен

переміщуватися в указаному порядку. На цьому обробка введення даних завершена. Можна починати обробку натискання кнопок.

Кнопка відміни змін

При відміні змін (або відмові від збереження набраних даних) потрібно зробити:

- очистити поля textBox1 – textBox4;
- прибрати з екрана (приховати) блок введення даних;
- очистити поле введення в об'єкті ***comboBox1***.

Завдання для самостійного виконання

Внесіть необхідні зміни в програму.

Кнопка збереження результатів введення даних

Введені дані повинні зберегтися в елементі масиву даних – це умова задачі. Але ми ще не створили масив! Де саме він повинен розміщуватися и в якому вигляді? В якому з класів: ***Form1, Program, Авто***? Якого типу повинен бути масив? У масиву повинні зберігатися всі дані про кожний автомобіль. А це означає, що елементи масиву повинні бути типу ***Авто***. Припустимо, що ми створюємо масив для зберігання інформації про 1000 автомобілів. Можна для оголошення використати наступний оператор:

```
Авто [] МД = new Авто[1000];
```

Запишемо цей оператор в самому початку класу ***Form1***. Це гарантує доступ до нього з будь-якого методу цього класу, тобто в межах вікна. Крім того, також на початку класу створимо зміну-лічильник, яка буде містити кількість заповнених елементів масиву:

```
int КількістьЕлем;
```

Цій змінній потрібно дати нульове значення при відкритті вікна. Тепер згадаємо, що при оголошені масиву в пам'яті виділено місце тільки під його елементи, а вони являються посиланнями на об'єкти – при створені масиву ці посилання будуть дорівнювати значенню ***null***. Виділимо місце під кожний об'єкт також при відкритті вікна:

```
for (i = 0; i < 1000; i++) // створення об'єктів
{ МД[i] = new Авто(); }
```

Тепер масив готовий до прийому даних. Зазначимо, що перший вільний елемент має номер, співпадаючий зі значенням ***КількістьЕлем***. Можна було б введені значення записати в поля першого вільного елемента масиву. Але, просто так це зробити не удасться: поля об'єкта являються закритими. Доведеться включити в склад класу ***Авто*** метод, який дозволяє оновити значення полів об'єкта. Наприклад, такий:

```
public void Оновити(string g, string m, string c, string
fio)
{ Держномер = g; Модель = m; Колір = c; ПІБ = fio; }
```

Цей метод – динамічний. Він може бути викликаний для роботи з об'єктом масиву `МД`. В обробник натискання на кнопку ***«Зберегти»*** запишемо:

```
МД[КількістьЕлем].Оновити(textBox1.Text, textBox2.Text,
textBox3.Text, textBox4.Text);
КількістьЕлем++;
```

Після збільшення значення ***КількістьЕлем*** наступний набір даних запишеться уже в наступний об'єкт.

Завдання для самостійного виконання

Додайте в обробник ***button2*** *очистку полів введення даних і приховування групи об'єктів набору даних, як це зроблено в обробнику* ***button3***. *Але не пишіть повторно оператори, які уже є в обробнику для* ***button3***. *Скористайтесь уже методом, який уже є.*

Відображення списку держномерів

Ми завершили обробку кнопок, пов'язаних з введенням даних. Але якщо з відміною набору все очевидно, то зі збереженням не ясно: чи зберігаються дані, чи ні. Адже в списку новий держномер не з'являється! А ми його туди і не додали. Список держномерів – це список в колекції ***comboBox1***. У нас уже є досвід запису в колекцію (ми уже укладали список кольорів в ***comboBox2***). Тільки в тому випадку ми використовували перерахування, а тут будемо використовувати масив об'єктів ***МД***.

Перше поле кожного об'єкта – це держномер.

Завдання для самостійного виконання

*1. В класі **Авто** створіть динамічний метод, що дозволяє отримати держномер об'єкта **Авто**.*

*2. В обробнику кнопки **«ЗБЕРЕГТИ»** очистіть колекцію **Items** об'єкта **comboBox1**.*

*3. Прогляньте масив **МД** в циклі, виберіть з кожного об'єкта поле **Держномер** за допомогою створеного раніше методу і додайте Держномер в колекцію **Items** об'єкта **comboBox1**. Алгоритм необхідно оформити у вигляді методу **СтворитиСписокДержс**. Метод викликати в обробник е натисканням кнопки **«ЗБЕРЕГТИ»**.*

Обробка вибору держномера зі списку

Перша особливість – це інша подія: не набір, а вибір значення зі списку. Ім'я події – ***SelectedIndexChanged***. Обробник цієї події можна отримати подвійним кліком по об'єкту. Всередину обробника потрібно записати оператори, які видобувають значення з обраного об'єкта і записують їх в поля введення ***textBox1***. А потім – відобразити групу.

Для заповнення поля введення держномера ***textBox1*** сам об'єкт не потрібний. Адже після вибору держномера він автоматично відображається в полі введення об'єкта ***comboBox1*** (властивість ***Text***), звідки його можна перекинути в поле ***textBox1***. Разом з оператором відображення групи це виглядає так:

```
textBox1.Text = comboBox1.Text;
groupBox1.Visible = true;
```

А ось далі – дві проблеми:

- отримати номер обраного об'єкта ***Авто***;
- отримати з об'єкта значення моделі, кольору і ПІБ.

Перша проблема вирішується легко. В об'єкта ***comboBox1*** є властивість ***SelectedIndex***, яка містить номер обраного рядка списку (рахуючи з нуля) [7]. Цей номер співпадає з індексом об'єкта в масиві. Обрати номер можна оператором:

```
int nom=comboBox1.SelectedIndex;
```

А отримати значення зразу не вдасться – прямий доступ до полів ***Модель***, ***Колір, ПІБ*** заборонено. Доведеться ще розширити функціональність класу ***Авто***. Наприклад, додати в нього опис властивостей, які дозволять отримати значення закритих полів [7] :

```
public string Мод { get { return Модель; } }
public string Кол { get { return Колір; } }
public string ПІБ { get { return ПІБ; } }
```

Скористуємося цими властивостями для отримання значень:

```
textBox2.Text = МД[nom].Мод;
textBox3.Text = МД[nom].Кол;
textBox4.Text = МД[nom].ПІБ;
```

Завдання для самостійного виконання

Наберіть і збережіть два різних набори значень. Потім послідовно відобразіть їх, обравши держномера. Зверніть увагу, що в одному з випадків значення в полі кольору не відповідає значенню в списку кольорів – там залишилось останнє значення, що набиралось. Зробіть так, щоб значення кольору в полі comboBox2 і textBox3 були однаковими.

Тепер оберемо якесь значення держномера і внесемо зміни в дані. Наприклад, замінимо ПІБ. Збережемо. В списку з’явиться подвійний номер, і це правильно: ми не установили контроль на повторний запис. Змінені дані повинні записуватися в той же об’єкт, якщо держномер залишився незмінним. Це означає, що алгоритм в обробнику ***button2*** повинен розрізняти, новий це держномер або вже існуючий. Зараз у нас оновлюється вільний елемент масиву, тобто в масив пишеться як би новий елемент.

Завдання для самостійного розв’язання

*1. Внесіть зміни в алгоритм обробника кнопки **button2**, які не допустять подвійних записів.*

2. Дозвольте користувачу набирати нові кольори і забезпечте їх збереження в списку. Подвійних записів в список кольорів не допускати.

*3. Створіть обробник кнопки **«ВИДАЛИТИ»** і складіть алгоритм роботи. Не забудьте, що видаляти потрібно не тільки держномер зі списку, але і дані з масиву.*

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Дайте визначення поняття об’єкту.
2. Що таке абстрагування і інкапсуляція?
3. Що таке наслідування?
4. Поясніть що ви розумієте під поняттям оліморфізм.
5. Сформулюйте переваги ООП.
6. Перелічіть недоліки ООП.
7. Сформулюйте класу. Як описати клас?.
8. Які специфікатори класу ви знаєте?
9. Які елементи класу ви знаєте?
10. Назвіть правила оголошення об’єкта.
11. Дайте характеристику даних класу: поля і константи.
12. Опишіть специфікатори полів і констант класу.
13. Призначення методів. Синтаксис методу.
14. Опишіть якими можуть бути параметри методів та способи передачі аргументів в метод.
15. Надайте характеристику типів параметрів методів.
16. Сформулюйте правила застосування параметрів.
17. Перерахуйте особлтвості методів зі змінною кількістю аргументів.

ТЕМА 10 ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ІНДИКАТОРНИХ ЕЛЕМЕНТІВ УПРАВЛІННЯ. ФАЙЛИ ПОСЛІДОВНОГО ДОСТУПУ

10.1 Збереження об’єктів (серіалізація)

Скористаємося проектом з попередньої лабораторної роботи. Скопіюємо рішення і перейменуємо теку в ***Серіалізація_Т10***. Наша програма забезпечує збереження даних в масиві об’єктів. В процесі роботи користувач може набрати відомості про автомобілі, але після завершення програми вся інформація губиться. Виконаємо доробку рішення і забезпечимо наступний сценарій.

Під час запуску програми дані вводяться в масив об’єктів з файлу ***«Дані»***, розміщеного в тій же теці, де знаходиться готова програма. При першому запуску програми файл може бути відсутнім. Після введення даних масив заповнюється і відображається список держномерів.

При завершенні роботи програми дані з масиву записуються в файл ***«Дані»***, при цьому файл створюється заново. В випадку, якщо даних в масиві немає, то створюється пустий файл. Будемо зберігати дані в двійковому форматі. Врахуємо, що наші дані – це масив, елементами якого являються посилання на об’єкти. Об’єкти можуть розміщуватися в оперативній пам’яті в різних місцях. Збереження таких об’єктів можна забезпечити, використовуючи можливості класів з простору імен [7]

```
System.Runtime.Serialization.Formatters.Binary.
```

Завдання для самостійного виконання

Підключіть указаний простір імен.

Для того, щоб можна було використовувати інструменти цього простору, клас ***Авто*** потрібно позначити як серіалізуємий за допомогою атрибута `[Serializable]`:

```
[Serializable] public class Авто
```

Простір імен містить клас ***BinaryFormatter***, в якому визначені два методи: ***Serialize*** (зберегти об’єкт) і ***Deserialize*** (відновити об’єкт). Скористатися цими методами можна, якщо зрозуміти принципи організації

обміну даними між диском і оперативною пам'яттю. Обмін даними завжди здійснюється за допомогою потоку. Потік зв'язує файл з пам'яттю. Потік – це теж об'єкт, але об'єкт класу ***FileStream*** з простору імен ***System.IO***. Таким чином, потрібно буде мати ще один простір імен. [7]

Завдання для самостійного виконання

Підключіть простір імен ***System.IO****.*

Почнемо з фінальної частини – реалізуємо алгоритм виведення даних з масиву в файл. Створимо обробник події **FormClosed** для вікна **Form1** і підключимо його до події. Алгоритм збереження буде описано в цьому обробнику. В посібнику [10] описано конструктор об'єктів класу **FileStream**. Скористаємося ним для створення об'єкта-потоку, який використаємо для запису в файл:

```
FileStream F;
F = new FileStream("Данные",
                    FileMode.Create, FileAccess.Write);
```

Тепер створимо об'єкт класу ***BinaryFormatter***, який будемо використовувати як інструмент запису в потік:

```
BinaryFormatter P = new BinaryFormatter();
```

Об'єкт ***F*** (потік) і ***P*** (засіб спілкування з потоком) будемо використовувати при роботі з методом ***Serialize()***. Організуємо цикл виведення об'єктів з масив:

```
int i;
for (i = 0; i < КількістьЕлем; i++)
{ P.Serialize(F, МД[i]); }
F.Close();
```

Запустіть програму, наберіть три набори даних, а потом завершіть роботу програми. Перевірте: після завершення в тій теці, де зберігається готова програма (exe-файл) утворився файл з іменем дані. Але, після запуску програми список держномерів пустий – алгоритм введення ще не готовий.

Алгоритм введення опишемо в окремому методі з іменем ***Введення()***, а оператор виклику цього методу запишемо в обробнику подій ***Load.*** Алгоритм методу трішки складніший за попередній. Файл може знаходитися або бути відсутнім в теці. Наприклад, при першому запуску програми. Це доведеться врахувати при використанні конструктору потоку. Зі всіх констант

перерахування ***FileMode*** найкраще підходить ***OpenOrCreate***: у випадку відсутності файлу буде створено пустий файл:

```
FileStream F = new FileStream ("Данные",
 FileMode.OpenOrCreate, FileAccess.Read);
```

Для спілкування з потоком створюємо також об'єкт ***P***:

```
BinaryFormatter P = new BinaryFormatter();
```

Так як ми формуємо масив даних на основі файлу, що вводиться, то початкове значення лічильника елементів дорівнює нулю:

```
КількістьЕлем = 0;
```

Тепер потрібно організувати введення. Файл містить різні об'єкти, довжина кожного – різна. Але, у кожного з них є своя позиція в файлі: номер байта початку об'єкта (або позиція об'єкта в потоці). Визначити цю позицію можна властивістю ***Position***. Рахунок позицій йде з нуля (з нульового байта файлу). Після введення останнього об'єкта покажчик позиції отримає значення, що дорівнює довжині файлу. Цим і скористаємося:

```
while (F.Position<F.Length)
{ МД[КількістьЕлем] = (Авто) P.Deserialize(F);
 КількістьЕлем ++; }
```

Залишилось закрити потік і створити список держномерів:

```
F.Close(); СтворитиСписокДерж();
```

Запустить програму і перевірте роботу.

10.2 Клас StreamReader і StreamWriter

Завдання для самостійної роботи

*Організуйте для збереження колекцію в **comboBox2**. Забезпечте відновлення колекції з файлу, а при відсутності файлу – з перерахування.*

Вказівки.

1. Використайте класи ***StreamWriter*** і ***StreamReader***. [7]
2. При відсутності файлу обробіть виключення; тип виключення ***IOException*** опишіть так: ***catch (IOException)*** .

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Яке признаяення має простір імен ***System.Runtime.Serialization.Formatters.Binary***?
2. Опишіть процес збереження об’єктів (серіалізація).
3. Чим відрізняється потік від файлу?
4. Опишіть призначення класів ***StreamReader*** і ***StreamWriter***.
5. Дайте характеристику класу ***BinaryFormatter.*** Опишіть його призначення та використання.
6. Опишіть призначення та механізм дії методу ***Serialize*** (зберегти об’єкт).
7. Опишіть призначення та механізм дії методу ***Deserialize*** (відновити об’єкт).
8. Які поля класу ***BinaryFormatter*** ви знаєте?
9. Яким чином працює метод ***Close()***?
10. Які режими відкриття фійлу ви знаєте?
11. Яким здійснюється прив’язка файлу з протоком?
12. Опишіть призначення класів ***StreamReader*** і ***StreamWriter***.
13. Назвіть методи класів StreamReader і StreamWriter для виконання файлових операцій.
14. Сформулюйте правила використання методи класів **StreamReader** і ***StreamWriter.***

ТЕМА 11 ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ МЕНЮ. РОБОТА З ФАЙЛАМИ ПРЯМОГО ДОСТУПУ

11.1 Постановка задачі

Потрібно організувати просту базу даних. База зберігає відомості про жителів міста: ***ПІБ, Стать Дата народження, Вулиця, Будинок, Номер квартири, Загальна площа оселі***. Необхідно забезпечити: введення нових даних, складання списку даних по відомим значенням (повні відомості або часткові), відображення повних відомостей за елементом зі списку.

Обговорення проблеми

На перший погляд задача похожа на задачу 5. Також можна організувати серіалізуємий клас. Введенні дані зберігати в файлі, який на початку роботи програми вводиться в масив, а після закриття вікна – записувати дані з масиву в файл. Але, тут об’єм даних може бути таким, що файл повністю просто не поміститься в масиві – уявимо собі, що крім перерахованих відомостей є ще паспортні дані, відомості про освіту і т. д. Та і кількість жителів дуже велика. Таким чином, подібний спосіб організації програми не годиться. Разом з тим, різнотипність даних змусить нас використати клас. І серіалізацію нам доведеться використовувати – по-іншому об’єкт на диск не записати. А в усьому іншому – рішення буде не таким.

11.2 Сценарій роботи користувача

Ми не станемо відображати на екрані список всіх ***ПІБ***. Спочатку на екрані відобразимо тільки дві кнопки: ***«ВВЕДЕННЯ ДАНИХ»*** і ***«ПОШУК ДАНИХ»***. Розглянемо обидва випадки.

1. При натисканні на кнопку ***«ВВЕДЕННЯ ДАНИХ»*** з’являються поля введення даних для набору і збереження нових даних. Разом з полями з’являються кнопки ***«ЗБЕРЕГТИ»***, ***«ВІДМІНИТИ»*** і ***«ОБРАТИ»***. Кнопка ***«ОБРАТИ»*** забезпечує вибір статі: ***М*** або ***Ж***. Повинні бути заповнені всі поля введення. При натисканні кнопки ***«ЗБЕРЕГТИ»*** перевіряється правильність заповнення полів (чи всі заповнені) і дані зберігаються в файлі ***«База»*** – виконується запис в кінець файлу. Користувач інформується про запис повідомленням на екран: ***«Відомості додано»***. При натисканні кнопки

«ВІДМІНИТИ» нічого не відбувається, поля введення приховуються, і знову на екрані відображаються дві вихідні кнопки.

2. При натисканні на кнопку ***«ПОШУК ДАНИХ»*** відображаються всі поля для набору даних, крім номера квартир і площі житла. Відображаються також кнопки ***«ПОШУК»***, ***«ВИБРАТИ»*** і ***«ВІДМІНИТИ»***. В поля користувач може набрати значення – в кожне поле або тільки в деякі. Введені значення являються ключами пошуку інформації в базі даних. При натисканні кнопки ***«ВІДМІНИТИ»*** нічого не відбувається, поля введення приховуються, і знову на екрані дві вихідні кнопки. При натисканні на кнопку ***«ПОШУК»*** виконується пошук записів в файлі ***«База»***. Після завершення пошуку на екрані відображається список знайдених даних в вигляді ***ПІБ***. Список може бути і пустим, якщо при пошуку не знайдено ні одного запису. Разом зі списком відображається кнопка ***«ЗАКРИТИ»***. Тепер користувач може обрати будь-який елемент списку, і на екрані відображається повна інформація. При натисканні на кнопку ***«ЗАКРИТИ»*** поля введення приховуються, і знову на екрані – дві вихідні кнопки.

11.3 Інтерфейс користувача

Інтерфейс представлено на рис. 11.1.

