Тема 10. Принципи створення візуальних інтерфейсів

Мета лекцій – Отримати теоретичні загнання та практичні навички розробки багатовіконних Windows-додатків, що містять основне меню в головному вікні.

Дана лекція сприяє напрацюванню наступних **компетентностей** у відповідності до Національної рамки кваліфікації:

Знання:

особливості взаємодії користувача з Windows;

типова структуру Windows-програми;

технологія створення керуючих елементів;

Уміння:

використовувати стандартні вікна повідомлень;

розробити Windows-додаток, що складається з декількох вікон і здійснює виклик з головного меню основного вікна відповідні програми.

Комунікації:

аргументована взаємодія з клієнтами та замовниками при виборі технології розробки Windows-програми;

робота в команді над окремими фрагментами основного та контекстного меню Windows-додатка.

Автономність і відповідальність:

самостійне формулювання рекомендацій щодо вибору ієрархічної структури основного та контекстного меню Windows-додатка.

здатність обґрунтувати доцільності застосування певної технології розробки Windows-програми.

Основні положення

Дві технології створення Windows-додатків.

Пакет Visual Studio включає великий набір засобів розробки, які автоматизують більшу частину процесу створення Windows-програми. За допомогою цих засобів можна створювати та поміщати в потрібне місце різні елементи управління і меню, які будуть використовувані додатком.

Visual Studio допомагає також створювати класи і методи, які необхідні для кожного керуючого елемента. Це дуже зручний інструмент для створення більшості Windows-додатків, хоча і не єдино можливий.

Windows-програми можна створювати і за допомогою текстового редактора з подальшою компіляцією вихідного коду подібно до того, як це робиться з консольними додатками.

Відносно прості Windows-програми, які розглядаються в цієї лабораторної роботі, досить невеликі за розміром, тому їх створення дається тут у формі, яка підходить для використання текстового редактора. Але загальна структура, розробка та організація програм залишаються такими ж, як і при використанні автоматизованих засобів розробки, які будуть розглянути у наступних лабораторних роботах.

Таким чином, матеріал цієї лабораторної роботи може бути застосовано до будь-якого способу створення програм.

Особливості взаємодії користувача з Windows.

Перш ніж приступати до Windows-програмування, необхідно зрозуміти, як користувач взаємодіє з Windows, оскільки саме цей фактор визначає архітектуру всіх Windows-програм.

Робота користувача з Windows в корені відрізняється від взаємодії, які реалізовані в консольних програмах..

У разі консольної програми саме ваша програма ініціює взаємодію з операційною системою. Прикладом може служити програма, яка запитує вхідні дані і виводить результати шляхом виклику методів Read () або WriteLine (). Таким чином, програми, написані «традиційним способом», самі звертаються до операційної системи, а не операційна система до них.

Але відносно «своїх» програм Windows передбачає зовсім іншу модель відношень: саме Windows повинна звертатися до вашій програмі.

Процес взаємодії організований таким чином: програма очікує до тих пір, поки не отримає повідомлення від Windows. Отримавши його, програма повинна вжити відповідну дію. Відповідаючи на повідомлення, вона може викликати метод, який визначено в Windows, але головне тут те, що ініціатором взаємодії все-таки є Windows.

Таким чином, загальний формат всіх Windows-програм продиктований механізмом спілкувань, який і лежить в основі взаємодії з Windows.

Існує безліч різних повідомлень, які Windows може послати програмі. Наприклад, при кожному клацанні кнопкою миші у вікні вашої програми буде послано повідомлення, пов'язане з клацанням кнопкою миші. З точки зору програми повідомлення надходять випадковим чином. Ось чому Windows-програми нагадують програми, які керуються перериваннями.

Windows-форми.

Ядром Windows-програм, написаних на C#, є форма.

Форма інкапсулює основні функції, необхідні для створення вікна, його відображення на екрані і отримання повідомлень. Форма може являти собою вікно будь-якого типу, включаючи основне вікно програми, дочірнє або навіть діалогове вікно.

Спочатку вікно створюється порожнім. Потім в нього додаються меню і елементи управління, наприклад екранні кнопки, списки і прапорці. Таким чином, форму можна представити у вигляді контейнера для інших Windows-об'єктів.

Коли вікну надсилається повідомлення, воно перетворюється на подію. Отже, щоб обробити Windows-повідомлення, достатньо для нього зареєструвати обробник подій. При отриманні цього повідомлення обробник подій буде викликатися автоматично.

Клас Form.

Форма створюється за допомогою реалізації об'єкта класу Form або класу, який є похідним від Form.

Клас Form крім поведінки, обумовленої власними елементами, які демонструють поведінку що успадкована від предків. Серед його базових класів виділяються своєю значимістю клас Control.

Клас Control.

Клас Control визначає риси, властиві всім Windows-елементам управління. Той факт, що клас Form виведений з класу Control, дозволяє використовувати форми для створення елементів управління.

Використання деяких елементів класів Form і Control демонструється в прикладах, які наведені нижче.

Приклад 1. Типова структура найпростішого Windows-програми.

```
using System;
using System.Windows.Forms;
// Клас WinSkel, похідний від класу Form
class WinSkel : Form
```

```
{
    public WinSkel()
    {
        // Привласнюємо вікна ім'я
        Text = "Скелет (шаблон) Windows-вікна";
    }
    // Метод Main, який використовується тільки для запуску програми
    [STAThread]
    public static void Main()
    {
        WinSkel skel = new WinSkel(); // створюємо форму
        // Запускаємо механізм функціонування вікна
        Application.Run(skel);
    }
}
```

Щоб скомпілювати цю програму за допомогою Visual Studio можна діяти за двома варіантами.

Варіант 1. Створити консольний додаток. Після чого перетворити його в графічний додаток шляхом відповідної настройки середовища розробки і підключенні до нього простору імен System.Windows.Forms. Проте надалі ми будемо використовувати більш простий шлях (варіант 2), який спочатку є орієнтований на графічні додатки.