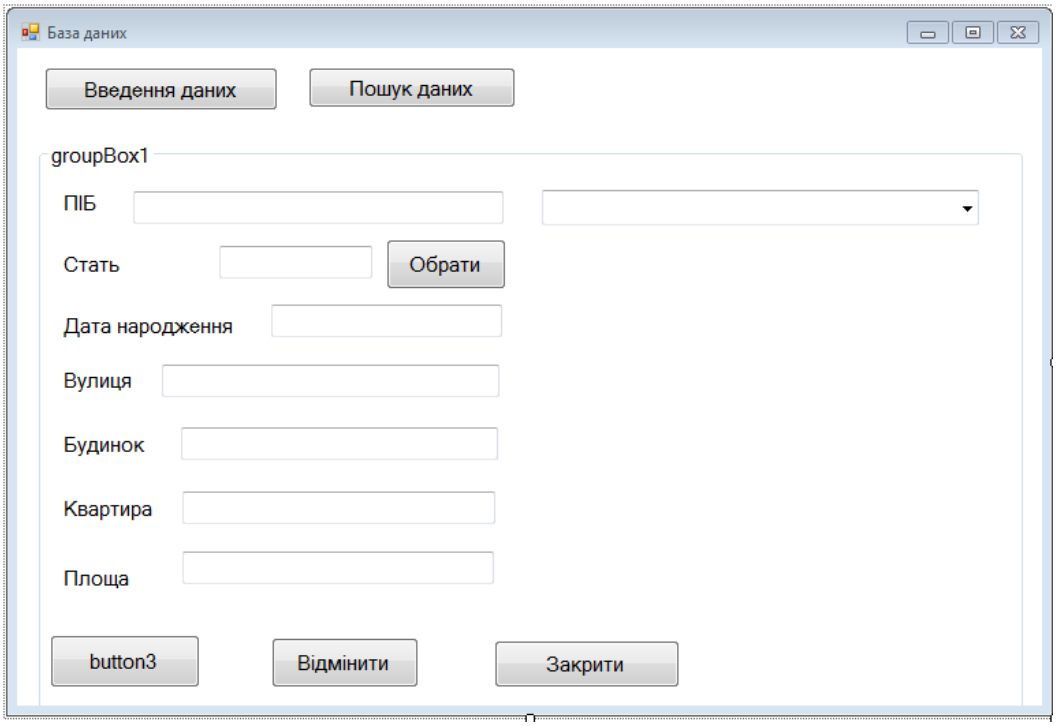

Рисунок 11.1 – Інтерфейс користувача

Всі надписи – це об’єкти ***label.*** Поля введення – це об’єкти ***textBox***. Для відображення списку знайдених записів використовується об’єкт ***comboBox1***.

Надпис на кнопці ***«BUTTON3»*** спочатку не формовано. Надпис з’явиться після вибору варіанта роботи: при вводі даних – надпис ***«Зберегти»***, при пошуку – ***«Пошук».*** Порядок появи об’єктів описано в сценарії.

Завдання для самостійного виконання

1. В візуальному режимі установіть невидимість всіх об’єктів, крім двох верхніх кнопок.

*2. Оформіть метод **ВидимВвод()**, в якому установіть видимість всіх надписів, полів введення і кнопок **«BUTTON3»**, **«ОБРАТИ»** і **«ВІДМІНИТИ»**. В цьому ж методі установіть пусті значення всіх полів введення.*

*3. В обробнику кнопки **«ВВЕДЕННЯ ДАНИХ»** запишіть оператор виклику методу **ВидимВвод()**. В **button3** сформуйте надпис **«Зберегти»**. Приховайте кнопку **«ПОШУК ДАНИХ»**, щоб її випадково не натиснув користувач.*

*4. Оформіть метод **ПриховатиВвод()**, який відміняє видимість об’єктів (дії, протилежні методу **ВидимВвод()**).*

*5. В обробнику кнопки **«ВІДМІНИТИ»** запишіть оператор виклику методу **ПриховатиВвод()**.*

*6. В обробнику кнопки **«ОБРАТИ»** забезпечте вибір статі. Значення **М** або **Ж** запишіть в **textBox2**.*

*7. Оформіть метод **Набрано()**, в якому виконується контроль повноти набору даних. Метод повертає **true**, якщо всі поля заповнені, і **false** в протилежному випадку.*

*8. В обробнику кнопки **«ЗБЕРЕГТИ»** запишіть оператор виклику методу **Набрано()**. Якщо метод повертає **false**, то видайте повідомлення про неповний набор на екран і завершіть метод-обробник (**return**).*

*9. Оформіть метод **Дата()**, в якому перевірте правильність заповнення поля Дата народження. Передбачається, що воно повинно бути заповнене в форматі: ЧЧ.ММ.ГГГГ. І число, і місяць, і рік – цілі числа. Відповідність числа найменуванню місяця не перевіряти (31 червня допускається). Метод повертає **true**, якщо формат вірний, і значення числові. В протилежному*

*випадку – **false**. Крім того, при правильному форматі метод формує три вихідних цілих параметра: день, місяць і рік. Значення потрібно зберегти в змінні.*

*10. В обробнику кнопки **«ЗБЕРЕГТИ»** забезпечте виклик методу **Дата()**. Якщо він повертає **false**, то повідомити про це на екран і обробник завершити.*

*11. В обробнику кнопки **«ЗБЕРЕГТИ»** перевірити правильність набору площі (числове значення). При невірному наборі повідомити про це на екран, обробник завершити. При правильному значенні зберегти значення площі в змінній. Тепер після всіх видів контролю продовжити розробку алгоритму кнопки **«ЗБЕРЕГТИ»**. Якщо всі поля заповнені, то потрібно вивести нові дані в файл. Але файлу у нас доки немає!*

Визначення формату файлу

Визначимося с форматом даних в файлі. Простіше за все описати новий клас об'єктів з полями, що відповідають даним, які вводяться.

Завдання для самостійного виконання

*1. Створіть новий клас **Житель** з полями **ПІБ** (рядок), **Стать** (символ), **ДатаНародження** (цілочисловий масив), **Вулиця** (рядок), **Будинок** (рядок), **Квартира** (рядок), **Площа** (дійсне число). Не забудьте, що ми будемо виводити дані в файл – потрібна серіалізація. Простір імен також.*

2. Включіть в клас конструктор об'єктів, який заповнює поля об'єкта нульовими (с точки зору типа) значеннями.

*3. Включіть в клас метод **Заповнити()**, який заповнює поля об'єкта **Житель**. Тепер можна в обробнику кнопки **«ЗБЕРЕГТИ»** оголосити змінну-об'єкт, створити цей об'єкт за допомогою конструктора і заповнити його поля значеннями (приблизно так):*

```
Житель T = new Житель();
T.Заповнити(textBox1.Text,textBox2.Text[0],c,m,g,
textBox4.Text, textBox5.Text,textBox6.Text,p);
```

Тут: `c, m, g, p` – це число, місяць, рік народження і площу (отримані вами раніше). Об'єкт сформовано і готовий до виводу в файл.

Створення файлу і виведення даних

Тепер описуємо потік, відкриваємо файл для виведення даних в кінець файлу і виводимо дані [7]:

```
FileStream F = new FileStream("База ", FileMode.Append,
FileAccess.Write);
BinaryFormatter W = new BinaryFormatter();
W.Serialize(F, T);
F.Close(); ПриховатиВвод();
```

11.4 Пошук даних

Завдання для самостійного виконання

*1. Створіть метод **ВидимПошук()**, в якому установлюється видимість всіх надписів і полів введення (крім номера і площі квартири). Відобразити кнопки **«BUTTON3»**, **«ОБРАТИ»** і **«ВІДМІНИТИ»**. В цьому ж методі установіть пусті значення всіх полів введення.*

*2. В обробнику кнопки **«ПОШУК ДАНИХ»** запишіть оператор виклику методу **ВидимПошук()**. В **button3** сформуйте надпис **«Пошук»**.*

*3. Приховайте кнопку **«ВВЕДЕННЯ ДАНИХ»**, щоб випадково користувач не зміг її натиснути.*

Тепер нам потрібно визначитися з алгоритмом пошуку. Справа в тому, що пошук ми замовили на ту ж кнопку, яка використовувалася для збереження даних – ми просто її перейменували. Якщо натиснути на цю кнопку, то буде виконуватися алгоритм збереження даних – адже обробник події налаштовано саме на це. Нам потрібен новий обробник події, що виконує пошук, але виконуватися він повинен від тієї ж кнопки. Створимо вручну обробник з схожим іменем:

```
private void button3_Click_1(object sender, EventArgs e)
{ }
```

В обробнику кнопки ***«ПОШУК ДАНИХ»*** одразу після перейменування кнопки ***«BUTTON3»*** підмінимо обробник події [7] :

```
this.button3.Click-=new
System.EventHandler(this.button3_Click);
this.button3.Click+=new
System.EventHandler(this.button3_Click_1);
```

Перший оператор операцією **-=** відключає попередній обробник, другий підключає новий обробник. Тепер в обробнику ***button3_Click_1()*** можна реалізувати алгоритм пошуку даних.

Завдання для самостійного виконання

*1. Оформіть метод **НабранПошук()**, в якому виконується контроль набору даних для пошуку. Метод повертає **true**, якщо є дані хоча б в одному полі, і **false** в протилежному випадку.*

*2. В обробнику кнопки **«ПОШУК»** запишіть оператор виклику методу **НабранПошук()**. Якщо метод повертає **false**, то видайте на екран повідомлення про відсутність ключів пошуку і завершіть метод-обробник (**return**).*

Організуємо пошук даних. За умовами задачі, кожний запис файлу, що має необхідні ключі, повинен відображуватися в списку на екрані – для цього у нас є об'єкт ***comboBox1.*** Уявимо, що такий список отримано, і користувач вирішив проглянути дані, обравши якесь прізвище зі списку. Дані потрібно буде знову доставати з файлу – адже при пошуку ми не зберігали повні відомості ніде. Значить, в процесі пошуку потрібно зберігати адресу даних в файлі: номер байта, з якого починаються потрібні дані.

Створимо в класі ***Form1*** масив:

```
long[] Адреса = new long[1000];
```

Масив ***Адреса*** являється індексним масивом. У нас буде відповідність: ***ПІБ*** – в списку об'єкта ***comboBox1***, а номер байта початку даних в файлі – в відповідному елементі масиву ***Адреса***.

Тепер потрібно організувати доступ до файлу. Змінну-потік потрібно розмістити так, щоб до неї був доступ не тільки при пошуку, але і при виборі ***ПІБ*** – а це уже буде інший метод. Значить, потік необхідно створити як поле класу, одразу після оголошення масиву:

```
FileStream БД;
```

Відкриємо файл в обробнику ***button3_Click_1***:

```
FileStream БД = new FileStream ("База",
FileMode.OpenOrCreate, FileAccess.ReadWrite);
```

А в кінці обробника (щоб не забути) одразу запишемо оператор закриття потоку:

```
БД.Close();
```

Переходимо до пошуку даних в файлі – всі дії в тому ж обробнику.

Завдання для самостійного виконання

1. Очистіть список в об'єкті ***comboBox1****.*

2. Створіть об'єкт ***R*** *класу* ***BinaryFormatter*** *для виконання десеріалізації об'єктів і службовий об'єкт* ***Ж*** *класу* ***Житель****.*

3. Зробіть видимим об'єкт ***comboBox*** *і кнопку* ***«ЗАКРИТИ»****.*

Запишіть цикл введення даних з файлу:

```
long i=0;
while (i < БД.Length)
{ Ж = (Житель)R.Deserialize(БД);
// далі буде алгоритм порівняння ключів
i = БД.Position;
}
```

Після того, як в об'єкт ***Ж*** введені дані, можна порівнювати ключі. Але до полів об'єкта неможливо добратися – вони закриті.

Завдання для самостійного виконання

1. Створіть в класі ***Житель*** *властивості для доступу до всіх закритим полів. Імена властивостей створіть з імені поля з додаванням знаку «підкреслення». Наприклад:* ***ПІБ_****. Оскільки стать і дата будуть використовуватися для порівняння, властивості повинні повертати їх в вигляді рядка.*

2. Створіть метод ***Порівняння()****, в якому буде забезпечуватися порівняння заданих полів з полями введеного об'єкта. Метод має один вхідний параметр – об'єкт класу* ***Житель****. Метод повертає* ***true****, якщо ключі співпадають.*

Вказівка. Порівнювати потрібно тільки задані ключі!

Далі запишемо в обробнику оператори перевірки результатів порівняння і збереження відомостей:

```
if (Порівняння(Ж) == true)
{
// збереження відомостей про записи в списку і в масиві
Адреса[comboBox1.Items.Count] = i;
comboBox1.Items.Add(Ж.ПІБ_);
}
```

Завдання для самостійного виконання

1. Перевірте роботу програми. Наберіть кілька різних відомостей, збережіть їх і виконуйте пошук за різними ключами. Все повинно працювати!

*2. Обробіть натискання на кнопку **«ЗАКРИТИ»**. Об'єкт **comboBox1** і сама кнопка повинні стати невидимі. Створіть обробник подію вибір зі списку. Вставте в нього такий алгоритм:*

```
if (comboBox1.SelectedIndex >= 0)
{ FileStream БД = new FileStream("База", FileMode.
OpenOrCreate, FileAccess.ReadWrite);
BinaryFormatter R = new BinaryFormatter();
Житель Ж;
БД.Position=Адрес[comboBox1.SelectedIndex]; // ***
Ж=(Житель) R.Deserialize(БД); БД.Close();
ВидимВвод();

textBox1.Text = Ж.ПІБ_; textBox2.Text = Ж.Стать_;
textBox3.Text = Ж.Дата_; textBox4.Text = Ж.Вулиця_;
textBox5.Text = Ж.Будинок_; textBox6.Text = Ж.Квартира_;
textBox7.Text = Ж.Площа_.ToString();
}
```

Зверніть увагу на оператор, позначений трьома зірочками. Він установлює файл в позицію початку потрібного об'єкта, після чого об'єкт зчитується і з допомогою властивостей розкладається по об'єктах ***TextBox***.

Завдання для самостійної роботи

*1. В алгоритмі кнопки **«ВІДМІНИТИ»** відновіть відображення кнопок **«ВВЕДЕННЯ ДАНИХ»** і **«ПОШУК ДАНИХ»**.*

*2. Зараз після виконання пошуку даних неможливо повернутися до вводу даних – адже ми переключили кнопку **«BUTTON3»** на інший обробник. Виправте дефект.*

11.5 Створення меню в C#. Елемент управління menuStrip

Умова задачі

Створити додаток типу ***Windows Forms Application***. На формі додатка створити меню за зразком, как показано на рис. 11.1.

File	Edit	Exit
New	Cut	
Open	Copy	
Save	Paste	
Close		

Рисунок 11.1 – Структура меню

Виконання

1. Для створення меню використовується елемент управління ***MenuStrip*** (він розміщується на вкладці ***Menus&Toolbars***). Після винесення компонента на форму (за дорогою мишки) форма додатка приймає вигляд (рис. 11.2). В нижній частині вікна проектування форми розміщується об’єкт з іменем ***menuStrip1***.[7]

Після наведення курсору на елемент меню ***TypeHere*** з’явиться кнопка виклику випливаючого меню, після розкриття якого (клік мишкою) буде надана можливість вибору одного з трьох видів елементів управління (рис.11.3) [7] :

- ***MenuItem*** – стандартний елемент меню;
- ***ComboBox*** – елемент меню типу «список»;
- ***TextBox*** – елемент меню типа «поле введення».

В нашому випадку обираємо перший варіант ***MenuItem***.

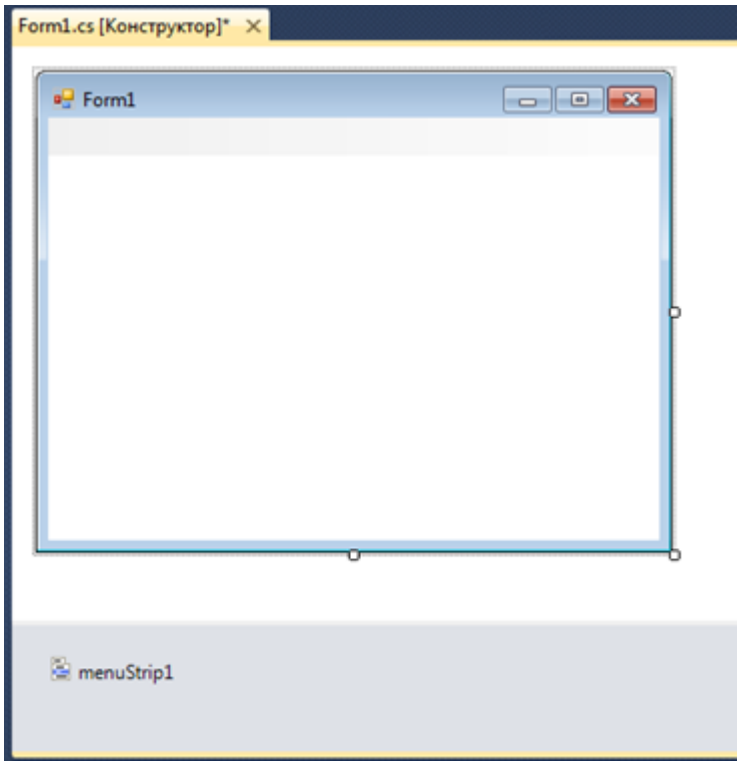

Рисунок 11.2 – Форма додатка після розміщення компонента ***MenuStrip***

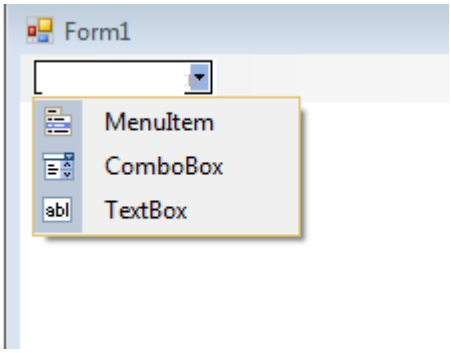

Рисунок 11.3 – Типи елементів меню

Для створення підменю ***File*** достатньо набрати текст «***File***» (рис. 11.4). За допомогою мишки і клавіатури можна додавати елементи меню. Для видалення елемента меню його необхідно виділити мишкою і натиснути клавішу ***Delete***.

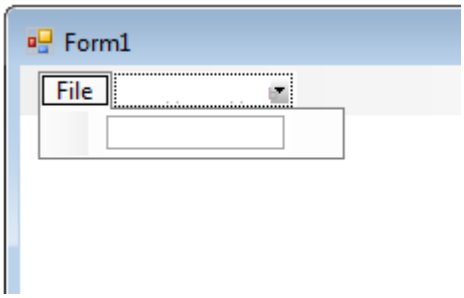

Рисунок 11.4 – Створення підменю ***File***

Після створення всіх елементів меню форма додатка буде мати вигляд як показано на рис. 11.5.

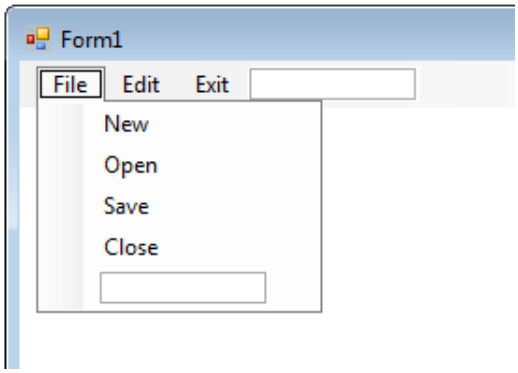

Рисунок 11.5 – Форма додатка після створення меню

Завдання для самостійної роботи

1. *Розробіть обробники подій, що пов'язані з вибором конкретного елемента меню.*
2. *Додайте в меню можливість використовувати позначки пунктів меню та клавіші швидкого доступу.*

Вказівка.

Скористайтесь інформацією, наведеною в додатку В.

ЗАПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗА ТЕМОЮ

1. Чим відрізняються файли послідовного та довільног доступу?
2. Який простір імен потрібно підключти, що використовувати файл довільного доступу?
3. Опишіть процес збереження об'єктів в файл довільного доступу.
4. Дайте характеристику атрибутів відкриття файлу довільного доступу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Голуб Б. М. C#. Концепція та синтаксис : навч. посіб. Львів : Видавничий центр ЛНУ імені Івана Франка, 2006. 136 с.
2. Коноваленко І. В., Марущак П. О., Платформа .NET та мова програмування C# 8.0 : навч. посіб. Тернопіль : ФОП Паляниця В. А., 2020. 320 с.
3. Візуальне програмування : навч. посіб. / Лозовська Л. І., Бандоріна Л. М., Савчук Р. В., Климкович Т. О. Дніпро : УДУНТ, 2022. 132 с.
4. Локазюк В. М. Надійність, помилки і тестування програмного забезпечення комп'ютерних пристроїв та систем : навч. посіб. Хмельницький : ТУП, 2003. 74 с.
5. Introduction to Data Structures and Algorithms. *W3Schools.com*. URL: https://www.w3schools.com/dsa/dsa_intro.php (date of access: 15.04.2025).
6. Вступ в C#. *Programm.top*. URL : https://programm.top/uk/c-sharp/tutorial/introduction/ (дата звернення 15.04.2024).
7. C# Guide – .NET managed language. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (date of access: 10.05.2024).
8. Мова програмування C#. *Портал знань, дистанційне навчання*. URL: http://www.znannya.org/?view=csharp (дата звернення: 10.05.2024).
9. Поняття про візуальне програмування. Засоби візуальної розробки програм. Середовище візуального програмування Delphi 7. Основні поняття. *Студопедія*. URL: https://studopedia.su/1_12485_ponyattya-pro-vizualne-programuvannya.html (дата звернення: 17.06.2024).
10. Основні відомості про мову програмування C#. *ArmedSoft*. URL: https://armedsoft.com/ua/blog/osnovni-vidomosti-pro-movu-programuvannya-c (дата звернення: 17.06.2024).
11. C# (Sharp) Підручник. Уроки для початківців. *W3Schools українською*. URL: https://w3schoolsua.github.io/cs/index.html#gsc.tab=0 (дата звернення: 17.06.2024).

ДОДАТОК А
КОМПОНЕНТИ ПРОСТОРУ ІМЕН SYSTEM.WINDOWS.FORMS

Простір імен ***System.Windows.Forms*** містить класи, використовуючи які можна розміщувати на вікні різноманітні об'єкти. Кожний клас має власний конструктор, що створює об'єкт, а також ряд властивостей, методів і подій. Властивості дозволяють задавати характеристики об'єкта. Методи забезпечують маніпуляцію з об'єктом. За допомогою подій програміст може організувати спостереження за станом об'єкта. Використовуючи механізм «джерело-спостерігач», програміст може забезпечити виконання різних алгоритмів в залежності від дій користувача. В середовищі ***Visual Studio*** заготовки об'єктів цих класів представлені на панелі інструментів (за виключенням ***Messagebox***). При перенесені заготовки на вікно створюється об'єкт з конкретним іменем, яке складається з імені класу і порядкового номера. Всі об'єкти, створені у вікні, реагують на подію ***Click***, але с кожним об'єктом пов'язані і власні події [7].

MessageBox. Клас забезпечує створення об'єкта «вікно повідомлення». Статичний метод ***Show()*** забезпечує відображення модального вікна на екрані. Можлива безліч варіантів відображення вікна: від простого повідомлення до складного тексту з можливістю вибору [7].

Label. Клас об'єктів «надпис». Крім тексту він може містити і рисунок. Властивість ***Text*** дозволяє задати надпис на об'єкті при його відображенні. Для показу рисунка використовується властивість ***Image***. Положення надпису і рисунка визначається властивостями ***TextAlign*** і ***ImageAlign***. Властивість ***AutoSize***, установлена як ***false***, забороняє автоматичну зміну розміру поля в залежності від набраного тексту [7].

LinkLabel. Клас об'єктів «надпис з посиланням». Компонент ***LinkLabel*** забезпечує зв'язок надпису з однією або кількома гіперпосиланнями. У властивість ***Text*** вміщується текст, частина якого або весь текст буде грати роль гіперпосилання. Властивість ***LinkArea*** дозволяє обрати фрагмент тексту або весь текст і визначити його як ім'я посилання [7].

Процес використання компонента є дещо складним через те, що не все можна зробити на етапі конструювання. Властивість ***Links*** доступна тільки

динамічно, тобто гіперпосилання потрібно описувати в процесі виконання програми.

Розглянемо приклад. Створимо пусту форму і розмістимо компонент ***LinkLabel***. У властивостях установлюємо ***AutoSize*** значення ***false***. Властивість ***Text*** визначаємо так: *Я і Ти.* При такому визначенні весь текст вважається посиланням (рис. А.1).

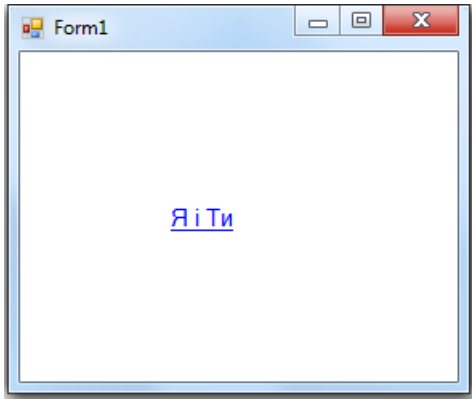

Рисунок А.1 – Надпис-посилання

У властивості ***LinkVisited*** встановимо значення ***true***, а текст розділимо на два посилання: ***«Я»*** і ***«Ти»***. Зробити це візуально неможна, можна визначити тільки перше посилання. Для цього відкриємо властивість ***LinkArea*** і установимо значення ***Start*** в ***0***, а ***Length*** зробимо рівним 1. Тим самим ми укажемо, що перше слово ***«Я»*** являється посиланням. Запустимо програму: виділено тільки перше слово (рис. А.2) [7].

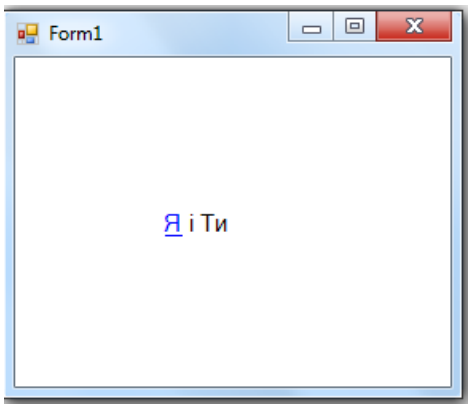

Рисунок А.2 – Одне посилання

Далі доведеться попрацювати з програмним кодом. Відкриємо програмний текст форми ***Form1*** і додамо наступну директиву:

```
using System.Diagnostics;
```

Це дозволить нам запускати процес прямо з додатка. Тепер відкриємо конструктор ***Form1***. Після оператора ініціалізації компонент добавимо ***URL*** для обох посилань в список властивості ***Links*** об'єкта ***linkLabel1***:

```
public Form1()
{ InitializeComponent();
linkLabel1.Links[0].LinkData = "WWW.i.ua";
linkLabel1.Links.Add(4, 4, "www.you.ua");
}
```

Знову запустимо програму: посилань уже два (рис. А.3).

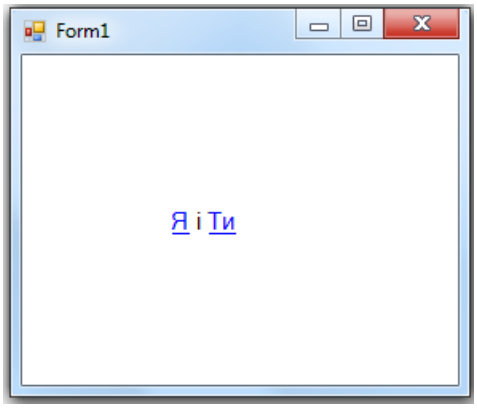

Рисунок А.3 – Два посилання

Подвійним клацанням мишки по об'єкту відкриємо ***Form1.cs*** і в пустий обробник запишемо наступний текст:

```
private void linkLabel1_LinkClicked (object sender,
LinkLabelLinkClickedEventArgs e)
{ string s = (string)e.Link.LinkData; // 1
Process.Start(s); // 2
}
```

Оператор ***1*** обирає з параметра *e* значення ***URL*** – воно передається в обробник у відповідності з обраним посиланням. Обране значення в операторі ***2*** використовується для запуску процесу.

Button. Клас об'єктів «кнопка». Використовується для відображення однойменного об'єкта, призначеного для імітації натискання користувачем. Властивість ***Text*** дозволяє дати кнопці ім'я. Кнопка має властивість ***Image***, яка

дозволяє відображати на об’єкті картину. Пов’язана з кнопкою основна подія ***Click*** дозволяє виконати алгоритм, що відповідає натисканню [7].

TextBox. Клас об’єктів «поле введення тексту». Використовується для введення даних у вигляді однієї або кількох рядків тексту. Якщо передбачається використовувати багаторядковий режим, то властивість ***Multiline*** повинна бути установлена як ***true***. Для доступу до одного рядка використовується властивість ***Text***, для роботи з багатьма рядками – властивість ***Lines***, організовану як рядковий масив. В останньому випадку доступ до окремих рядів за індексом [7]. Розглянемо простий приклад (рис. А.4).

Маємо вікно і два обробники для кнопок:

```
private void button1_Click(object sender, EventArgs e)
{ textBox1.Clear(); }
private void button2_Click(object sender, EventArgs e)
{ textBox1.Copy();
MessageBox.Show(textBox1.Text, "Выбранный текст");
textBox1.Clear(); textBox1.Paste();
}
```

Набираємо п’ять рядків тексту, виділяємо фрагмент (рис. А.4).

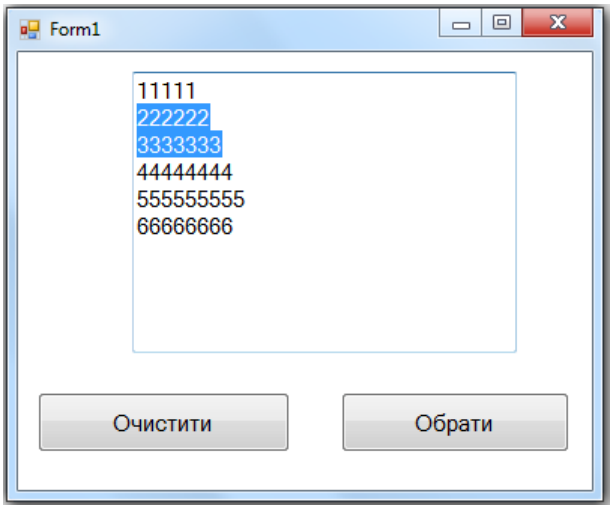

Рисунок А.4 – Об’єкт TextBox. Виділення двох рядків

При натисканні на кнопку ***«ОБРАТИ»*** текст відображається повністю, в буфер обміну запишеться тільки виділений фрагмент, об’єкт очиститься і із буфера в поле введення повернеться тільки що збережений там текст.

ListBox. Клас об’єктів «список». Може використовуватися для організації різних списків як при введенні даних, так і в процесі виконання програми. Користувач має можливість вибору одного або кількох пунктів. Чи можна обирати, і якщо так, то скільки, залежить від значення властивості ***SelectMode*** [7]:

- ***None*** – нічого не можна обрати.
- ***One*** – один пункт.
- ***MultiSimple*** – багато пунктів.
- ***MultiExtended*** – багато пунктів з урахуванням ***Shift*** (неперервний діапазон) і ***Ctrl*** (декілька діапазонів).

Властивість ***MultiColumn*** дозволяє визначити режим відображення даних: в одну або кілька колонок (***true***) [7]. Якщо кілька колонок, то за допомогою властивості ***ColumnWidth*** можна задати ширину колонок. Якщо колонки не уміщаються в вікні, то автоматично додається полоса горизонтальної прокрутки. Об’єкт має властивість ***Items***, яка представляє собою колекцію рядків. Тут зберігаються значення. Додавання елемента в колекцію здійснюється методом ***Add()***. Заповнення компонента даними може проводитися окремим алгоритмом. Наприклад, разом з об’єктом ***ListBox*** у вікні може розміщуватися компонент ***Button*** (кнопка), для якої передбачено метод-обробник, що заповнює список . Обробник подія ***Click*** може бути таким:

```
private void butt on1_Click(object sender, EventArgs e)
{ for (int i = 1; i < 21; i++)
     listBox1.Items.Add(string.Format("Элемент
{0}",i));}
```

ComboBox. Клас об’єктів «комбінований список». Об’єкт включає в себе однорядкове текстове поле і список ***ListBox***. Він використовується для зберігання даних у вигляді рядків з можливістю вибору будь-якого рядка списку. Крім того, можна ввести в список нове значення за допомогою текстового поля. Компонент можна використовувати в одному з трьох варіантів, що визначається властивістю ***DropDowmStyle*** [7] :

- ***Simple*** – рядок зі списком, можливо введення нового;
- ***DropDown*** – тільки рядок, можливо введення нового;
- ***DropDownList*** – рядок без можливості введення нового.

В тому випадку, коли новий варіант необхідно запам'ятати в списку, можна використати подію ***Leave***, яка виникає при втраті фокуса введення (при завершені набору нового значення). Обробник даної події повинен містити алгоритм, який запам'ятовує в списку нове значення. При цьому бажано не записувати в список ті значення, які в ньому уже є:

```
private void comboBox1_Leave(object sender, EventArgs e)
{ if (comboBox1.Items.Contains(comboBox1.Text)) { }
     else comboBox1.Items.Add(comboBox1.Text);
}
```

При цьому потрібно підписатися на подію:

```
this.comboBox1.Leave +=
    new System.EventHandler (this.comboBox1_Leave);
```

Зазначимо, що цей обробник отримує управління тільки при втраті компонентом фокуса введення, тобто якщо відбулося переключення на другий компонент форми. Якщо ж такого переключення не відбулося, то і обробник не запускається. Тому, можна рекомендувати інший спосіб негайного «відлову» нового значення. Коли закінчується набір, то натискається клавіша ***Enter***. Можна організувати «відлов» натискання на цю клавішу, використовуючи цей факт як подію. Додамо підписку на натискання клавіші:

```
this.comboBox1.KeyPress+=new
System.Windows.Forms.KeyPressEventHandler
(comboBox1_KeyPress);
```

Додамо метод-обробник:

```
private void comboBox1_KeyPress
(object sender, KeyPressEventArgs e)
{ if (e.KeyChar == (char)Keys.Enter)
    { if (comboBox1.Items.Contains(comboBox1.Text)) { }
         else comboBox1.Items.Add(comboBox1.Text);
    }
}
```

Основне призначення об'єкта – дати користувачу можливість вибрати зі списку потрібне значення. З об'єктом пов'язана подія ***SelectedIndexChanged***, яка виникає при виборі значення. Ця подія являється основною для об'єкта, тобто підключення до події виконується автоматично разом зі створенням

обробника. Є кілька властивостей (***SelectedIndex***, ***SelectedItem***, ***SelectedValue***, ***SelectedText***), аналізуючи які можна визначити, що саме вибрано [7].

CheckBox. Клас об'єктів «прапорець». Об'єкт використовувати для того, щоб користувач мав можливість визначити напрям виконання алгоритму. Наприклад, можна видавати список студентів на екран повністю з усіма даними або тільки прізвища. В формі можна організувати прапорець с таким текстом: «тільки прізвища». При виборі прапорця (помітка «галочкою») алгоритм відпрацює по другому варіанту, при відміні вибору – по першому. З об'єктом пов'язано багато подій, але головною являється подія ***CheckedChanged***, яка пов'язана зі зміною стану прапорця. Встановлені значення можуть бути «так – ні» або «так – ні – не впевнений». В останньому випадку галочка установлюється, але прапорець буде затінений. Перевірка стану прапорця забезпечується властивістю ***Checked*** (***true/false***) або властивістю ***CheckState*** (значення ***Checked***, ***Unchecked***, ***Indeterminate***). При використанні трьох значень властивість ***ThreeState*** повинна бути установлена як ***true*** [7].

RadioButton. Клас об'єктів «перемикач». За сенсом об'єкт схожий на прапорець, але ніколи не використовується по одному – завжди групою з двох і більше об'єктів. Кожний об'єкт має якийсь самостійний сенс. Обрати можна лише один об'єкт із групи. З об'єктом пов'язана подія ***CheckedChanged***, яку можна використати для визначення моменту зміни стану об'єкта. Для контролю стану конкретного об'єкта використовується властивість ***Checked*** (***true***, ***false***) [7].

PictureBox. Клас об'єктів «картинка». Об'єкт представляє собою контейнер для зберігання зображення. Властивість ***Image*** дозволяє зберігати рисунок в одному з форматів, що підтримуються операційною системою. Властивість ***SizeMode*** дозволяє розмістити рисунок так, як потрібно [7].

GroupBox. Клас об'єктів «панель групування». По суті об'єкт цього класу представляє собою об'єднання тих об'єктів, які обмежені лінією-межею. З групою об'єктів можна виконувати ті ж операції, що і зі звичайним об'єктом. Наприклад, при зміні властивостей групи змінюються відповідні властивості у всіх об'єктів, що входять в групу [7].

ДОДАТОК Б
ВИКОРИСТАННЯ СТАНДАРТНИХ ДІАЛОГОВИХ ВІКОН

Діалоговим називається вікно, яке виводиться в контексті іншого вікна. Окремий клас ***Dialog*** в **.NET** не передбачено. Діалогове вікно – це форма з деякими спеціальними характеристиками. Перша відмінна риса більшості діалогових вікон – їх розмір змінювати не можна. Крім того, в діалогових вікнах зазвичай не використовуються елементи управління, що поміщаються в верхню частину звичайних форм: ***ControlBox***, ***MinimizeBox*** і ***MaximizeBox*** [7]. Для користувача діалогове вікно є незмінним.