Варіант 2. Створіть новий проект Windows Form Application. Як результат автоматично буде получено шаблон Windows-додатку. (рис.1).

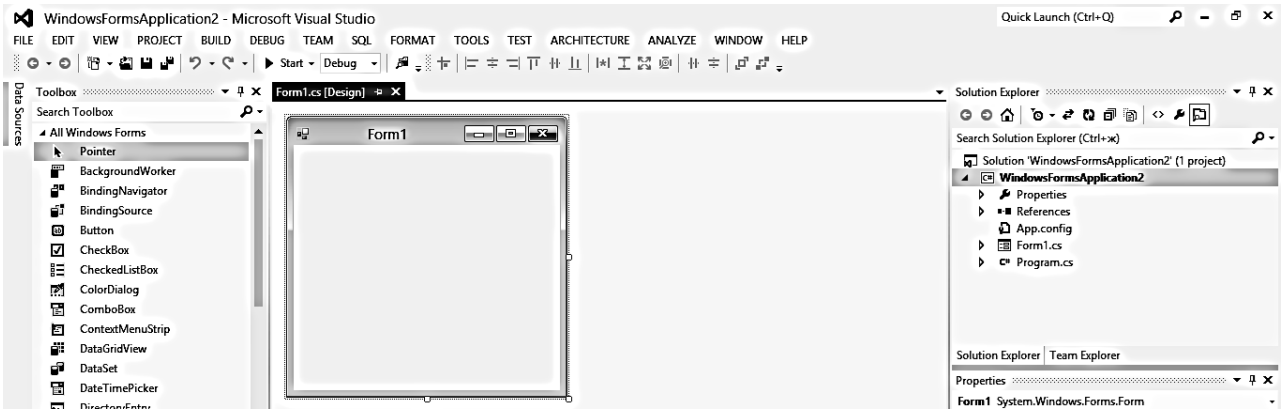


Рис. 1. Шаблон Windows-додатку, який створюється автоматично

З цим проектом буде пов'язаний файл Forml.cs. Оскільки наша задача створити цей додаток «вручну» за допомогою відповідного коду, треба видалити цей файл.

Потім, клацнувши лівою кнопкою миші на імені файлу Program.cs, Ви повинні побачити наступне діалогове вікно (рис.2):

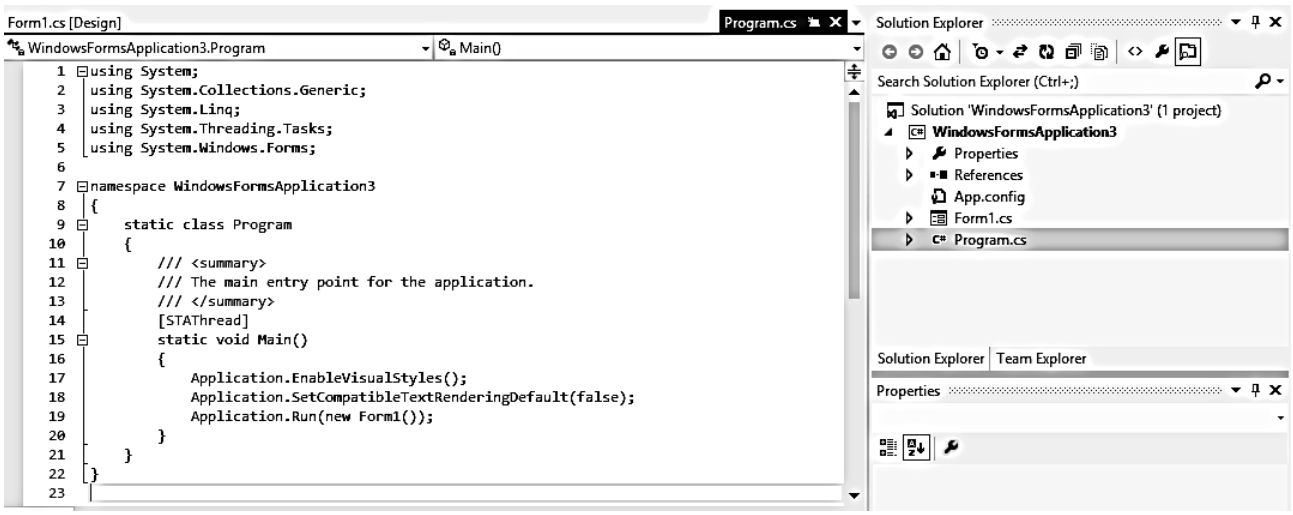


Рис. 2. Вміст файлу Program.cs, який Visual Studio створює автоматично

Далі слід замінити вміст вікна редактора тексту на код, який надано в прикладі 1. Після чого виконати компіляцію і запуск програми на виконання. Результат наведено нижче (рис. 3).