Для відкриття модального діалогового вікна (це таке діалогове вікно, яке потрібно обов'язково закрити перед поверненням до вихідної форми) використовується метод ***ShowDialog()***. Припустимо, що діалогове вікно в нашому додатку відкривається при натисканні користувачем на пункті в меню. Код для відкриття діалогового вікна може виглядати наступним чином:

```
// Відкриваємо модальне діалогове вікно
protected void menuModalBox_Click (object sender,
System.EventArgs e)
{
SomeCustomForm myForm = new SomeCustomForm();
// Те ж саме можна зробити в конструкторі
myForm.BorderStyle = FormBorderStyle.FixedDialog;
myForm.ControlBox = false;
myForm.MinimizeBox = false;
myForm.MaximizeBox = false;
// Передаємо як посилання батьківській формі
myForm.ShowDialog(this);
DoSomeWork();}
```

Зверніть увагу, що за викликом методу ***ShowDialog ()*** у нас відразу слідує допоміжна функція ***DoSomeWork ()***. Модальність форми визначає саме метод ***ShowDialog()***: при використанні нашого коду весь хід виконання програми буде припинено аж до того моменту, поки ***ShowDialog()*** не поверне відповідне значення. Для користувача це означає, що йому доведеться закрити

діалогове вікно, перш ніж він зможе виконати будь-які операції на головній формі. Якщо для нашого діалогового вікна модальність не потрібна (тобто в фокус можуть потрапляти як діалогове вікно, так і головна форма), достатньо замість ***ShowDialog()*** використовувати метод ***Show()***. В цьому випадку негайно після відкриття діалогового вікна почнеться виконання ***DoSomeWork*** [7]**.**

Для отримання результату виконання діалогового вікна слід скористатися призначенням певним кнопкам значень з перерахування ***DialogResult***. У більшості діалогових вікон передбачені кнопки **ОК** і **Cancel**. Натисканням на кнопку **ОК** користувач як би повідомляє: «*Я все вибрав, і можна йти далі і використовувати в програмі обрані мною значення*». Натискання на кнопку **Cancel** означає: «*Я передумав працювати з цим діалоговим вікном, закрийте його, будь ласка*». Призначення кнопкам відповідних значень проводиться таким чином:

```
private void InitializeComponent()
{// Призначення кнопці значення ОК
btnOK.DialogResult =
System.Windows.Forms.DialogResult.OK;
// Призначення кнопці значення Cancel
btnCancel.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
}
```

Тепер, при натисканні на ці кнопки діалогове вікно автоматично закриється. В ході подальшого виконання програми ми можемо з’ясувати значення властивості ***DialogResult***, щоб дізнатися, яку кнопку натиснув користувач, і в залежності від результату виконати певні дії:

```
protected void menuModalBox_Click (object sender,
System, EventArgs e)
{// Створюємо діалогове вікно
SomeCustomForm myForm = new SomeCustomForm();
```

```
// Передаємо посилання
myForrn . ShowDialog ( this);
if(myForm. DialogResult = = DialogResult.OK)
{
// Користувач натиснув OK! Виконуємо необхідні дії.
}
DoSomeWork ( ) ;
}
```

Значення перерахування ***DialogResult*** представлені в табл. Б.1. В коді діалогового вікна ці значення можна привласнювати кнопкам. У коді ж для головної форми додатка ми порівнюємо ці значення зі значенням властивості ***DialogResult*** для самого діалогового вікна.

Таблиця Б.1 – Значення перерахування ***DialogResult*** [7]

Значення	Опис
Abort	Аварійне або позапланове завершення. Зазвичай привласнюється кнопці ***Abort***.
Cancel	Відміна. В більшості діалогових вікон передбачена спеціальна кнопка ***Cancel***.
Ignore	Ігнорувати, пропустити. Як правило, існує спеціальна кнопка ***Ignore*** в діалоговому вікні.
No	***Hi.*** Для цього значення звичайно використовується спеціальна кнопка.
None	З діалогового вікна нічого не повертається. Це означає, що модальне діалогове вікно все ще відкрите.
OK	***OK*** (Добре). Для генерації цього значення зазвичай використовується спеціальна кнопка ОК.
Retry	Повторити. Для цього значення передбачена кнопка.
Yes	***Yes*** (так). Зазвичай посилається за допомогою кнопки ***Yes***.

Для виконання **стандартних операцій** (відкриття і збереження файлів, виведення на друк) рекомендується використовувати заздалегідь створений клас діалогових вікон. В ***.NET Framework*** є класи, які дозволяють використовувати діалогові вікна ***Windows*** для відкриття та збереження файлів, для виведення на принтер, а також для вибору кольору та шрифту.

Таким чином, стандартне діалогове вікно – це вікно, яке використовується для отримання від користувача найбільш поширеної інформації. Воно є частиною операційної системи ***Windows***. Наявні в ***.NET***

Framework класи діалогових вікон представлені на рисунку Б.1. Всі вони – за винятком класу ***PrintPreviewDialog*** – є похідними від абстрактного класу ***CommonDialog*** – базового класу, в якому представлені методи управління загальним діалоговим вікном в ***Windows*** [7]**.**

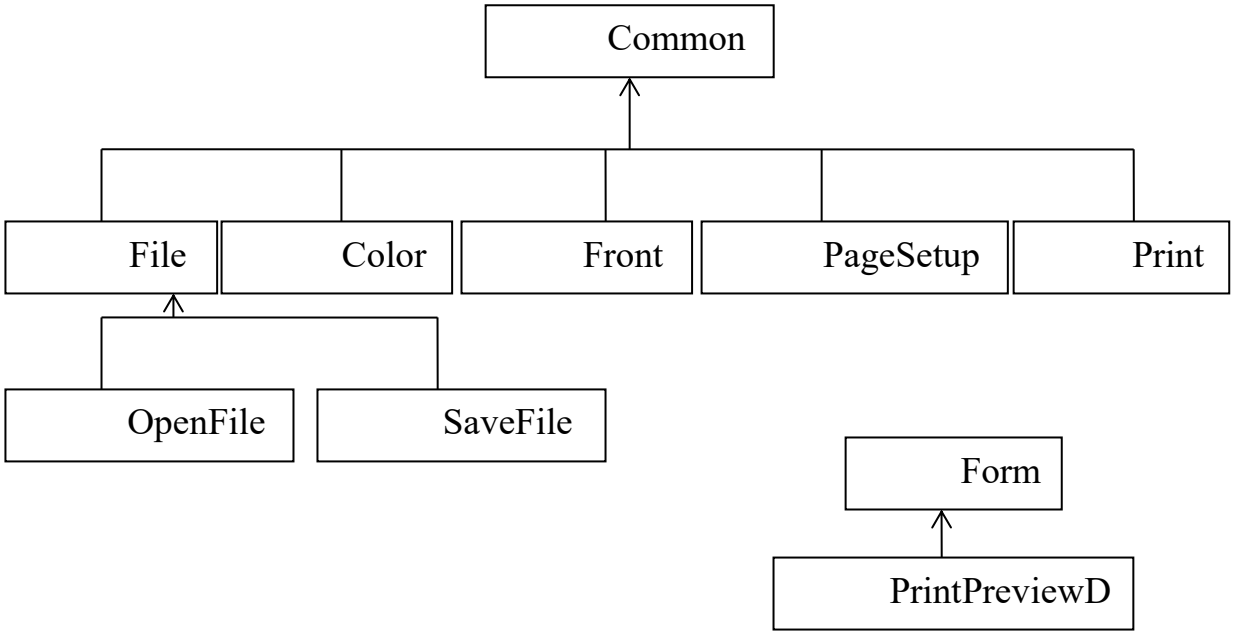

Рисунок Б.1 – Ієрархія класів стандартних діалогових вікон.

У класі ***CommonDialog*** описані методи і події (табл. Б.2), доступні будь-якому класу загального діалогового вікна.

Таблиця Б.2 – Загальні методи стандартних діалогових вікон [7]

Загальні екземпляри методів і подій	**Пояснення**
ShowDialog()	Реалізується в похідних класах для виведення діалогового вікна.
Reset()	Кожний похідний клас діалогового вікна реалізує метод Reset() для установки значень за замовчуванням для всіх властивостей даного класу.
HelpRequest	Генерується в момент, коли користувач клацає мишею на кнопці Help загального діалогового вікна.

Стандартні діалогові вікна можуть бути використані:

– Для надання користувачеві можливості шукати і вибирати файли для відкриття використовується діалогове вікно ***OpenFileDialog***.

- За допомогою діалогового вікна ***SaveFileDialog*** користувач може задавати назву попереднього Файла, здійснювати пошук директорії, в якій цей файл повинен бути збережений.
- Діалогове вікно ***PrintDialog*** використовується для вибору принтера і завдання опцій, що використовуються для друку.
- Діалогове вікно ***PageSetupDialog*** зазвичай використовується для визначення параметрів полів сторінки.
- Діалогове вікно ***PrintPreviewDialog*** дозволяє здійснювати попередній перегляд виведеної на друк інформації на екрані.
- Діалог ***FontDialog*** видає список всіх інстальованих в ***Windows*** шрифтів разом зі стилями і розмірами, а також дозволяє здійснювати попередній перегляд вибраного шрифту.
- Клас ColorDialog спрощує вибір необхідного кольору.

Оскільки для всіх класів діалогу базовим класом є ***CommonDialog***, то всі класи діалогу можуть використовуватися аналогічним чином.

Наведений нижче фрагмент коду демонструє приклад використання класу діалогу. Він дозволяє познайомитися із загальною концепцією використання діалогів:

- Спочатку створюється новий екземпляр класу діалогу.
- Потім необхідно присвоїти значення деяким властивостям, для того щоб дозволити/заборонити використання додаткових можливостей і визначити стан діалогу. В даному випадку властивості ***Title*** присвоюється значення ***«Sample»***, а прапорця ***ShowReadOnly – true***.
- Після цього викликається метод ***ShowDialog()***, що виводить діалогове вікно, яке переходить в режим очікування і реагує на те, що вводиться користувачем.
- У той момент, коли користувач натискає кнопку **ОК**, діалогове вікно закривається (перевірка на натиснуту кнопку **ОК** здійснюється шляхом порівняння результату діалогу з властивістю ***DialogResult.OK***).
- Після цього можна отримати доступ до значень, введених користувачем, звернувшись до значення певних властивостей. В

даному випадку ми зберігаємо значення властивості ***FileName*** в змінній ***fileName***:

```
string fileName;
OpenFileDialog dlg = newOpenFileDialog();
         dlg.Title = "Sample";
         dlg.ShowReadOnly = true;
if (dlg.ShowDialog() == DialogResult.OK)
fileName = dlg.FileName;
```

Клас ***OpenFileDialog*** дозволяє обирати ім'я файлу. Перед тим як викликати метод ***ShowDialog(),*** спочатку необхідно створити новий екземпляр класу ***OpenFileDialog*** (результат зображено на рис. Б.2):

```
OpenFileDialog dlg = new OpenFileDialog();
dlg.ShowDialog();
```

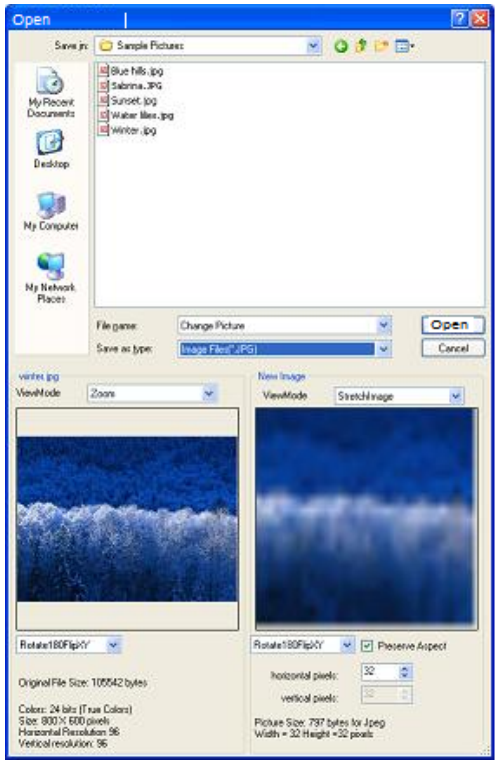

Рисунок Б.2 – Зовнішній вигляд діалогу OpenFileDialog

Параметри для роботи вікна можна задавати вручну, а можна вказувати у властивостях об'єкта ***OpenFileDialog*** (таблиця Б.3). Для цього його необхідно просто перетягнути на форму з панелі інструментів.

Фільтр для файлів визначає типи файлів, які користувач може вибирати для відкриття. Рядок простого фільтра для файлів виглядає наступним чином:

```
TextDocuments (* .txt) | * .txt | AllFiles | *. *
```

Цей фільтр використовується для виведення входжень у вікні зі списком ***Files of Type.*** Фільтр складається з декількох частин, розділених вертикальною лінією (|). Він може складатися тільки з парних рядків, тому кількість частин завжди має бути парною. Перший рядок визначає текст, який буде виводитися у вікні зі списком; другий рядок використовується для завдання розширень файлів, які будуть виводитися в процесі діалогу. Прогалини перед або після рядка фільтра є неприпустимими.

Таблиця Б.3 – Властивості класу ***OpenFileDialog*** [7]

Властивість	**Призначення**
Title	Дозволяє змінити заголовок діалогового вікна.
InitialDirectory	Дозволяє задати початкову директорію в діалозі. Для цього використовується функція ***GetFolderPath().***
Fiter	Фільтр для файлів визначає типи файлів, які користувач може обирати для відкриття.
FilterIndex	Визначає номер фільтра, що використовується за замовчуванням.
Multiselect	Якщо ***true***, то дозволяється виділення кількох файлів для відкриття.

Властивість ***FilterIndex*** визначає вибір за замовчуванням у вікні зі списком. Зверніть увагу на те, що ***FilterIndex*** відраховується з 1!

Класи ***SaveFileDialog*** і ***OpenFileDialog*** дуже схожі і мають багато однакових властивостей. Розглянемо властивості, притаманні тільки діалоговому вікну збереження файлів, і загальні для них властивості, використання яких відмінно. Розширення файлів використовуються для прив’язки файлів до додатків.

AddExtension – це властивість логічного типу, яка визначає, чи повинне розширення автоматично додаватися до імені файлу, що вводиться користувачем. Якщо користувач самостійно ввів розширення файлу, то в цьому випадку ніяких додаткових розширень додаватися не буде. [7]

DefaultExt – визначає розширення файлів, яке використовується в тих випадках, коли користувач не вказує розширення. Якщо ця властивість

залишається порожньою, то буде використовуватися розширення, яке визначається обраним на даний момент значенням властивості ***Filter***. [7]

Для перевірки допустимості імен файлів у вікні ***SaveFileDialog*** слід дотримуватися деяких додаткових перевірок і завдання значень деяких додаткових властивостей. По-перше, якщо властивості ***CreatePrompt*** присвоєно значення ***true***, буде виведений запит, чи повинен створюватися новий файл. Якщо властивості ***OverwritePrompt*** присвоєно значення ***true***, то це означає, що буде виводитися запит, чи дійсно користувач хоче здійснити запис на місце вже існуючого файлу. За замовчуванням властивості ***OverwritePrompt*** присвоюється значення ***true***, а властивості ***CreatePrompt*** – ***false***. При таких установках, якщо користувач спробує зберегти вже існуючий файл, буде виведено повідомлення, яке потребує підтвердження про заміну файлу. [7]

Діалогове вікно ***FontDialog*** дозволяє користувачеві додатки здійснювати вибір шрифту. Користувачеві надається можливість змінювати шрифт, стиль, розмір і колір шрифту [7].

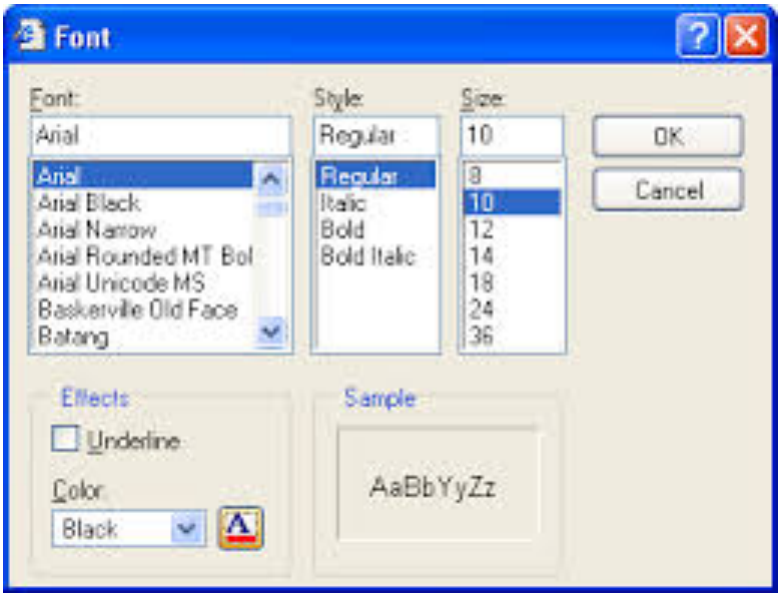

Рисунок Б.3 – Зовнішній вигляд діалога FontDialog

Призначення властивостей класу ***FontDialog*** наведені в таблиці Б.4:

Діалогове вікно ***ColorDialog*** дозволяє користувачеві створити свої власні кольори, якщо його не влаштовує жоден із запропонованих базових квітів; це досягається установкою властивості ***AllowFullColor.*** Частина

діалогового вікна, що відповідає за створення кольорів, може бути автоматично розширена за допомогою властивості ***Fullcolor.*** Властивість ***SolidColorOnly*** вказує на те, що користувачем можуть вибиратися тільки однорідні кольори. Властивість ***CustomColors*** може бути використано для зчитування і запису нових значень параметрів створюваного кольору.

Таблиця Б.4 – Властивості класу ***FontDialog*** [7]

Властивість	**Пояснення**
AllowVectorFonts	Логічне значення, яке визначає, чи можна обирати векторні шрифти.
AllowVerticalFonts	Логічне значення, яке визначає, чи можна обирати вертикальні шрифти.
FixedPitchOnly	в списку шрифтів будуть виводитися тільки шрифти з фіксованими розмірами. Значення за замовчуванням – ***false***.
MaxSize	Визначає максимальний розмір шрифту, що може обрати користувач.
MinSize	Визначає мінімальний розмір шрифту, що може обрати користувач.
ShowApply	Якщо необхідно вивести кнопку ***Apply,*** то властивості надається значення ***true***. Натискаючи кнопку ***Apply***, користувач отримує можливість побачити, як виглядає новий шрифт в додатку, не покидаючи діалог вибору шрифтів.
ShowColor	Коли ви бажаєте надати користувачеві можливість вибирати колір в діалозі вибору шрифту, то властивості ***ShowColor*** необхідно задати значення ***true.***
ShowEffects	Якщо ви бажаєте, щоб виводились на екран можливості , вибору елементів ***Strikeout*** (закреслення) і ***Underline*** (підкреслення) то встановлюють значення ***false***.
AllowScript Change	Завдання значення ***false*** забороняє зміну користувачем стилю обраного шрифту.

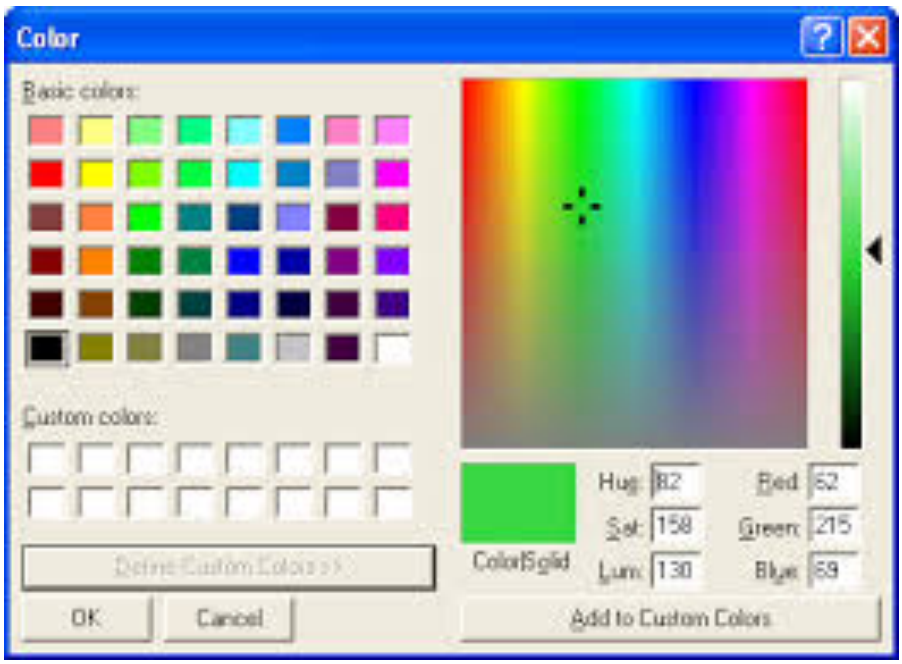

Рисунок Б.4 – Зовнішній вигляд діалогу ColorDialog

Всі властивості, які впливають на зовнішнє подання цього вікна, наведені в таблиці Б.5.