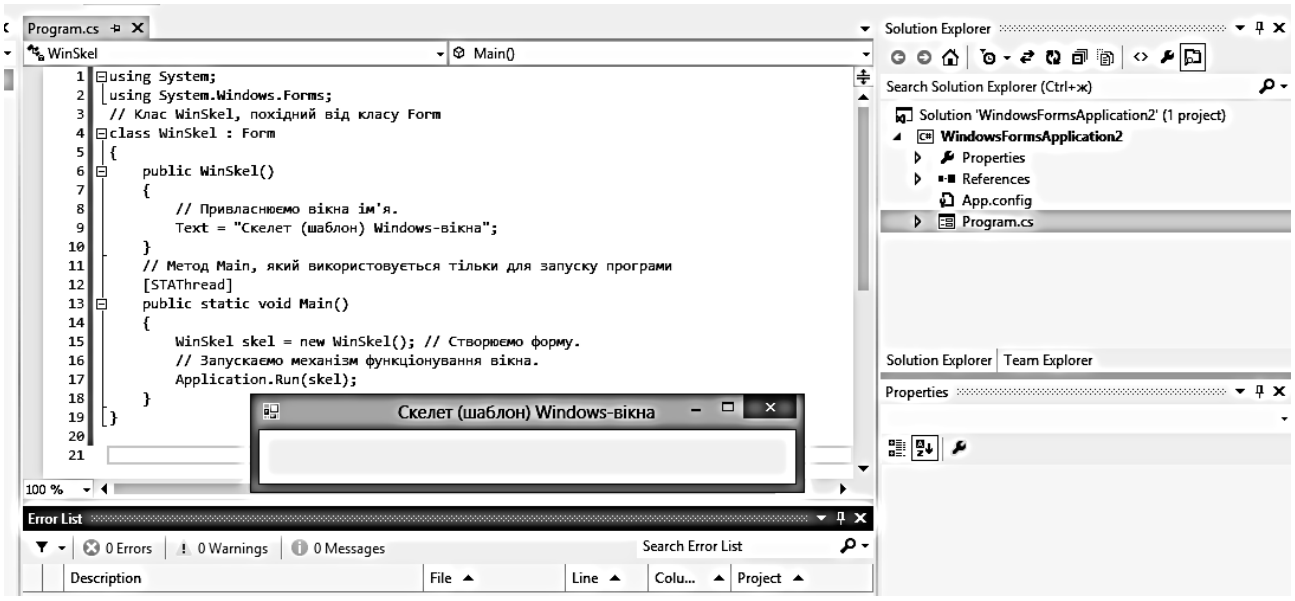


Рис. 3. Типова структура найпростіший Windows-програми і результат її виконання

Ця програма створює і відображає вікно, яке не містить інших елементів управління. Тим не менше, воно дозволяє показати дії, необхідні для побудови повнофункціональної Windows-програми. Іншими словами, вікно є стартовий майданчик, на якій можна побудувати більшість Windows-додатків.

Розглянемо код прикладу 1 по строкам.

Перш за все, зверніть увагу на те, що програма включає два простору імен: System і System.Windows.Forms.

Простір імен System необхідне для використання атрибута STAThread, який передує методу Main (), а System.Windows.Forms призначене для підтримки підсистеми Windows.Forms.

Потім створюється клас WinSkel, який успадковує клас Form. Отже, клас WinSkel визначає тип форми. В даному випадку це найпростіша (мінімальна) форма на відміну від тієї, яку автоматично генерує майстер WindowsForms (див. рис. 1 і рис. 2).

У тілі конструктора класу WinSkel міститься тільки один рядок коду:

```
Text = "Скелет (шаблон) Windows-вікна";
```

Тут встановлюється властивість Text, яке містить назву вікна. Таким чином, після виконання інструкції присвоєння рядок заголовка вікна буде містити текст «Скелет (шаблон) Windows-вікна».

Властивість Text успадковане від класу Control і визначається так:

```
public virtual string Text {get; set; }
```

Метод Main () оголошено подібно іншим методам Main (), що входять до складу програм.

Заголовку методу Main () передує властивість STAThread.

Microsoft заявляє, що ця властивість повинна мати метод Main () в кожній Windows-програмі.

Властивість STAThread встановлює модель організації потокової обробки (threading model). В даному випадку мова йде про модель з одно поточним управлінням, тобто таку обробку даних, коли всі об'єкти виконуються в єдиному процесі (single-threaded apartment - STA).

Розгляд моделей організації потокової обробки виходить за рамки даної лабораторної роботи.

У методі Main () створюється об'єкт класу WinSkel з ім'ям skel. Цей об'єкт потім передається методу Run (), визначеним в класі Application:

```
Application.Run (skel);
```

Ця інструкція запускає механізм функціонування вікна. Клас Application визначається в просторі імен System.Windows.Forms і инкапсулює можливості, які властиві всім Windows-додаткам.

Ось як визначається використовуваний тут метод Run ():

```
public static void Run (Form ob)
```

Як параметр він приймає посилання на форму. Оскільки клас WinSkel виведено з класу Form, об'єкт типу WinSkel можна передати методу Run ().

Результат виконання програми – Windows-вікно (див. рис. 3). Воно має стандартний розмір (300 пікселів по ширині і 300 пікселів по висоті).

Це вікно повністю функціонально. Можна змінити його розміри (на рис. 3 розмір вікна зменшено), перемістити, згорнути, відновити і закрити. Таким чином, основні властивості, притаманні практично всім вікнам, були досягнуті написанням всього декількох рядків програмного коду.

Для порівняння: така ж програма, але написана на мові C++ безпосередньо викликає інтерфейс Windows API, вона зажадала б приблизно в п'ять разів більше програмних рядків!

Попередній приклад продемонстрував основні принципи створення Windows додатків, заснованих на застосуванні вікон.

Отже, щоб створити форму необхідно:

створіть клас, похідний від класу Form;

ініціалізувать цю форму відповідно до вимог програми;

створіть об'єкт похідного класу;

викликати метод Application.Run () для цього об'єкта.

Створення керуючих елементів (на прикладі кнопки).

У загальному випадку функціональність вікна забезпечується елементами двох типів: елементами управління і меню. Саме за допомогою цих елементів і взаємодіє користувач з програмою.

У Windows визначені різні типи керуючих елементів, включаючи екранні кнопки, прапорці, перемикачі та вікна списків. Незважаючи на відмінності між ними, технології їх створення приблизно однакові. Для прикладу розглянемо технологію створення кнопки.

Екранна кнопка інкапсулювана в класі Button, який виведено з абстрактного класу ButtonBase.

Оскільки клас ButtonBase призначено для реалізації поведінки віконного елемента управління, він успадковує клас Control.

У класі Button визначений тільки один конструктор:

```
public Button ();
```

Цей конструктор створює кнопку стандартного розміру, розташовану всередині вікна. Ця кнопка не містить опису, тому, перш ніж використовувати її, необхідно описати її, присвоївши властивості Text відповідний текстовий рядок.

Для вказівки місця розташування кнопки у вікні необхідно привласнити властивості Location координати її верхнього лівого кута.

Властивість Location успадкована від класу Control і визначається так:

```
public Point Location {get; set; }
```

Координати зберігаються в структурі Point, яка визначена в просторі імен System.Drawing. Вона включає наступні властивості:

```
public int X {get; set; }
public int Y {get; set; }
```

Таким чином, щоб створити кнопку з написом «Клацніть» і прив'язати її до точки з координатами 100, 200, треба використати наступну послідовність інструкцій:

```
MyButton = new Button ();
MyButton.Text = "Клацніть";
MyButton.Location = new Point (100, 200);
```

Розміщення кнопки на формі.

Після створення кнопки її необхідно помістити на форму. Це реалізується за допомогою методу Add (), який викликається з колекції елементів управління, пов'язаних з формою. Ця колекція доступна за допомогою властивості Controls, яка успадкована від класу Control. Ось як визначається метод Add ():

```
public virtual void Add (Control cnt)
```

Тут параметр cntl означає елемент керування, що додається. Після того як елемент буде додано до складу форми, він стане видимим при відображенні самої форми.

Приклад 2. Розміщення кнопки на формі.

```
using System;
using System.Windows.Forms;
using System.Drawing;

class ButtonForm : Form
{
   Button MyButton = new Button();
   public ButtonForm()
   {
```

```
        Text = "Використання кнопки";
        MyButton.Text = "Клацніть";
        MyButton.Location = new Point(100, 200);
        Controls.Add(MyButton);
    }
    [STAThread]
    public static void Main()
    {
        ButtonForm skel = new ButtonForm();
        Application.Run(skel);
    }
}
```

У цій програмі створюється клас ButtonForm, який є похідним від класу Form. Він містить поле типу Button з ім'ям MyButton. У конструкторі класу ButtonForm кнопка створюється, ініціалізується і поміщається на форму. При виконанні цієї програми відображається наступне вікно (рис. 4):

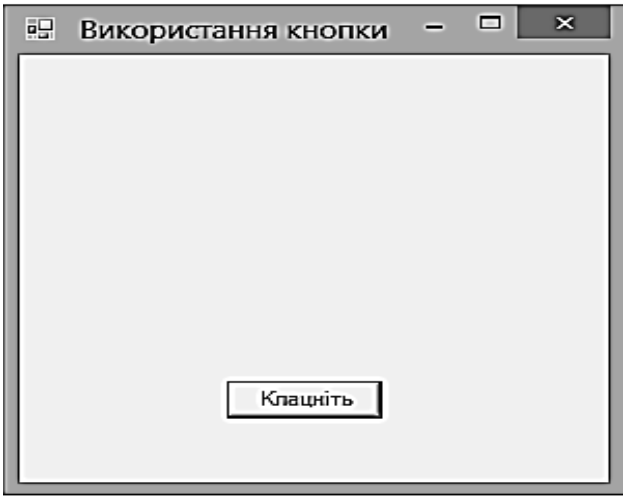


Рис. 4. Розміщення кнопки на форму

Ви можете клацнути на кнопці, але нічого не станеться. Щоб змусити кнопку виконувати будь-які дії, необхідно додати в програму обробник подій.

Обробка подій Windows-додатках (на прикладі обробки повідомлення від кнопки).

Щоб програма реагувала на клацання на кнопці (або на яку-небудь іншу дію користувача), необхідно забезпечити обробку повідомлення, яке генерує ця кнопка.

У загальному випадку, коли користувач впливає на елемент управління, його дія передається програмі у вигляді повідомлення.

У C# -програмі, яка заснована на застосуванні вікон, такі повідомлення обробляються обробниками подій.

Отже, щоб отримати повідомлення, в програму необхідно включити власний обробник подій, який слід додати в список обробників, що викликаються при генеруванні повідомлення.

Для повідомлень, пов'язаних з клацанням на кнопці, це означає додавання обрабника для події Click.

Подія Click визначається в класі Button. Вона успадкована від класу Control. ЇЇї загальний формат такий:

```
public Event EventHandler Click;
```

Делегат EventHandler визначається так:

```
public delegate void EventHandler (object who, EventArgs args)
```

Об'єкт, який згенерував подію, передається в параметрі who, а інформація, яка пов'язана з цією подією, - в параметрі args.

Для багатьох подій у якості параметра args буде служити об'єкт класу, який виведено з класу EventArgs. Оскільки клацання на кнопці не вимагає додаткової інформації, тому при обробці події клацання не потрібно турбуватися про аргументи цієї події.