Таблиця Б.5 – Властивості діалогу ***ColorDialog*** [7]

Властивість	**Пояснення**
AllowFullopen	Задаючи значення ***false***, можна заборонити користувачу самостійно визначати кольори, відключаючи кнопку ***Define Custom Colors***. Значення за замовчуванням – ***true***.
FullOpen	Значення ***true*** означає, що перед виведенням діалогу автоматично відкриється діалогове вікно з можливістю створення власних кольорів.
AnyColor	Значення true дозволяє отримати всі допустимі кольори в списку основних кольорів.
CustomColors	За допомогою цієї властивості можна створювати масиви наперед визначених кольорів і вводити кольори, визначені користувачем.
SolidCoiorOnly	Коли ця властивість має значення ***true***, то користувач має можливість обрати лише однорідні кольори.

ДОДАТОК В

СТВОРЕННЯ МЕНЮ MENUSTRIP

Для створення меню в ***Windows Forms*** застосовується елемент ***MenuStrip***. Даний клас є похідними від ***ToolStrip*** і тому наслідує його функціональність [7].

Найбільш важливі властивості компонента ***MenuStrip***:

- ***Dock***: прикріплює меню до однієї зі сторін форми;
- ***LayoutStyle***: задає орієнтацію панелі меню на формі. Може, як і з ***ToolStrip***, приймати наступні значення:
  - ***HorizontalStackWithOverflow***: розміщується по горизонталі з переповненням – якщо довжина меню перевищує довжину контейнера, то нові елементи, що виходять за межі контейнера, не відображаються, тобто панель переповнюється елементами;
  - ***StackWithOverflow***: елементи розміщуються автоматично з переповненням;
  - ***VerticalStackWithOverflow***: елементи розміщуються вертикально з переповненням;
  - ***Flow***: елементи розміщуються автоматично, але без переповнення – якщо довжина панелі меню менша довжини контейнера, то елементи, що виходять за межі, переносяться;
  - ***Table***: елементи позиціонуються у вигляді таблиці;
- ***ShowItemToolTips***: вказує, чи будуть відображуватися випливаючі підказки для окремих елементів меню;
- ***Stretch***: дозволяє розтягти панель по всій довжині контейнера;
- ***TextDirection***: задає напрям тексту в пунктах меню.

MenuStrip виступає свого роду контейнером для окремих пунктів меню, що представлені об’єктом ***ToolStripMenuItem*** [7].

Додати нові елементи в меню можна в режимі конструктора (рис. В.1). Для додавання доступними є три види елементів: ***MenuItem*** (об’єкт ***ToolStripMenuItem***), ***ComboBox*** і ***TextBox***. Таким чином, ми можемо

використовувати в меню списки, що випадають, і текстові поля, але, як правило, ці елементи застосовуються переважно на панелі інструментів. Меню зазвичай містить набір об'єктів ***ToolStripMenuItem.***

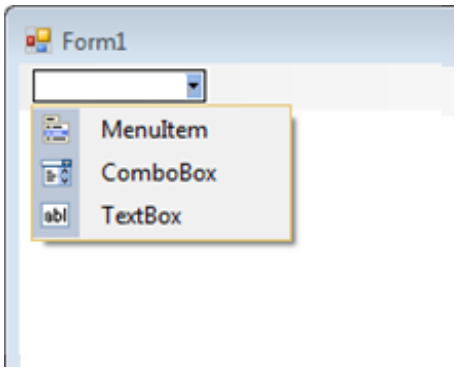

Рисунок В.1 – Режим конструктора для меню

Також ми можемо додати пункти меню в коді C#:

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
    ToolStripMenuItem file Item= new ToolStripMenuItem("Файл");
    fileItem.DropDownItems.Add("Створити");
    fileItem.DropDownItems.Add(new
ToolStripMenuItem("Зберегти"));
    menuStrip1.Items.Add(fileItem);
    ToolStripMenuItem aboutItem =
new ToolStripMenuItem ("Про програму");
    aboutItem.Click += aboutItem_Click;
    menuStrip1.Items.Add(aboutItem);
  }
void aboutItem_Click(object sender, EventArgs e)
 {     MessageBox.Show("Про програму"); }
}
```

Вигляд вікна додатка з програмно створеним меню зображено на рис. В.2.

ToolStripMenuItem в конструкторі приймає текстову мітку, яка буде використовуватися в якості тексту меню. Кожний такий об'єкт має колекцію

DropDownItems, яка зберігає дочірні об’єкти ***ToolStripMenuItem***. Тобто один елемент ***ToolStripMenuItem*** може містити набір інших об’єктів ***ToolStripMenuItem***. Таким чином створюється ієрархічне меню або структура в вигляді дерева [7].

Якщо передавати при додаванні рядок тексту, то для нього неявно буде створено об’єкт ***ToolStripMenuItem***:

```
fileItem.DropDownItems.Add("Створити").
```

Призначаючи обробники для події ***Click***, можна обробити натискання на пункти меню:

```
aboutItem.Click += aboutItem_Click().
```

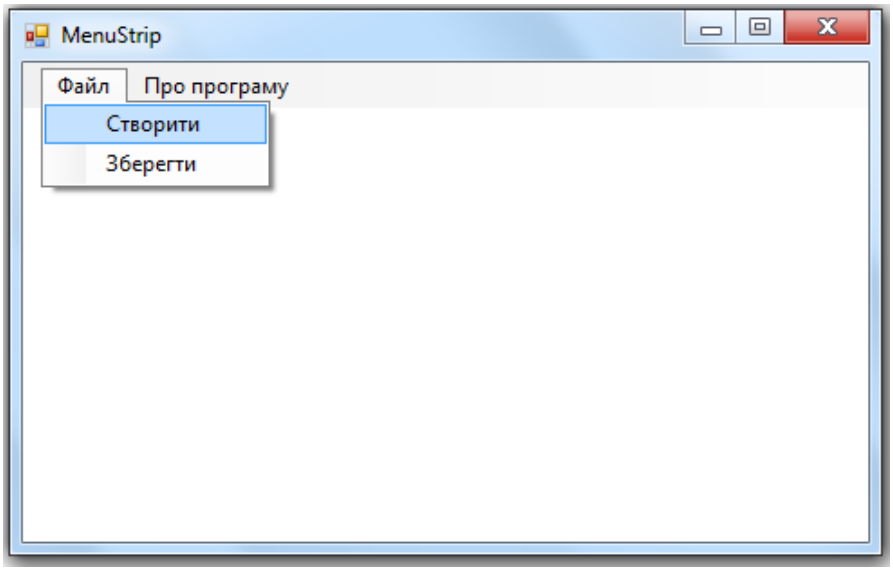

Рисунок В.2 – Додаток з меню

Позначки пунктів меню

Властивість ***CheckOnClick*** при значенні ***true*** дозволяє кликом позначити пункт меню. А за допомогою властивості ***Checked*** можна установити, чи буде пункт меню позначеним при запуску програми (рис. В.3).

Властивість ***CheckState*** повертає стан пункту меню – позначено його чи ні. Вона може приймати три значення: ***Checked*** (позначений), ***Unchecked*** (не позначений) і ***Indeterminate*** (стан невизначений).

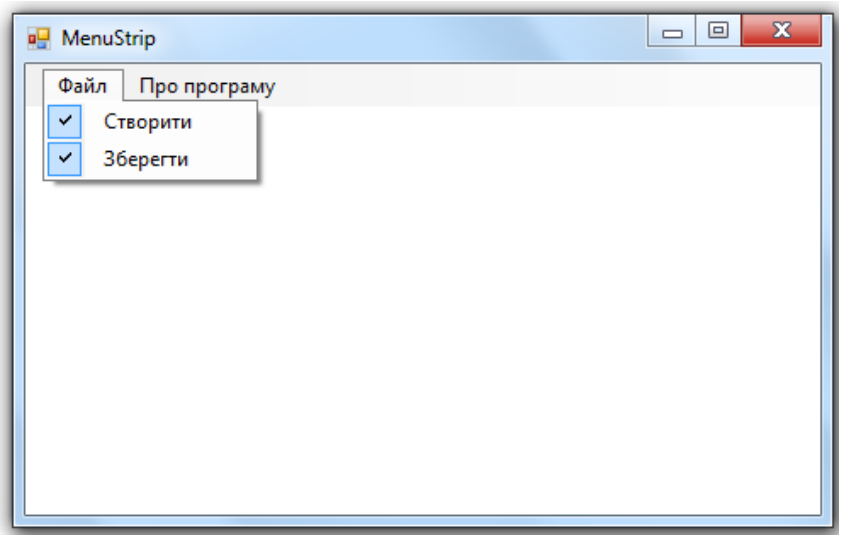

Рисунок В.3 – Меню з позначками

Наприклад, створимо ряд позначених пунктів меню і обробимо подію установки / зняття позначки:

```
public cpartial class Form1 : Form
{  public Form1()
  { InitializeComponent();
     ToolStripMenuItem fileItem =
                    new ToolStripMenuItem("Файл");
     ToolStripMenuItem          newItem          =          new
     ToolStripMenuItem("Створити")
     { Checked = true, CheckOnClick = true };
     fileItem.DropDownItems.Add(newItem);
     ToolStripMenuItem saveItem =
                         new ToolStripMenuItem("Зберегти")
     { Checked = true, CheckOnClick = true };
     saveItem.CheckedChanged += menuItem_CheckedChanged;
     fileItem.DropDownItems.Add(saveItem);
     menuStrip1.Items.Add(fileItem);
     }
```

```
voidmenuItem_CheckedChanged(object sender, EventArgs e)
{
      ToolStripMenuItem menuItem = sender as ToolStripMenuItem;
      if (menuItem.CheckState == CheckState.Checked)
      MessageBox.Show("Позначений");
      else if (menuItem.CheckState == CheckState.Unchecked)
            MessageBox.Show("Позначка знята");
}
}
```

Клавіші швидкого доступу

У випадку, коло необхідно швидко звернутися до якогось пункту меню, можна використовувати клавіші швидкого доступу. Для призначення клавіш швидкого доступу використовується властивість ***ShortcutKeys*** [7] :

```
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
      ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
ToolStripMenuItem saveItem = new ToolStripMenuItem("Зберегти")
           { Checked = true, CheckOnClick = true };
saveItem.Click+=saveItem_Click;
saveItem.ShortcutKeys = Keys.Control | Keys.P;
fileItem.DropDownItems.Add(saveItem);
menuStrip1.Items.Add(fileItem);
}
void saveItem_Click(object sender, EventArgs e)
{ MessageBox.Show("Збереження");}
}
```

Клавіші задаються за допомогою перерахування ***Keys***. В даному випадку натискання комбінації клавіш ***Ctrl + P*** приведе до обрання та натискання пункту меню ***«Зберегти»***.

За допомогою зображень можна урізноманітнити зовнішній вигляд пунктів меню. Для цього використовуються наступні властивості [7]:

– ***DisplayStyle***: визначає чи буде відображатися на елементі текст або зображення, або те і інше;
– ***Image****:* вказує на зображення;
– ***ImageAlign:*** встановлює вирівнювання зображення відносно елемента;
– ***ImageScaling:*** вказує, чи буде зображення розтягуватися, щоб заповнити весь простір елемента;
– ***ImageTransparentColor:*** вказує, чи буде колір зображення прозорим.

Якщо зображення для пункту меню обирається в режимі конструктора, то насамперед обираємо в вікні ***Властивість*** пункт ***Image***, після чого відкриється вікно для імпорту ресурсу зображення в проект (рис. В.4).

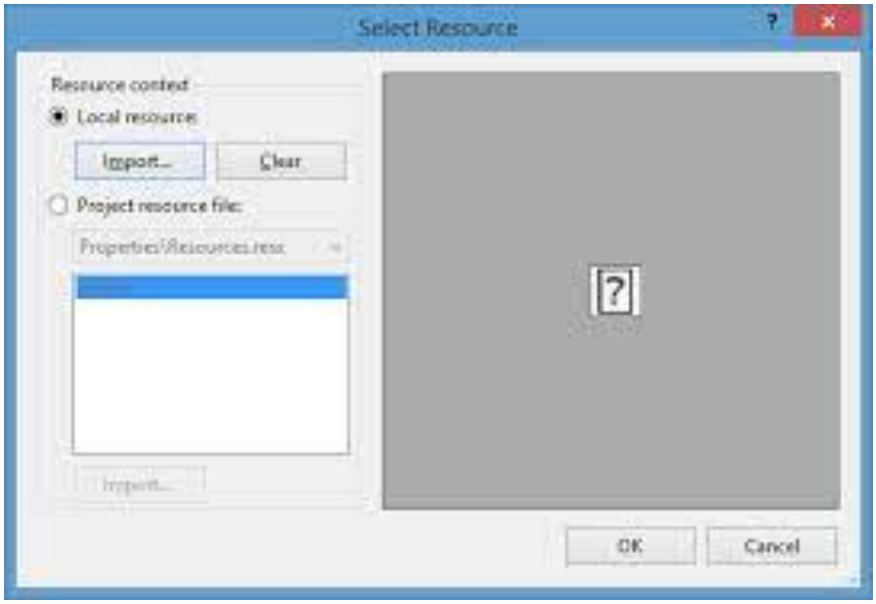

Рисунок В.4 – Меню з клавішами швидкого доступу в конструкторі

Щоб указати, як розмістити зображення, у властивості ***DisplayStyle*** треба встановити значення ***Image.*** Якщо ми хочемо, щоб кнопка відображала тільки текст, то треба обрати значення ***Text***, або можна комбінувати два значення за допомогою іншого значення ***ImageAndText***. За замовчуванням зображення розміщується зліва від тексту (рис. В.5).

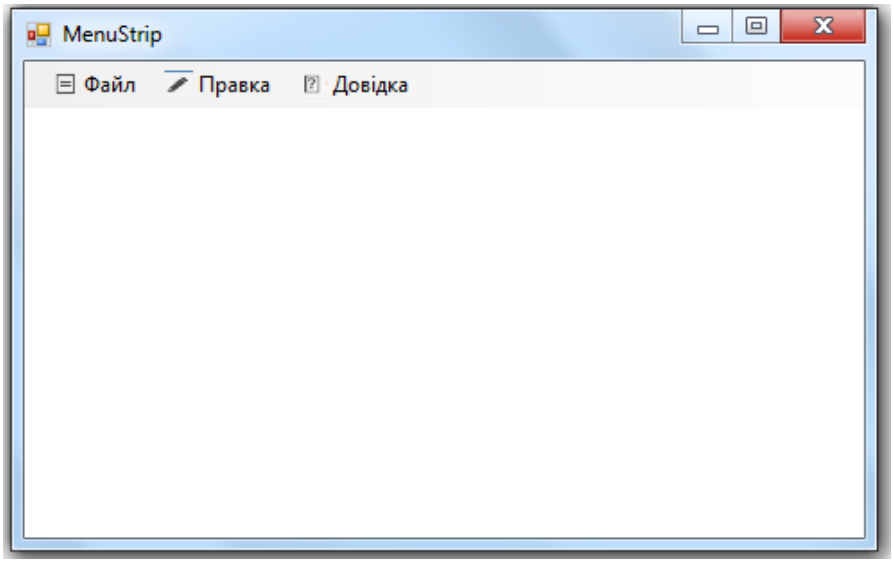

Рисунок В.5 – Меню з зображенням

Також можна установити зображення динамічно в коді програми:

```
fileToolStripMenuItem.Image =
        Image.FromFile(@"D:\Icons\0023\block32.png");
```

Рядок стану StatusStrip

StatusStrip є рядком стану, аналогічним панелі інструментів ***ToolStrip***. Рядок стану використовується для відображення поточної інформації про стан роботи додатка. [7]

При додаванні на форму ***StatusStrip*** автоматично розміщується в нижній частині вікна додатка. Але за необхідності його можна позиціонувати по-іншому, управляючи властивістю ***Dock***, що може приймати наступні значення [7]:

- ***Bottom:*** розміщується внизу (значення за замовчуванням);
- ***Top:*** прикріплює статусний рядок до верхньої частини форми;
- ***Fill:*** розтягає на всю форму;
- ***Left:*** розміщується в лівій частині форми;
- ***Right:*** розміщується в правій частині форми;
- ***None:*** довільне положення.

StatusStrip може містити різноманітні елементи. В режимі конструктора можна додати наступні типи елементів:

– ***StatusLabel:*** метка для виведення текстової інформації. Являється об’єктом ***ToolStripLabel***;
– ***ProgressBar:*** індикатор прогресу. Являється об’єктом ***ToolStripProgressBar***;
– ***DropDownButton:*** кнопка с випадаючим списком по кліку. Являється об’єктом ***ToolStripDropDownButton***;
– ***SplitButton:*** кнопка, багато в чому аналогічна ***DropDownButton***. Являється об’єктом ***ToolStripSplitButton.***

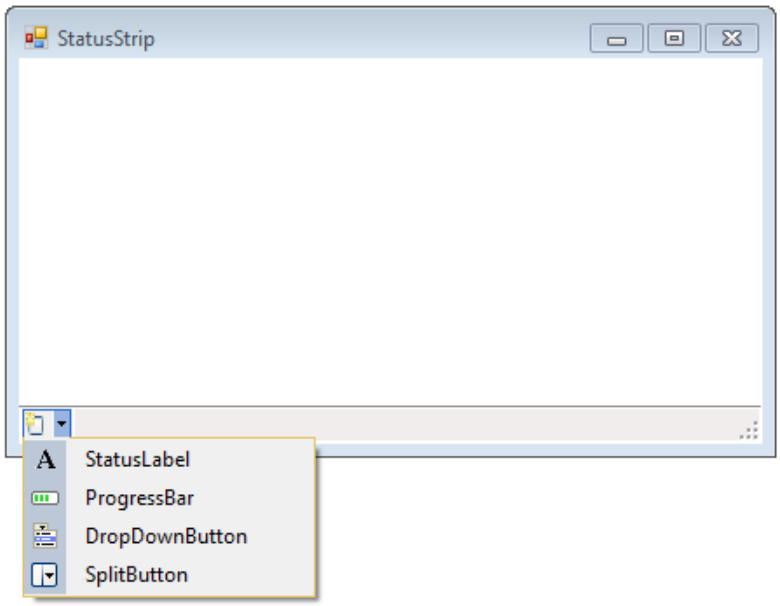

Рисунок В.6 – Елементи статусного рядка (рядка стану).

Також можна звернутися на панелі властивостей до властивості ***Items*** компонента ***StatusStrip*** і у вікні, що відкриється, додати і налаштувати всі елементи (рис. В.7).

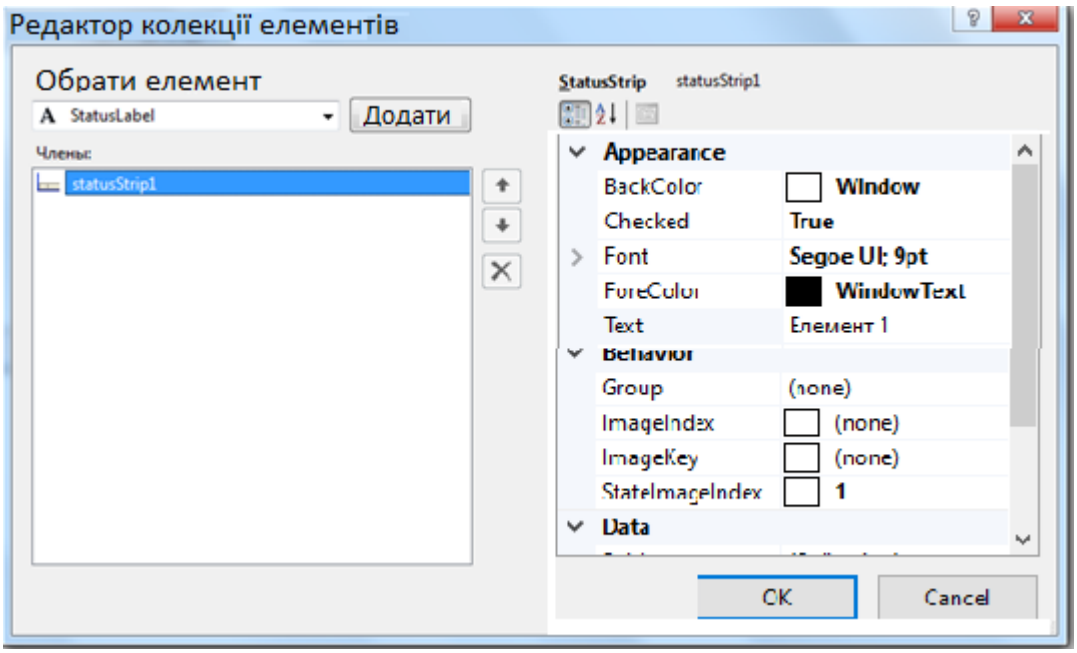

Рисунок В.7 – Редактор колекції елементів

Також можна додати елементи програмно. Наприклад:

```
public partial class Form1 : Form
{
    ToolStripLabel dateLabel;
    ToolStripLabel timeLabel;
    ToolStripLabel infoLabel;
    Timer timer;
    public Form1()
    {
        InitializeComponent();
        infoLabel = new ToolStripLabel();
        infoLabel.Text = "Поточні дата і час:";
        dateLabel = new ToolStripLabel();
        timeLabel = new ToolStripLabel();
        statusStrip1.Items.Add(infoLabel);
        statusStrip1.Items.Add(dateLabel);
        statusStrip1.Items.Add(timeLabel);
        timer = new Timer() { Interval = 1000 };
        timer.Tick += timer_Tick;
        timer.Start();
    }
```

```
 void timer_Tick(object sender, EventArgs e)
 {
    dateLabel.Text = DateTime.Now.ToLongDateString();
   timeLabel.Text= DateTime.Now.ToLongTimeString();
 }
}
```

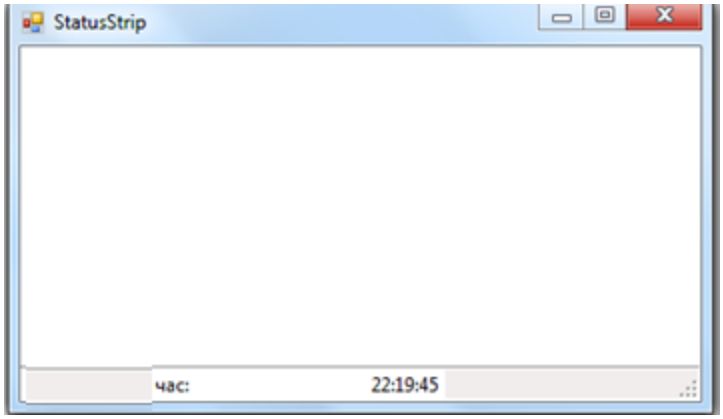

Рисунок В.8 – Вікно додатка з рядком стану

В програмі створюються три мітки в рядку стану і таймер. Після створення форми таймер запускається, і відбувається його подія ***Tick***, в обробнику якої встановлюємо текст міток.

Контекстне меню ContextMenuStrip

ContextMenuStrip являє собою контекстне меню. Даний компонент дуже схожий до елементу ***MenuStrip*** за виключенням того, що контекстне меню не може використовуватися саме по собі, воно обов'язково застосовується до якогось іншого елементу, наприклад, текстового поля [7].

Нові елементи в контекстне меню можна додавати в режимі конструктора (рис. В.9).

При цьому ми можемо додавати ті ж самі елементи, що і в ***MenuStrip***. Але, як правило, використовується ***ToolStripMenuItem***, або елемент ***ToolStripSeparator***, який є горизонтальною смугою-розмежувачем між іншими пунктами меню.

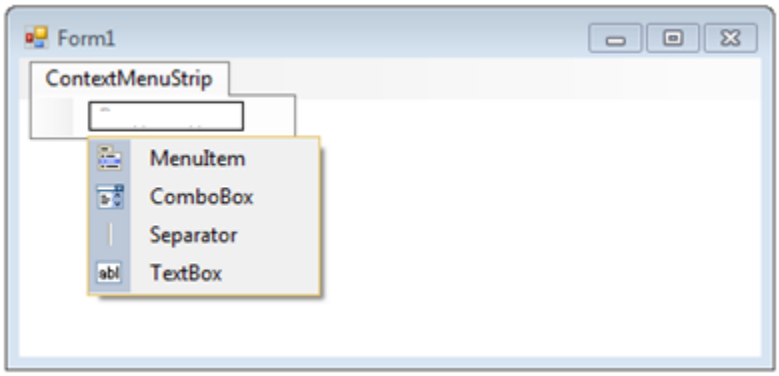

Рисунок В.9 – Вікно конструктора контекстного меню

Також створити контекстне меню через панель властивостей звертаючись до властивості ***Items*** компонента ***ContextMenuStrip*** і відповідному вікні налаштувати всі елементи меню (рис. В.10).

І третя можливість створення контекстного меню – динамічне контекстне меню програмними засобами. Додамо на форму елементи ***ContextMenuStrip*** і ***TextBox***, з іменами ***contextMenuStrip1*** і ***textBox1*** відповідно.

Потім внесемо зміни в програмний код:

```
public partial class Form1 : Form
{
  string buffer;
  public Form1()
  {
    InitializeComponent();
    textBox1.Multiline = true;
    textBox1.Dock = DockStyle.Fill;
// створюємо елементи меню
ToolStripMenuItem copyMenuItem =
                  new ToolStripMenuItem ("Копіювати");
ToolStripMenuItem pasteMenuItem =
                  new ToolStripMenuItem("Вставити");
// додаємо елементи в меню
contextMenuStrip1.Items.AddRange
              (new [] { copyMenuItem, pasteMenuItem });
```

```
// асоціюємо контекстне меню с текстовим полем
textBox1.ContextMenuStrip = contextMenuStrip1;
// встановлюємо обробники подій для меню
copyMenuItem.Click += copyMenuItem_Click;
pasteMenuItem.Click += pasteMenuItem_Click;
}
// вставка тексту
void pasteMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Paste(buffer);
}
// копіювання тексту
void copyMenuItem_Click(objectsender, EventArgs e)
 {
// якщо виділено текст в текстовому полі,
// то копіюємо його в буфер
   buffer = textBox1.SelectedText;
 }
}
```

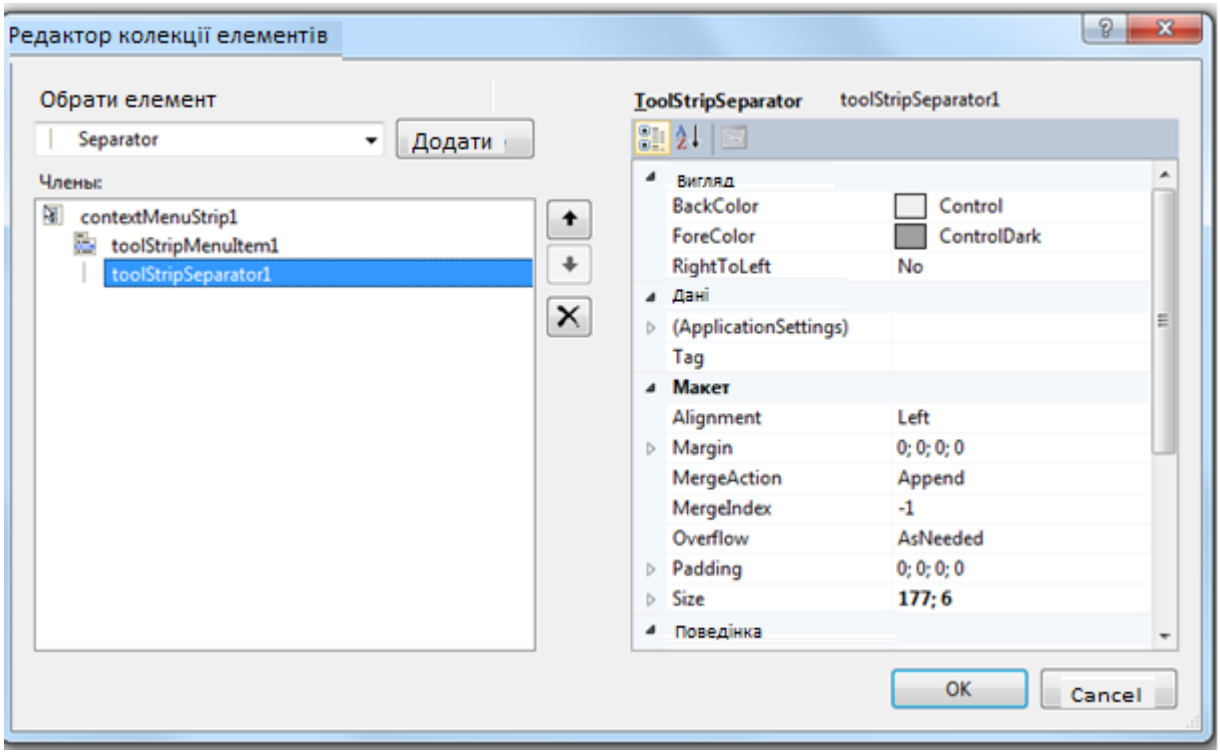

Рисунок В.10 – Вікно налаштування контекстного меню через властивість

В даному випадку виконана найпростіша реалізація функціональності ***copy-paste***. В меню додається два елемента. А у текстового поля встановлюється багаторядковість, і воно розтягується за шириною контейнера.

Багато компонентів мають властивість ***ContextMenuStrip***, що дозволяє асоціювати контекстне меню з даним елементом. У випадку з ***TextBox*** асоціація виконується наступним чином:

```
textBox1.ContextMenuStrip = contextMenuStrip1.
```

По натисканню на текстове поле правою кнопкою мишки ми зможемо викликати асоційоване контекстне меню.

За допомогою обробників натискання пунктів меню встановлюються дії з копіювання і вставки рядків (рис. В.11).

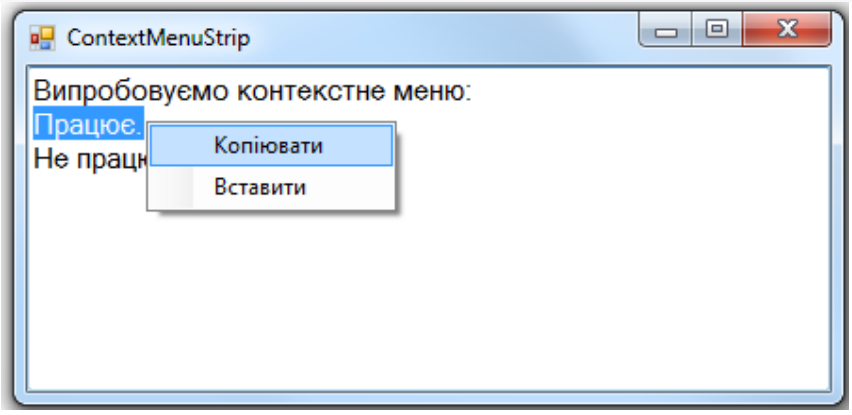

Рисунок В.11 – Приклад роботи контекстного меню

ДОДАТОК Г
ПРИКЛАД СТВОРЕННЯ БАГАТОВІКОННОГО ДОДАТКУ

Г.1 Інтерфейс першого вікна

Створимо перше вікно (рис. Г.1). Є три надписи. Надпис ***«Состояние»*** буде змінюватися по ходу виконання програми. В об'єкті ***pictureBox1*** постійно розміщується фото автора програми. Фото потрібно обрати відразу візуально. Об'єкт ***pictureBox2*** службовий: в ньому буде відображатися фото друга, коли прийде момент його обирати.

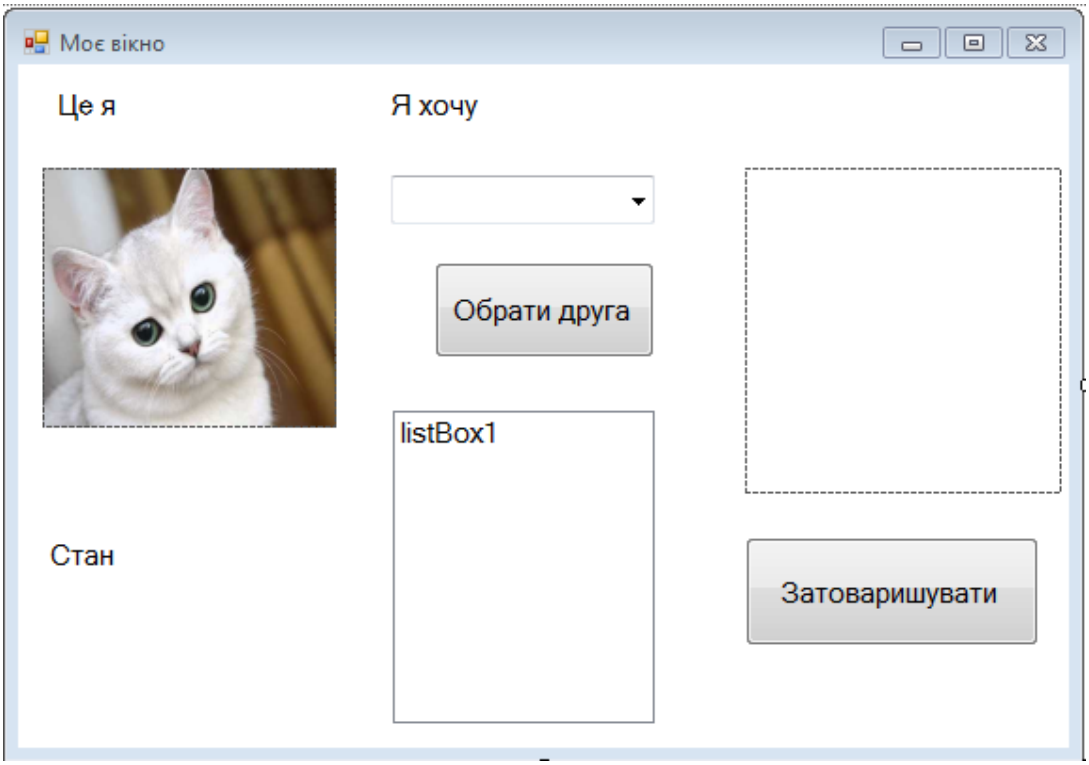

Рисунок Г.1 – Перше вікно

В об'єкті ***listBox1*** буде відображатися список файлів с різними фотографіями потенціальних друзів. Будемо вважати, що у нас є дві теки з фотографіями: ***Фото_1*** (це автори програми) і ***Фото_2*** (друзі). Для визначеності: теки розміщені за адресою ***D:\USERS.***

В об'єкті ***comboBox1*** буде список побажань ***автора*** програми. Спочатку він пустий, але буде поповнюватися в процесі виконання програми. Є дві кнопки. Одна запускає процес вибору друга, інша фіксує вибір.

Сценарій

Під час запуску програми об'єкти ***listBox1***, ***pictureBox2*** і обидві кнопки не видимі. При виборі (або наборі) будь-якого бажання воно заноситься в список (якщо його там немає), а також відображається в надписі ***«Стан»***. Якщо обране (або набране) бажання ***«Запросити друга»***, то стає видимою кнопка ***«ОБРАТИ ДРУГА»***. При натисканні на неї стає видимою кнопка ***«ЗАТОВАРИШУВАТИ»*** і відкривається список ***listBox1***, в якому відображається вміст теки ***Фото_2*** (список імен файлів). Обраний файл відображається в ***pictureBox2***.

При натисканні кнопки ***«ЗАТОВАРИШУВАТИ»*** вибір вважається зробленим і відображається друге вікно (про нього – пізніше).

Завдання для самостійного виконання

1. Створіть інтерфейс першого вікна. Забезпечте при запуску видимість об'єктів у відповідності до сценарію.

2. Візуально включіть в список ***ComboBox1*** *одне бажання:* ***«Подивитися фільм»****.*

3. Створіть для об'єкта ***comboBox1*** *обробник події* ***SelectedIndexChanged****. Включіть в нього оператор, формуючий замість надпису* ***«Стан»*** *текст:* ***«Виконую бажання: xxxxxx»****, де* ***xxxxxx*** *– це текст обраного (набраного) бажання.*

4. Перевірте роботу програми. При виборі бажання стан змінюється. А ось при наборі – ні. Забезпечте обробку набору, використовуючи подію ***KeyPress*** *для об'єкта* ***comboBox1****. У відповідності до сценарію повинно змінюватися стан, а набране бажання повинне додаватися в список об'єкта* ***comboBox1****, якщо його там ще немає.*

5. При виборі (наборі) бажання ***«Запросити друга»*** *зробіть видимою кнопку* ***«ОБРАТИ ДРУГА»****. При виборі (наборі) будь-якого іншого бажання кнопку потрібно приховати.*

6. При кожному новому запуску програми зі списку бажань пропадають бажання, накопичені в минулому сеансі. Збережіть список бажань в окремому файлі ***«Бажання»*** *при закритті вікна. Під час запуску програми відновіть список бажань з файлу.*

Вказівка. Скориставшись подіями ***Load*** і ***FormClosing***. Для організації файлу використовуйте ***StreamReader*** і ***StreamWriter***. Розмістіть його в тій же теці, де знаходиться готова програма (тека ***Debug***).[7]

Тепер оживимо кнопку ***«ОБРАТИ ДРУГА»***. Створимо обробник і вставимо всередину код (скопіюйте, але постарайтесь запам'ятати код):