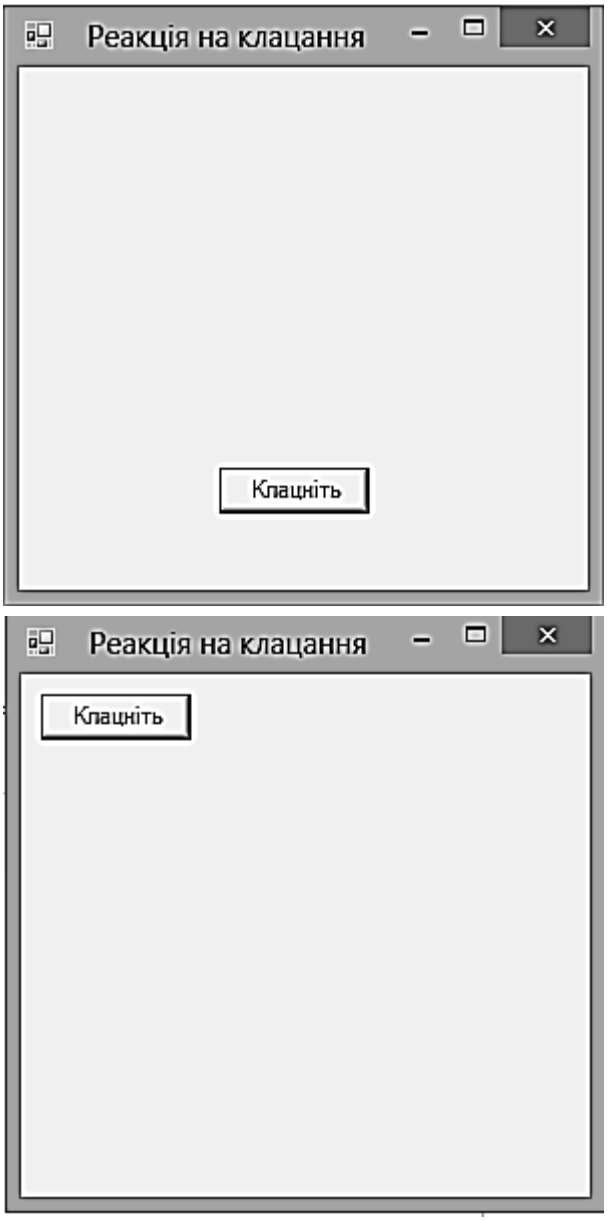
Наступна програма заснована на попередній, але з додаванням коду реакції на клацання. При кожному натисканні на кнопці буде змінюватися її місце розташування.

Приклад 3. Обробка повідомлень від кнопки.

```
using System;
using System.Windows.Forms;
using System.Drawing;
class ButtonForm : Form
{
   Button MyButton = new Button();  // створення екземпляру кнопки
```

```
    public ButtonForm()                    // конструктор
    {
    Text = "Реакція на клацання";          // напис на формі
    MyButton.Text = "Клацніть";            // напис на кнопці
    MyButton.Location = new Point (100, 200); // координати кнопки
    // Додаємо в список подій обробник подій кнопки
    MyButton.Click += new EventHandler (MyButtonClick);
    Controls.Add (MyButton); // додаємо кнопку на форму
    }
    [STAThread]
    public static void Main()
    {
       ButtonForm skel = new ButtonForm();
       Application.Run(skel);
    }
    // Оброблювач для кнопки MyButton.
    protected void MyButtonClick(object who, EventArgs e)
    {
       if (MyButton.Top == 200)
          MyButton.Location = new Point(10, 10);
       else
          MyButton.Location = new Point(100, 200);
    }
}
```

Результат виконання програми:



Початковий стан вікна Стан після натискання на кнопку

Рис.5. Обробка повідомлень від кнопки

Оброблювач MyButtonClick () використовує таку ж сигнатуру, як і наведений вище делегат EventHandler, а це означає, що обробник можна додати в ланцюжок обробників для події Click.

Зверніть увагу на те, що в його визначенні використано модифікатор типу protected. І хоча це не є обов'язковою вимогою, така модифікація має сенс, оскільки обробники подій не призначені для виклику кодом, а використовуються лише для відповіді на події.

У коді обробника координати верхньої межі кнопки визначаються за допомогою властивості Top.

Наступні властивості визначаються для всіх елементів управління (вони задають координати верхнього лівого та нижнього правого кутів):

```
public int Top {get; set; }
public int Bottom {get; }
public int Left {get; set; }
public int Right {get; }
```

Зверніть увагу на те, що місце розташування елемента керування можна змінити за допомогою властивостей Top і Left, але не властивостей Bottom і Right, оскільки останні призначені тільки для читання. (Для зміни розміру елемента керування можна використовувати властивості Width і Height.)

При отриманні події, яка пов'язана з клацанням на кнопці, перевіряється координата її верхньої межі, і, якщо вона дорівнює початковому значенню 200, для кнопки встановлюються нові координати: 10, 10. В іншому випадку кнопка повертається у вихідне положення з координатами 100 , 200. Тому при кожному натисканні на кнопці її місце розташування змінюється.

Перш ніж обробник MyButtonClick () зможе отримувати повідомлення, його необхідно додати в ланцюжок обробників подій, які пов'язані з подією Click. Це реалізується в конструкторі класу ButtonForm за допомогою такої інструкції:

```
MyButton.Click + = new EventHandler (MyButtonClick);
```

Після виконання цієї інструкції при кожному натисканні на кнопці буде викликатися обробник подій MyButtonClick ().

Використання вікна повідомлень.

Одним з найбільш корисних вбудованих засобів Windows-додатків є вікно повідомлень. Воно дозволяє відображати повідомлення. З його допомогою можна також отримати від користувача такі прості відповіді (на поставлені питання), як Так, Ні або ОК.

У програмі, яка заснована на застосуванні вікон, вікно повідомлень підтримується класом MessageBox. При цьому об'єкт класу створювати

не потрібно. Для його досить викликати певний в цьому класі статичний метод Show ().

Метод Show () використовується в декількох форматах. Один з них виглядає так:

```
public static DialogResult Show (
                                    string    msg,
                                    string    caption,
                                    MessageBoxButtons    mbb
                                    )
```

Рядок, що відображається всередині вікна, передається в параметрі msg.

Тема вікна повідомлення - в параметрі caption.

Кнопки, які відображаються у вікні, задаються параметром mbb.

Метод повертає відповідь користувача.

Значення, що повертається методом Show (), означає, яка кнопка натиснута користувачем. Це може бути одне з наступних значень:

Abort	Cancel	Ignore	No
None	OK	Retry	Yes

Наприклад: if (result == DialogResult.Yes) Application.Exit ();

MessageBoxButtons - це перерахування, яке визначає наступні значення:

AbortRetrylgnore	OK	OKCancel
RetryCancel	YesNo	YesNoCancel

Кожне з цих значень описує кнопки, які будуть включені у вікно повідомлень.