```
// FolderBrowserDialog - вікно Обзор папок
FolderBrowserDialogfb = new FolderBrowserDialog();
// заголовок вікна
fb.Description = "Виберіть теку";
// в вікні не буде дозволено створювати  нові теки
fb.ShowNewFolderButton = false;
// відображаємо діалогове вікно
if (fb.ShowDialog() == DialogResult.OK)
{ // користувач обрав каталог і клацнув по кнопці OK
papka = fb.SelectedPath;
// інформація про каталог
DirectoryInfo di = new DirectoryInfo(papka);
// інформація про файли
FileInfo[] fi = di.GetFiles("*.jpg");
// очистіть список listBox
listBox1.Items.Clear();
// додаємо в listBox1 імена файлів, що містяться
// в каталозі papka
foreach (FileInfo fc in fi)
{ listBox1.Items.Add(fc.Name); }
if (fi.Length > 0) // обираємо перший файл зі списку
listBox1.SelectedIndex = 0;
pictureBox2.Visible = true; // зрозуміло для чого?
listBox1 .Visible = true; button2.Visible = true;
}
```

Код організує діалогове вікно вибору теки. Користувач зможе знайти і обрати теку ***Фото_2***, де зберігаються фотографії друзів. Після підтвердження вибору (***ОК***) в список об'єкта ***listBox1*** запишуться імена файлів із теки. Штучно підсвічується ім'я першого файлу списку. Відображаються список,

друга кнопка і картинка ***pictureBox2***. Але, картинки в ній поки не буде, потому що властивість ***Image*** ще не містить посилання.

Зробимо ще одну вставку. Клацнемо двічі по об'єкту ***listBox1***. Відкриється обробник події ***SelectedIndexChanged*** [7]. Додамо в нього текст:

```
// режим відображення картинки в повне вікно
pictureBox2.SizeMode = PictureBoxSizeMode.Zoom;
// загружаємо посилання на зображення в pictureBox2
pictureBox2.Image = new Bitmap(papka + " \\"
 +listBox1.SelectedItem.ToString());
```

При виборі імені файлу в спису відповідне зображення тепер повинне з'явитися в ***pictureBox2***.

Спробуйте запустити програму. Виявиться три помилки, що виникли з однієї і тої ж причини: не оголошена змінна ***papka***. За сенсом вставленого коду ця змінна містить рядкове значення – адресу теки з фотографіями друзів. Потрібно її оголосити.

Завдання для самостійного виконання

*Оголосіть змінну **papka** так, щоб вона була доступна одразу двом обробникам. Запустіть програму і переконайтесь, що тека обирається, і обраний файл відображається у вигляді картинки в **pictureBox2**.*

Але, зафіксувати вибір кнопкою «***ЗАТОВАРИШУВАТИ***» поки неможливо. Необхідно ще одне вікно – вікно друга.

Г.2 Інтерфейс другого вікна

Використовуючи головне меню, додамо в програму нову форму ***Form2***: ***Проект – Добавить форму Windows***. Це буде вікно друга (рис. Г.2).

Надпис ***«Пропозиція»*** буде містити відомості про те, що робить той, хто запросив – бажання автора програми.

Надпис ***«Моя думка»*** повинен містити реакцію друга на бажання автора. Об'єкт ***pictureBox1*** буде містити фото друга, якого обрано в першому вікні. Кнопка має той же сенс, що і в вікні ***Form1***.

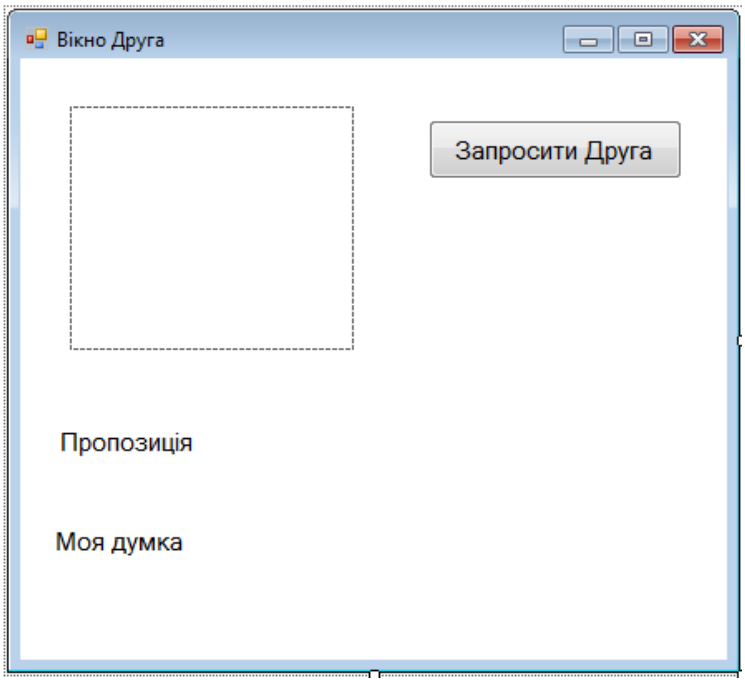

Рисунок Г.2 – Друге вікно.

Сценарій

Коли користувач в першому вікні обере фото друга, він натискає кнопку ***«ЗАТОВАРИШУВАТИ»***. Повинне відобразитися друге вікно з фотографією друга.

Підключення другого вікна

Вікна ***Form1*** і ***Form2*** обслуговуються різними класами. Об'єкти класів (і вікон) доступні лише всередині своїх класів. При запуску другого вікна доведеться картинку (посилання на неї) записувати у властивість ***Image*** об'єкта ***pictuteBox1***. Оператор буде знаходитися в обробнику кнопки ***«ЗАТОВАРИШУВАТИ»***, а це ***Form1***.

Завдання для самостійного виконання

*1. В класі **Form2** для забезпечення доступу до його поля **pictureBox1** створіть відкриту властивість, що повертає посилання на **pictureBox1**. Можна назвати цю властивість, наприклад, так: **ПосиланняНаФото**.*

*2. В обробник кнопки **«ЗАТОВАРИШУВАТИ»** додайте код, що реалізує наступний алгоритм:*

*Якщо в **pictureBox2** відображено фото (в **Image** є ненульове посилання на об'єкт), то:*

– створюється об’єкт **F2** класу **Form2** (змінну **F2** потрібно оголосити як поле класу **Form1**);
– у об’єкта **F2** в **pictureBox1** властивості **SizeMode** привласнюється значення **Zoom** (повне зображення в об’єкті) з перерахування **PictureBoxSizeMode** (доступ до **pictureBox1** – через властивість, створену в пункті 1);
– у об’єкта **F2** в **pictureBox1** посилання на картинку **Image** отримує те ж значення, що і в **pictureBox2** у вікні **Form1**;
– об’єкти **listBox1**, **pictureBox2** і обидві кнопки робляться невидимими (після вибору фото друга їх можна приховати);
– текст в полі набору в об’єкті **comboBox1** видаляється (формується пустий рядок);
– створений об’єкт F2 показується на екрані.

Після виконання завдань запустіть програму і оберіть друга. Повинні отримати приблизно таке (рис. Г.3). Тепер, коли є можливість «відкрити» друга, доробимо перше вікно.

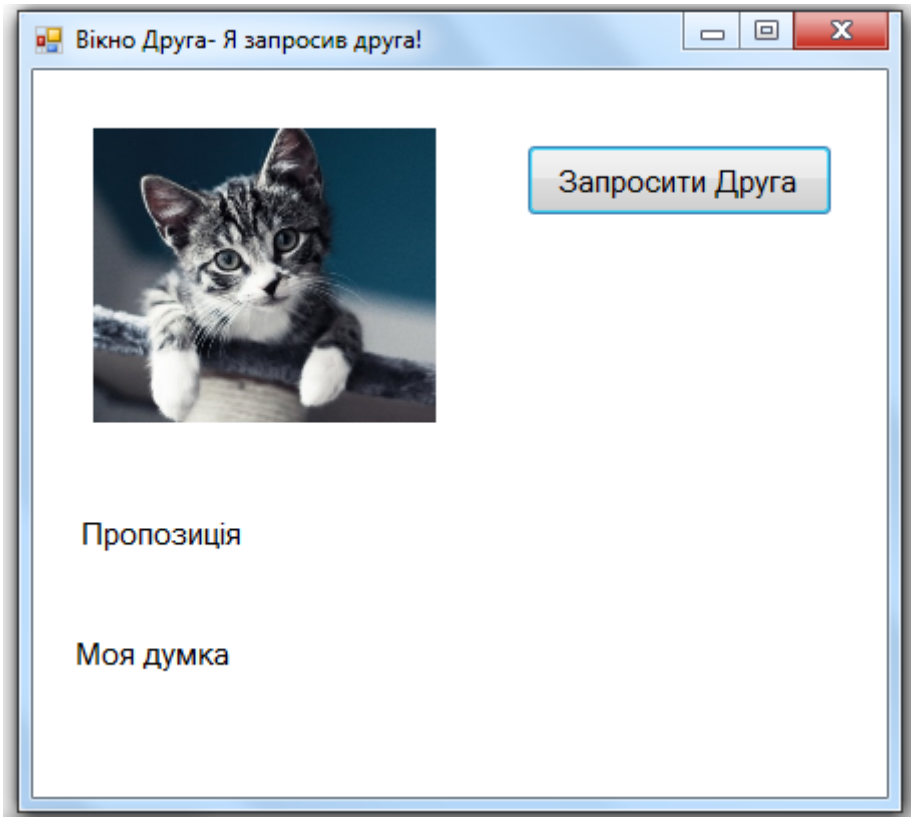

Рисунок Г.3 – Вікно друга

Завдання для самостійного виконання

1. Оскільки є друг, то в заголовку першого вікна потрібно це відобразити. Нехай вікно автора тепер називається так: Моє вікно – я запросив друга!

*2. Зараз при повторному виборі бажання **«Запросити друга»** знову відкривається кнопка **«ОБРАТИ ДРУГА»**. Забороніть повторне запрошення. Якщо відкрите вікно **Form2** (об'єкт-форма має ненульове посилання!), то потрібно просто поспівчувати собі – дати, наприклад в рядку стану повідомлення: **«на двох, нажаль, не досить грошей…»***

3. При закритті вікна друга відновіть попередні режими першого вікна. Для цього:

- створіть в класі **Form1** обробник закриття другої форми;
- підпишіть цей обробник на подію **Form_Closed** другого вікна – краще за все одразу після відкриття **F2**;
- в обробнику у **F2** обнуліть посилання на друге вікно, відновіть попередній заголовок першого вікна і прокоментуйте в стані факт прощання з другом: **«Щасливо бути!»**.

Друге вікно як спостерігач

Коли обидва вікна відкриті, користувач може продовжувати працювати з першим вікном. Наприклад, виконувати якісь інші бажання (підключати ще одного друга заборонено). Бажання, що виконуються, фіксуються в стані першого вікна.

Сценарій спостерігача

Друг (друге вікно) може виступити в ролі ***спостерігача*** за діями ***автора*** (перше вікно) і висловлювати своє відношення. Для цього йому необхідно отримати інформацію про те, яке саме бажання виконується. ***Друг*** повинен:

- піймати момент вибору чергового бажання;
- мати можливість прочитати бажання, обране в ***comboBox1*** першого вікна;
- відобразити це бажання у себе в надпису ***«Пропозиція»***;
- вивести в надпис ***«Моя думка»*** своє відношення до цього.

Щоб це стало можливим, потрібно виконати ряд робіт.

Завдання для самостійного виконання

*1. Оголосіть в класі **Form2** відкрите поле **F1** класу **Form1**. При показі другого вікна запишіть в це поле посилання на **Form1**.*

*2. Створіть в **Form1** відкриту властивість, що повертає посилання на **comboBox1**. Ім'я властивості: **ПосиланняНаСписокБажань**.*

*3. Створіть у друга метод-обробник події **Вибір_бажання** в першому вікні. Підпишіть його на цю подію. Підписку краще за все організувати при завантаженні вікна **Form2**.*

*4. В обробнику відобразіть текст бажання автора в об'єкті **label1**, а відношення друга до цієї події – в **label2**.*

5. Дайте можливість другу запросити свого друга – у нього для цього є кнопка. Нехай друг викликає свого друга, а той спостерігає відразу за двома товаришами: за бажанням першого і реакцією другого. Свою думку він також повинен висловити в якомусь надпису. Алгоритм на ваш розсуд.

ДОДАТОК Д
ПРИКЛАД РОЗРОБКИ ДОДАТКІВ З ВИКОРИСТАННЯМ ГРАФІКИ ТА АНІМАЦІЇ

Д.1 Робота з графікою

Створимо новий проект і розташуємо об’єкти ***Button*** та ***PictureBox*** на формі (рис. Д.1). Налаштуємо об’єкт ***PictureBox***, наприклад: на вкладці ***«Макет – Size»*** можемо вказати точні розміри об’єкта в пікселях. Для кнопки створимо обробник події ***button1_Click().***

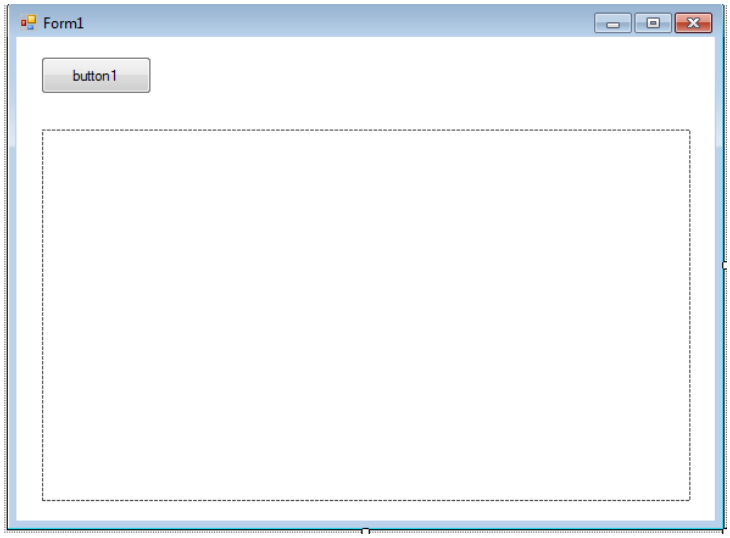

Рисунок Д.1 – Вікно задачі

В мові C# існує бібліотека для створення графіки ***«System.Drawing».*** Вона автоматично підключається при створенні проекту.[7] Наприклад, можна нарисувати прямокутник наступним чином (результат зображено на рис. Д.2):

```
private void button1_Click(object sender, EventArgs e)
{
//Обираємо перо "myPen" чорного кольору Black
//товщиною в 1 піксель:
Pen myPen = new Pen(Color.Black, 1);
//Оголошуємо об’єкт "g" класу Graphics і надаємо
//йому можливість зображення на pictureBox1:
Graphics g = Graphics.FromHwnd(pictureBox1.Handle);

//Рисуємо прямокутник:
g.DrawRectangle(myPen, 10,10,50,50);
}
```

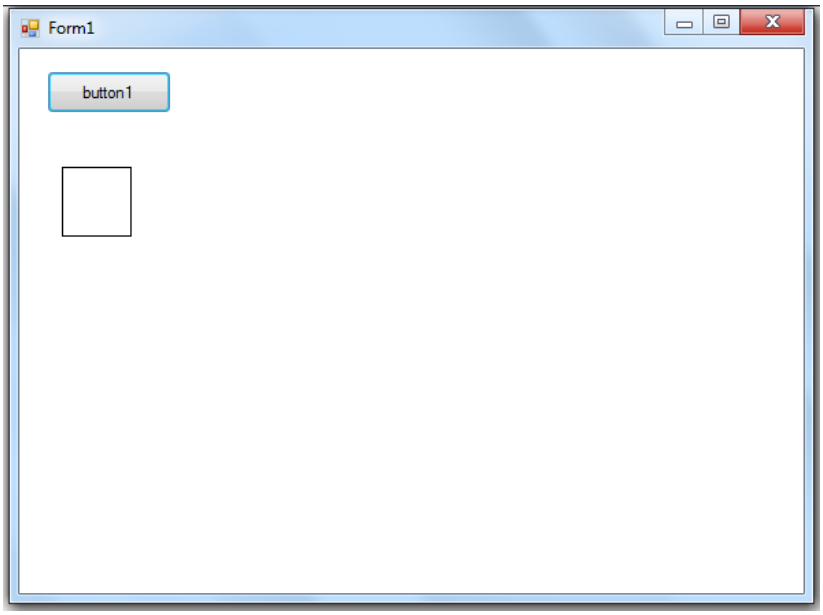

Рисунок Д.2 – Прямокутник в зоні об’єкта ***«pictureBox1»***

Завдання для самостійного виконанн

Ознайомтесь з основними графічними примітивами та напишіть програму, яка демонструє роботу з ними.

Д.2 Створення анімації в C#

Створюємо проект, розміщуємо на формі об’єкти: ***PictureBox***, ***Timer*** і ***Button*** [7].

Спробуємо створити примітивну анімацію за допомогою ***PictureBox*** і ***Timer***, яка запуститься після натискання на кнопку. Для цього будемо обробляти подію натискання на кнопку і подію «відпрацювання» таймера. Також оголосимо всі необхідні для рисування об'єкти і змінні.