Наприклад, якщо параметр mbb містить значення YesNo, то у вікні повідомлень будуть відображені кнопки Yes і No.

У програмі можна перевірити значення, що повертається методом Show (), і визначити лінію поведінки, яка обрана користувачем.

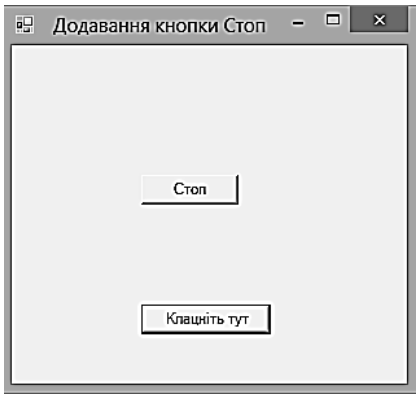
Наприклад, якщо у вікні повідомлення користувач попереджається про можливість перезапису файлу, то програма запобіжить перезапис, якщо користувач клацне на кнопці Cancel, або виконає її, якщо користувач клацне на ОК.

Приклад 4. Наступна програма заснована на попередній, але з додаванням кнопки «Стоп» і вікна повідомлень.

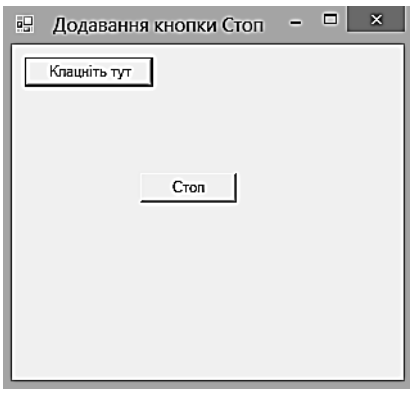
```
using System;
using System.Windows.Forms;
using System.Drawing;
class ButtonForm: Form
{
Button MyButton;
Button StopButton;

        public ButtonForm ()   // конструктор
        {
        Text = "Додавання кнопки Стоп";

        // Створюємо кнопки.
        MyButton = new Button ();
        MyButton.Text = "Клацніть тут";
        MyButton.Location = new Point (100, 200);
        MyButton.Width = 100;
        StopButton = new Button ();
        StopButton.Text = "Стоп";
        StopButton.Location = new Point (100, 100);

        // Додаємо обробники подій.
        MyButton.Click += new EventHandler (MyButtonClick);
        Controls.Add (MyButton);
        StopButton.Click+= new EventHandler (StopButtonClick);
        Controls.Add (StopButton);
        }
[STAThread]
public static void Main ()
        {
                ButtonForm skel = new ButtonForm ();
```

```
            Application.Run (skel);
        }
// Оброблювач подій для кнопки MyButton.
protected void MyButtonClick (object who, EventArgs e)
        {
        if (MyButton.Top == 200)
            MyButton.Location = new Point (10, 10);
        else
        MyButton.Location = new Point (100, 200);
        }
// Оброблювач подій для кнопки StopButton.
protected void StopButtonClick (object who, EventArgs e)
        {
        // Якщо користувач відповість клацанням на кнопці Yes,
        // програма буде завершена.
        DialogResult result = MessageBox.Show ("Зупинити програму?",
                                "Завершення", MessageBoxButtons.YesNo);
        if (result == DialogResult.Yes)
        Application.Exit ();
        }
}
```

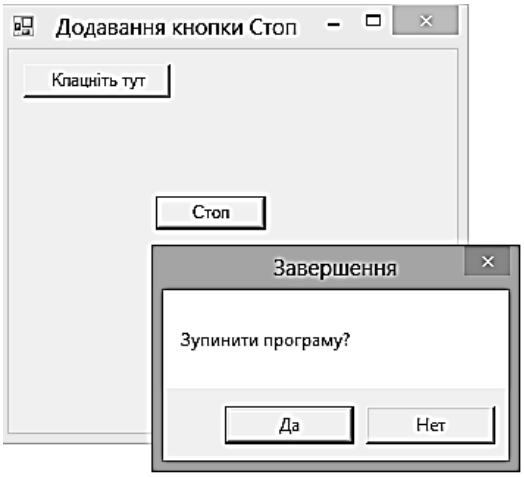
Результат виконання програми наведено на рис. 6.



Після запуску програми



Після натискання кнопки «Клацніть тут»



Після натискання кнопки «Стоп»

Рис.6. Додавання кнопки «Стоп»

У обробнику подій, що пов'язано з кнопкою «Стоп», організовано відображення вікна повідомлень, в якому користувачеві пропонується відповісти на питання, чи бажає він зупинити програму. Якщо користувач відповість клацанням на кнопці «Да», програма зупиниться. Якщо ж він клацне на кнопці «Нет», виконання програми буде продовжено.

Розробка елементів основного меню Windows-додатка.

Головне вікно практично всіх Windows-додатків включає меню, яке розташовано вздовж верхньої його межі. Воно називається основним.

Основне меню зазвичай містить такі категорії верхнього рівня, як Файл, Виправлення і Сервіс. З основного меню можна отримати меню, що розкриваються, які в свою чергу містять команди, пов'язані з відповідною категорією.

При виборі елемента меню генерується повідомлення. Отже, щоб виконати команду меню, програма повинна присвоїти кожному елементу меню відповідний обробник подій.

Основне меню створюється шляхом комбінації двох класів. Перший клас - MainMenu – інкапсулює загальну структуру меню, а другий - Menultem - окремий його елемент.

Елемент меню може становити або кінцеве дію (наприклад, закрити), чи-бо активізувати інше спадне меню.

Обидва класи - MainMenu і Menultem - успадковують клас Menu.

При виборі елемента меню генерується подія Click, яке визначено в класі MenuItem. Отже, щоб обробити вибір елемента меню, в список обробників події Click, які пов'язані з цим елементом, необхідно додати відповідний обробник.