```
//оголошуємо об'єкт - графіку, на якому будемо рисувати
Graphics gr;
// оголошуємо об'єкт-олівець, яким будемо рисувати контур
Pen p;
// оголошуємо об'єкт-заливки для заливки фону
SolidBrush fon;
//і середини Фігури
SolidBrush fig;
int rad;      // змінна для збереження радіуса кругів
Random rand;  // об'єкт для отримання випадкових чисел
```

Оголошуємо функцію, яка буде рисувати круг за координатами його центру:

```
void DrawCircle(int x, int y)
{
   int xc, yc;
   xc = x - rad;
   yc = y - rad;
   gr.FillEllipse(fig, xc, yc, rad, rad);
   gr.DrawEllipse(p, xc, yc, rad, rad);
}
```

Створимо обробник ***button1_Click()*** через подвійний клік по кнопці та додамо до нього наступний код:

```
private void button1_Click(object sender, EventArgs e)
{
// ініціалізуємо об’єкт типу графіка,
// прив’язуючи  до PictureBox
    gr = pictureBox1.CreateGraphics();
    p = new Pen(Color.Lime); // задали колір для олівця
    fon = new SolidBrush(Color.Black); // і для заливки
    fig = new SolidBrush(Color.Purple);
    rad = 40;                  //задали радіус для круга
// ініціалізуємо об’єкт для випадкових чисел
    rand = new Random();
// зафарбуємо чорним нашу область рисування
    gr.FillRectangle
      (fon,       0,       0,       pictureBox1.Width,
pictureBox1.Height);
// викликаємо  функцію, для прорисовки круга
// випадковим чином обравши перед цим координати центра
    int x,  y;
    for (int i = 0; i < 15; i++)
    {
      x = rand.Next(pictureBox1.Width);
      y = rand.Next(pictureBox1.Height);
      DrawCircle(x, y);
     }
// включимо в роботу наш таймер, тобто тепер
// буде відбуватися подія Tick
// і її буде обробляти функція On_Tick
     timer1.Enabled = true;
}
```

Щоб мати змогу використовувати таймер, перейдемо до конструктора форми і виконаємо подвійний клік по таймеру, добавленному на форму та додамо в нього наступний код:

```
private void timer1_Tick(object sender, EventArgs e)
{
// спочатку очищуємо область рисування кольором фону
   gr.FillRectangle(fon,   0,   0,   pictureBox1.Width,
pictureBox1.Height);
// потім знову випадковим чином обираємо координати
// центрів кругів і рисуємо їх
   int x, y;
   for (int i = 0; i < 15; i++)
   {
     x = rand.Next(pictureBox1.Width);
     y = rand.Next(pictureBox1.Height);
     DrawCircle(x, y);
    }
 }
```

Тепер можна запустити програму і після натискання на кнопку побачимо просту анімацію – випадкове переміщення кругів по блідо-блакитному фону, кадр якої зображено на рис Д.3.

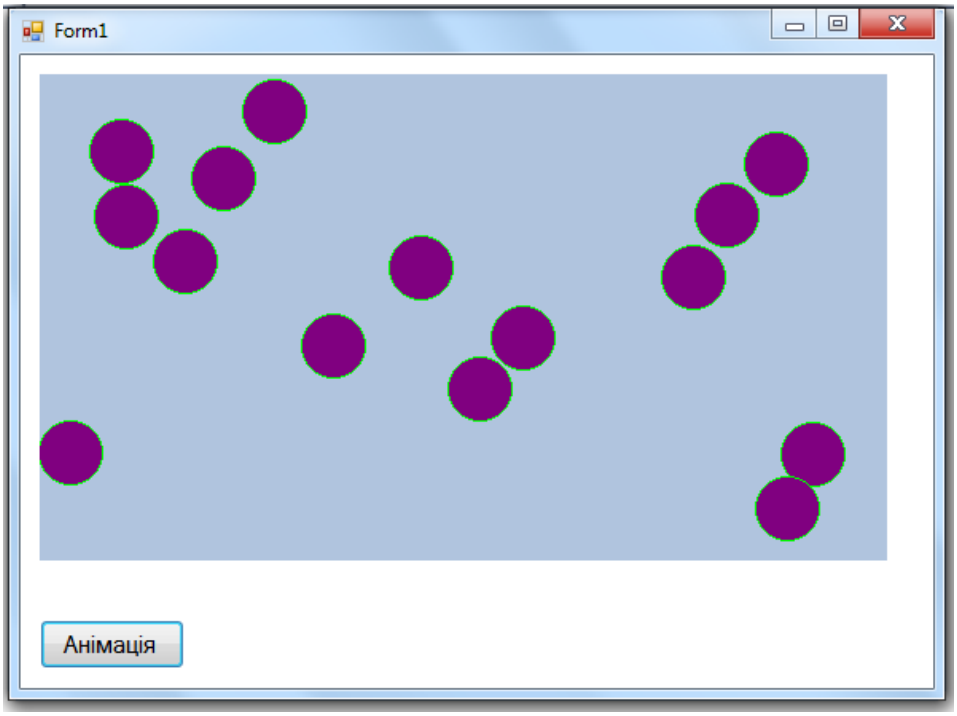

Рисунок Д.3 – Проста анімація

Для того, щоб анімація відповідала вимогам іноді необхідно змінювати так званий тік таймера, тобто проміжок виконання наступного кроку анімації. Це виконується і ***Інспекторі об'єктів*** [7].

Для цього оберемо елемент ***Timer***, перейдемо до його ***Властивостей*** і там оберемо і змінимо параметр ***Interval*** (вимірюється в мілісекундах). В нашому випадку ***Interval*** дорівнює 150 мс.

Завдання для самостійного виконання.

Додайте в програму кілька графічних примітивів різного кольору та розмірів.

Д.3 Візуалізація графіка функції

Умова завдання

Розглянемо процес розробки програми, задачею якої буде візуалізація графіка заданої функції. В програмі буде демонструватися процес зміни значення функції на графіку. Це можна реалізувати наступним чином: по графіку рухається червона точка, з координатами **(x, y)**, біля курсору будуть зображуватися її координати (рис. Д.4).

Створіть основу додатка розмістивши на формі елемент ***SimpleOpenGLControl*** [7]. Додайте в проект таймер під ім'ям ***PointInGrap*** і установіть в його властивостях інтервал **30** мілісекунд. Потім подвійним кліком по ньому створіть функцію ***PointInGrap_Tick*** – обробник події ***ontimer***.

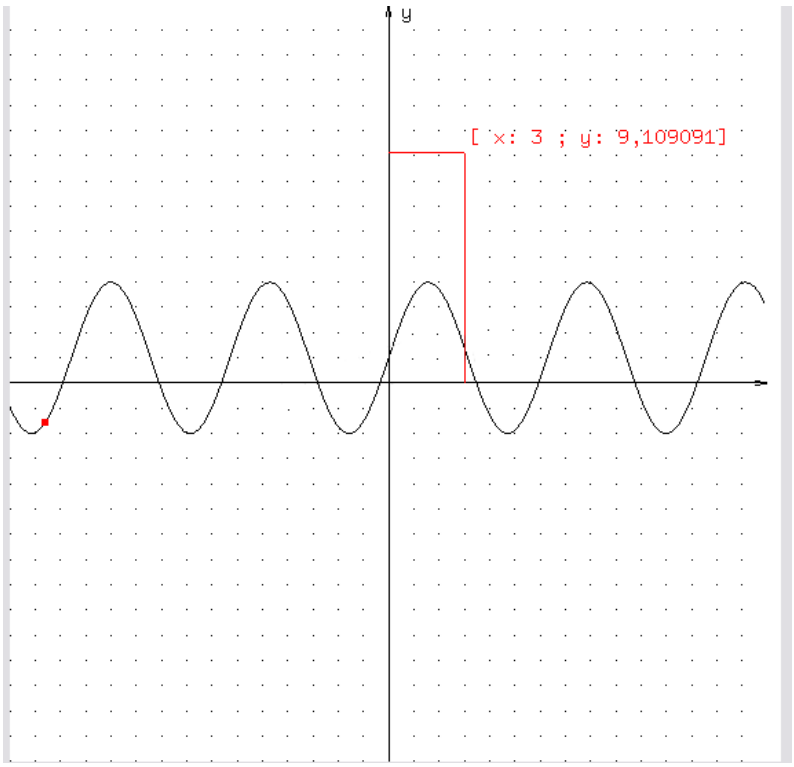

Рисунок Д.4 – Вигляд графіка функції в області форми

Ініціалізуйте змінні:

```
// розміри вікна
double ScreenW, ScreenH;
// відношення сторін вікна візуалізації
// для коректного переводу координат мишки в координати,
// прийняті в програмі
private float devX;
private float devY;
// масив, у якому будуть зберігатися
// значення x,y точок графіка
private float[,] GrapValuesArray;
// кількість елементів в масиві
private int elements_count = 0;
// прапорець, який означає, що масив значень координат
// графіка ще не заповнено
private bool not_calculate = true;
```

```
// номер комірки масиву, з якої будуть взяті координати
// для червоної точки для візуалізації поточного кадру
private int pointPosition = 0;
// допоміжні змінні для побудови ліній
// від курсору мишки до координатних осей
float lineX, lineY;
// поточні координати курсору мишки
float Mcoord_X = 0, Mcoord_Y = 0;
```

Розробимо код обробника події завантаження форми ***Form1_Load***. Тут необхідно ініціалізувати ***OpenGL*** для наступної візуалізації:

```
private void Form1_Load(object sender, EventArgs e)
{
  // ініціалізація бібліотеки glut
  Glut.glutInit();
  // ініціалізація режиму екрана
  Glut.glutInitDisplayMode
          (Glut.GLUT_RGB | Glut.GLUT_DOUBLE);
  // установка кольору очистки екрана (RGBA)
  Gl.glClearColor(255, 255, 255, 1);
  // установка порту вивода
  Gl.glViewport(0, 0, AnT.Width, AnT.Height);
  // активація проекційної матриці
  Gl.glMatrixMode(Gl.GL_PROJECTION);
  // очистка матриці
  Gl.glLoadIdentity();
  // визначення параметрів настройки проекції в
  // залежності від розміру сторін елемента AnT.
  if ((float)AnT.Width <= (float)AnT.Height)
  {
    ScreenW = 30.0;
    ScreenH = 30.0*(float)AnT.Height / (float)AnT.Width;
    Glu.gluOrtho2D(0.0, ScreenW, 0.0, ScreenH);
  }
```

```
  else
  {
    ScreenW = 30.0*(float)AnT.Width / (float)AnT.Height;
    ScreenH = 30.0;
    Glu.gluOrtho2D(0.0,
     30.0*(float)AnT.Width/(float)AnT.Height,      0.0,
30.0);
  }
  // збереження коефіцієнтів, необхідних для переведення
  //координат покажчика в віконній системі в координати,
  // прийняті в нашій OpenGL сцені
  devX = (float)ScreenW / (float)AnT.Width;
  devY = (float)ScreenH / (float)AnT.Height;
  // установка об'єктно-видової матриці
  Gl.glMatrixMode(Gl.GL_MODELVIEW);
  // старт лічильника, що відповідає виклик функції
  // візуалізації сцени
  PointInGrap.Start();
}
```

Тепер перейдемо до функції ***PointInGrap_Tick***. Вона викликається з затримкою в **30** мілісекунд, в ній ведеться відлік того, з якого елемента масиву з координатами графіка будуть братися координати, які використовуються для рисування червоної точки. Також з неї викликається функція ***Draw***, що відповідає за візуалізацію:

```
// функція обробник події таймера
private void PointInGrap_Tick(object sender, EventArgs
e)
{
  // якщо ми дійшли до останнього елемента масиву
  if (pointPosition == elements_count-1)
  // переходимо до початкового елемента
```

```
    pointPosition = 0;
  // функція візуалізаціїи
  Draw();
  // перехід до наступного елементу масиву
  pointPosition++;
}
```

Розглянемо кілька допоміжних функцій. Почнемо з функції ***AnT_MouseMove***. Це функція-обробник події ***MouseMove*** для елемента ***SimpleOpnGLControl (AnT)*** [7].

В даній функції виконується збереження поточних координат мишки, обчислення розмірів ліній, які будуть з'єднувати координати покажчика мишки з координатними осями (дві червоні лінії на рис. Д.4):

```
// обробка руху мишки над елементом AnT
private void AnT_MouseMove
           (object sender, MouseEventArgs e)
{
// зберігаємо координати мишки
Mcoord_X = e.X;
Mcoord_Y = e.Y;
// обчислюємо параметри для дорисовки ліній
// від покажчика мишки до координатних осей
lineX = devX * e.X;
lineY = (float)(ScreenH - devY * e.Y);
}
```

Тепер розробимо функцію, що встановлює координати вивода растрових символів у відповідності до координат **(x, y)**, а потім в цикле перебирає всі символи з указаного в параметрі рядка тексту. Кожний із символів візуалізується за допомогою функції, в якій задається шрифт для виведення і змінна типу ***char*** для візуалізації:

```
private void PrintText2D(float x, float y, string text)
{
  // встановлюємо позицію вивода растрових символів,
  // що передана в координатах x і y.
  Gl.glRasterPos2f(x, y);
  // в цикле foreach перебираємо значення з масиву text,
  // який містить значення рядка для візуалізації
  foreach (char char_for_draw in text)
  {
    //      символ      С      візуалізуємо      функцією
glutBitmapCharacter,
    // використовуючи шрифт GLUT_BITMAP_9_BY_15.
    Glut.glutBitmapCharacter
         (Glut.GLUT_BITMAP_9_BY_15, char_for_draw);
  }
}
```

В наступній функції ініціалізується масив координат і обчислюються всі координати графіка в залежності від заданого діапазону значень **x** і кроку їх зміни:

```
private void functionCalculation()
{
  // визначення локальних змінних X і Y
  float x = 0, y = 0;
  // ініціалізація масиву для значень 300 точок графіка
  GrapValuesArray = new float[300, 2];
  // лічильник елементів масиву
  elements_count = 0;
  // обчислення всіх значень y для x з проміжку
  // від -15 до 15 з кроком в 0.01f
  for (x = -15; x < 15; x += 0.1f)
  {
    // обчислення y для поточного x
    // за формулою y = (float)Math.Sin(x)*3 + 1;
    y = (float)Math.Sin(x)*3 + 1;
```

```
      // запис координати x
      GrapValuesArray[elements_count, 0] = x;
      // запис координати y
      GrapValuesArray[elements_count, 1] = y;
      // підрахунок елементів
      elements_count++;
    }
  // змінюємо прапорець «координати графіка не обчисленні»
    not_calculate = false;
  }
```

Функція, що виконує візуалізацію графіка

В цій функції перевіряється прапорець, що координати графіка обчислені і занесені в масив (змінна ***not_calculate***). У випадку, якщо прапорець вказує, що підрахунку значень ще не було, викликається функція, яка обчислить значення координат точок графіка і занесе їх в массив. Далі реалізується прохід циклом ***for*** по массиву значень координат точок графіка і їх візуалізація, при цьому точки об’єднуються в линию. Потім виконується рисування червоної точки в тих координатах, до яких дійшли, послйдовно перебираючи значення елементів в масиві координат:

```
  private void DrawDiagram()
  {
    // перевірка прарпорця
    if (not_calculate)
    {
      // якщо координати не обчислені, то викликаємо
      // функцию обчислення координат графіка
      functionCalculation();
    }
    // стартуємо відрисовку в режимі візуалізації точок,
    // що об’єднуються в лінії (GL_LINE_STRIP)
    Gl.glBegin(Gl.GL_LINE_STRIP);
```

```
  // рисуємо початкову точку
  Gl.glVertex2d (GrapValuesArray[0,0], GrapValuesArray[0,1]);
  // проходимо по масиву з координатами обчислених точок
  for (int ax = 1; ax < elements_count; ax+=2)
  {
    // передаємо в OpenGL інформацію про вершину,
    // що бере участь в побудові ліній
    Gl.glVertex2d(GrapValuesArray[ax,0],    GrapValuesArray[ax,
1]);
  }
  // завершуємо режим рисування
  Gl.glEnd();
  // установлюємо размір точок в 5 пікселів
  Gl.glPointSize(5);
  // поточний колір - червоний
  Gl.glColor3f(255, 0, 0);
  // активуємо режим виведеня точок (GL_POINTS)
  Gl.glBegin(Gl.GL_POINTS);
  // виводимо червону точку
  Gl.glVertex2d(GrapValuesArray[pointPosition, 0],
                GrapValuesArray[pointPosition, 1]);
  // завершуємо режим рисування
  Gl.glEnd();
  // встановлюємо розмір точок
  Gl.glPointSize(1);
}
```

В функції ***Draw()*** візуалірується координатна сітка під графіком, координатні осі і літери, що їх позначають, а також викликається функція рисування графіка і виводяться координати мишки з перпендикулярами на осі координат:

```
private void Draw()
{
  // очистка буфера кольору і буфера глибини
  Gl.glClear(Gl.GL_COLOR_BUFFER_BIT | Gl.GL_DEPTH_BUFFER_BIT);
  // очищення поточної матриці
  Gl.glLoadIdentity();
  // установка чорного кольору
  Gl.glColor3f(0, 0, 0);
  // стан матриці запам’ятовуємо в стек
  Gl.glPushMatrix();
  // виконуємо переміщення в просторі по осям X і Y
  Gl.glTranslated(15, 15, 0);
  // активуємо режим рисування
  Gl.glBegin(Gl.GL_POINTS);
  // створюємо сітку з точок
  for (int ax = -15; ax < 15; ax++)
  {
    for (int bx = -15; bx < 15; bx++)
    {
      // виведення точки
      Gl.glVertex2d(ax, bx);
    }
  }
  // завершуємо режим рисування примітивів
  Gl.glEnd();
  // активуемо режим рисування, кожні 2 послідовні
  // команди glVertex об’єднуються в лінії
  Gl.glBegin(Gl.GL_LINES);
  // рисуємо координатні осі і стрілки на їх кінцях
  Gl.glVertex2d(0, -15);
  Gl.glVertex2d(0, 15);
  Gl.glVertex2d(-15, 0);
  Gl.glVertex2d(15, 0);
```

```
  // вертикальна стрілка
  Gl.glVertex2d(0, 15);
  Gl.glVertex2d(0.1, 14.5);
  Gl.glVertex2d(0, 15);
  Gl.glVertex2d(-0.1, 14.5);
  // горизонтальна стрілка
  Gl.glVertex2d(15, 0);
  Gl.glVertex2d(14.5, 0.1);
  Gl.glVertex2d(15, 0);
  Gl.glVertex2d(14.5, -0.1);
  // завершуємо режим рисування
  Gl.glEnd();
  // виводимо підписи осей "x" і "y"
  PrintText2D(15.5f, 0, "x");
  PrintText2D(0.5f, 14.5f, "y");
  // викликаємо функцію рисування графіка
  DrawDiagram();
  // повертаємо матрицю зі стека
  Gl.glPopMatrix();
  // виводимо текст зі значеннями координат біля курсора
  PrintText2D(devX * Mcoord_X + 0.2f,
    (float)ScreenH - devY * Mcoord_Y + 0.4f,
    "[ x: " + (devX * Mcoord_X - 15).ToString() +
    " ; y: " +
    ((float)ScreenH - devY * Mcoord_Y - 15).ToString()
+"]");
  // встановлюємо червоний колір
  Gl.glColor3f(255, 0, 0);
  // включаємо режим рисування ліній
  Gl.glBegin(Gl.GL_LINES);
  Gl.glVertex2d(lineX, 15);
  Gl.glVertex2d(lineX, lineY);
  Gl.glVertex2d(15, lineY);
  Gl.glVertex2d(lineX, lineY);
  Gl.glEnd();
```

```
  // чекаємо завершення візуалізації кадра
  Gl.glFlush();
  // сигнал для оновлення елемента,
  // що реалізє візуалізацію.
  AnT.Invalidate();
}
```

Завдання для самостійного виконання

Послідовно змінючи параметри в коді, проаналізуйте отримані результати, і зробіть висновки про роботу програми.

Зауваження

Не забудьте ініціалізувати компонент ***SimpleOpenGLControl*** в конструкторі форми:

```
public Form1()
{
  InitializeComponent();
  AnT.InitializeContexts();
}
```

Навчальне видання

Лозовська Людмила Іванівна, Бандоріна Лілія Миколаївна,

Савчук Роман Вячеславович, Климкович Тетяна Олександрівна

ПРИКЛАДНЕ ПРОГРАМУВАННЯ В БІЗНЕСІ

Навчальний посібник

Відповідальний редактор Л. І. Лозовська
Комп'ютерна верстка Л. І. Лозовська
Дизайн обкладинки Л. І. Лозовська

Експертний висновок склав канд. екон. наук, доц. К. О. Удачина

Зареєстровано НМВ УДУНТ (№ 4 від 14.02.2025)

Формат 60х84 *1/16.* Ум. друк. арк. 8,37. Обл.-вид. арк. 8,47.
Зам. № 45.

Видавець: Український державний університет науки і технологій
вул. Лазаряна, 2, ауд.2216, ауд. 263 (наукова бібліотека)
м. Дніпро, 49010.
Свідоцтво суб'єкта видавничої справи ДК №7709 від 14.12.2022


IT в бізнесі