Кожна форма має властивість Menu, яка визначена таким чином:

```
public MainMenu Menu {get; set; }
```

За замовчуванням цій властивості не присвоїли ніяке меню. Щоб відобразити основне меню, цю властивість необхідно «налаштувати» відповідним чином.

Для створення основного меню виконайте такі дії.

1. Створіть об'єкт класу MainMenu.

2. У об'єкт класу MainMenu додайте об'єкти класу MenuItem, які описують категорії верхнього рівня. Ці елементи меню додаються в колекцію типу MenuItems, яка пов'язана з основним меню.

3. Для кожного MenuItem-об'єкта верхнього рівня додайте список MenuItem-об'єктів, який визначає спливаюче меню що пов'язане з елементом меню верхнього рівня. Ці елементи меню додаються в колекцію MenuItems, яка пов'язана з кожним елементом меню верхнього рівня.

4. Додайте обробники подій для кожного елемента меню.

5. Дайте об'єкту класу MainMenu властивості Menu, які пов'язані з формою.

У наступному фрагменті коду показано, як створити меню Файл, який містить три команди: «Відкрити», «Закрити» та «Вийти».

```
MyMenu = new MainMenu (); // створюємо об'єкт основного меню

// Додаємо в це меню елемент верхнього рівня.
MenuItem m1 = new MenuItem ("Файл");
MyMenu.MenuItems.Add (m1);

// Створення підміню "Файл".
MenuItem subm1 = new MenuItem ("Відкрити");
m1.MenuItems.Add (subm1);
 MenuItem subm2 = new MenuItem ("Закрити");
```

```
m1.MenuItems.Add (subm2);
 MenuItem subm3 = new MenuItem ("Вийти");
m1.MenuItems.Add (subm3);
```

Наведена послідовність інструкцій починається зі створення об'єкта класу MainMenu з ім'ям MyMenu. Цей об'єкт буде знаходитися на верхньому рівні структури меню.

Потім створюється елемент меню m1 із заголовком «Файл». Він додається безпосередньо до об'єкту MyMenu.

Після цього створюється спливаюче меню, яке пов'язане з командою «Файл» основного меню. Зверніть увагу на те, що елементи меню, що розкривається, додаються до об'єкту m1, який є елементом «Файл» основного меню.

Якщо один MenuItem-об'єкт додається до іншого, то об'єкт що додається стає частиною спадному меню, яке пов'язане з елементом, до якого додається MenuItem-об'єкт.

Отже, після того як елементи subm1 - subm3 будуть додані до елементу m1, при виборі команди «Файл» відобразиться спадне меню, що містить команди «Відкрити», «Закрити» та «Вийти».

Створивши меню, для кожного його елемента необхідно створити пов'язані з ним обробники подій.

При виборі користувачем команди меню генерується подія Click. Тому при виконанні наступної послідовності інструкцій елементам subm1 - subm3 будуть призначені відповідні обробники подій.

```
// Додаємо обробники подій для елементів меню.
subm1.Click += new EventHandler (MMOpenClick);
subm2.Click += new EventHandler (MMCloseClick);
subm3.Click += new EventHandler (MMExitClick);
```

Таким чином, якщо користувач вибере команду «Вийти», виконається обробник подій MMExitClick ().

Нарешті, властивості Menu форми потрібно присвоїти об'єкт класу MainMenu:

```
Menu = MyMenu; // призначаємо меню формі
```

Після виконання цієї інструкції вікно буде відображатися разом з меню, при виборі команд якого будуть викликатися відповідні обробники подій.

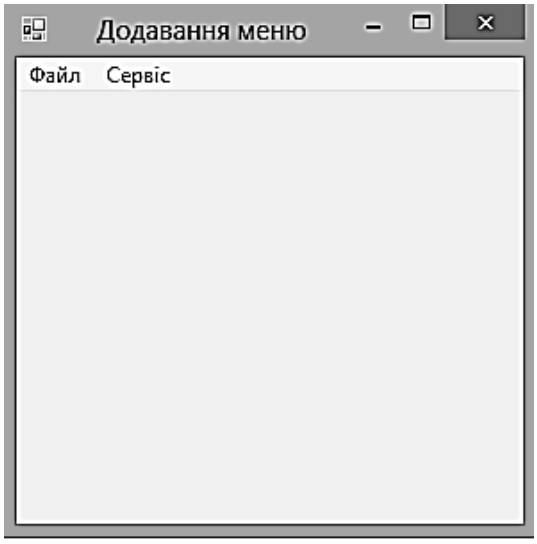
Приклад 5. Програма демонструє створення основного меню та обробку подій, які пов’язані з вибором відповідних команд.

```
using System;
using System.Windows.Forms;
class MenuForm: Form
{
    MainMenu MyMenu;     // оголошуємо ім'я об'єкта основного меню
    public MenuForm ()    // конструктор
    {
    Text = "Додавання меню";
    // Створюємо об'єкт основного меню.
    MyMenu = new MainMenu ();
    // Додаємо в це меню елементи (m1, m2) верхнього рівня.
    MenuItem m1 = new MenuItem ("Файл");
    MyMenu.MenuItems.Add (m1);
    MenuItem m2 = new MenuItem ("Сервіс");
    MyMenu.MenuItems.Add (m2);

    // Створення підміню "Файл".
    MenuItem subm1 = new MenuItem ("Відкрити");
    m1.MenuItems.Add (subm1);
    MenuItem subm2 = new MenuItem ("Закрити");
    m1.MenuItems.Add (subm2);
    MenuItem subm3 = new MenuItem ("Вийти");
    m1.MenuItems.Add (subm3);

    // Створюємо підміню "Сервіс".
    MenuItem subm4 = new MenuItem ("Координати");
    m2.MenuItems.Add (subm4);
    MenuItem subm5 = new MenuItem ("Змінити розмір");
    m2.MenuItems.Add (subm5);
    MenuItem subm6 = new MenuItem ("Відновити");
    m2.MenuItems.Add (subm6);
```

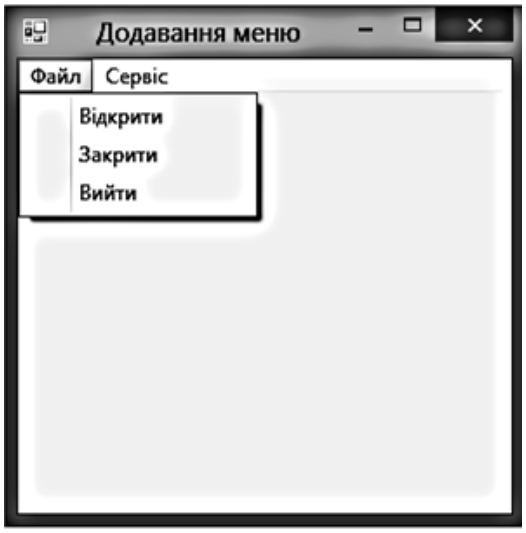
```
// Додаємо обробники подій для елементів меню.
subm1.Click+= new EventHandler (MMOpenClick);
subm2.Click+= new EventHandler (MMCloseClick);
subm3.Click+= new EventHandler (MMExitClick);
subm4.Click+= new EventHandler (MMCoordClick);
subm5.Click+= new EventHandler (MMChangeClick);
subm6.Click+= new EventHandler (MMRestoreClick);

// Призначаємо меню формі.
Menu = MyMenu;
}  // закінчення конструктора
[STAThread]
public static void Main ()
{
MenuForm skel = new MenuForm ();
Application.Run (skel);
}

// Оброблювач для команди меню "Координати".
protected void MMCoordClick (object who, EventArgs e)
{
// Створюємо рядок, яка містить три координати.
string size =
String.Format ("{0}: {1}, {2} \n {3}: {4}, {5}",
"Вгорі, Ліворуч", Top, Left,
"Унизу, Ліворуч", Bottom, Right);
// Відображаємо вікно повідомлень.
MessageBox.Show (size, "Координати вікна",
MessageBoxButtons.OK);
}

// Оброблювач для команди меню "Змінити розмір".
protected void MMChangeClick (object who, EventArgs e)
{
Width = Height = 200;
}
```

```
        // Оброблювач для команди меню "Відновити".
        protected void MMRestoreClick (object who, EventArgs e)
        {
        Width = Height = 300;
        }

        // Оброблювач для команди меню "Відкрити".
        protected void MMOpenClick (object who, EventArgs e)
        {
        MessageBox.Show ("Неактивна команда", "Заглушка",
        MessageBoxButtons.OK);
        }
    // Оброблювач для команди меню "Закрити".
    protected void MMCloseClick (object who, EventArgs e)
    {
    MessageBox.Show ("Неактивна команда", "Заглушка",
    MessageBoxButtons.OK);
    }

    // Оброблювач для команди меню "Вийти".
    protected void MMExitClick (object who, EventArgs e)
    {
    DialogResult result = MessageBox.Show ("Зупинити програму?",
    "Завершення",
    MessageBoxButtons.YesNo);
    if (result == DialogResult.Yes) Application.Exit ();
    }
}
```

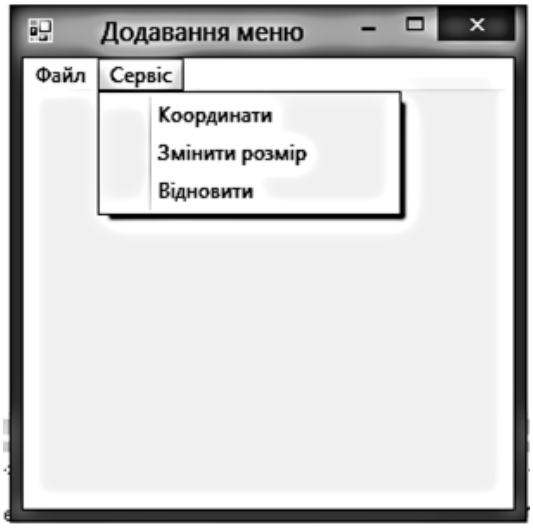
Результат виконання програми наведено на рис. 7.



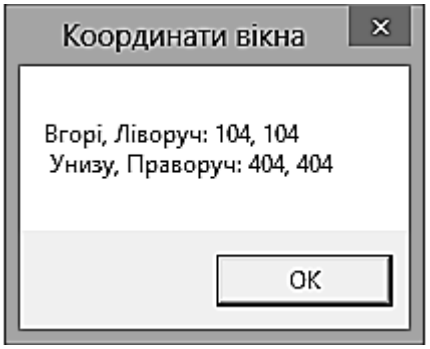
Після запуску програми



Виконана команда «Файл»



Виконана команда «Сервіс»



Виконані команди «Сервіс» / «Координати»

Рис. 7. Результат виконання програми, яка створює основне меню

Завдання.

1. Набрати, відкомпілювати і запустити на виконання приклад програми, який був наведено в розділі «Основні положення» даної лабораторної роботи.

2. Проекспериментуйте з програмою.

Індивідуальна частина.

Модифікувати програму з прикладу 5 таким чином, щоб головне меню мало N пунктів, а кожне спадне меню - по M опцій. Значення N і M, а також найменування відповідних пунктів погодити з викладачем.

При виборі кожного з пунктів меню, що розкривається, повинна бути сформована відповідна програмна заглушка.

Контрольні питання

1. У чому відмінність графічних додатків від консольних?
2. Опишіть структуру найпростішого графічного додатку.
3. Які оператори необхідно використовувати для створення і розміщення на формі кнопки.
4. Як здійснюється обробка подій в Windows-додатку?
5. Перерахуйте можливі варіанти інтерфейсу вікна повідомлень.
6. Опишіть алгоритм створення головного меню Windows- додатку.