

## Лабораторна робота №2. Створення діалогової програми. Windows Forms. Елементи Label, Button, TextBox

**Тема:** Створення діалогової програми. Windows Forms. Елементи Label, Button, TextBox.

**Мета заняття:** навчитись створювати діалогові програми з допомогою Windows Forms мовою C#; навчитись працювати з елементами Label, Button, TextBox Windows Forms у середовищі Microsoft Visual Studio.

### 1. Теоретичні відомості

#### 1.1 Вступ до C # Windows Forms

На рис. 1 показано, як створити новий проект у Microsoft Visual Studio. Для цього в налаштуваннях вибирається C # (#1) та Windows Forms (#2).

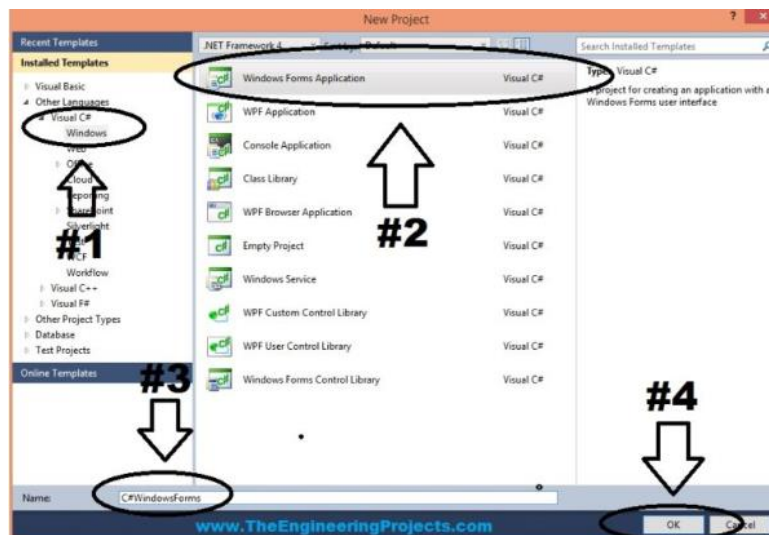


Рисунок 1.1 – Створення проекту з Windows Forms у C#

Після натиснення кнопки ОК (#4), відкриється новий проект, як показано на рис. 1.2.

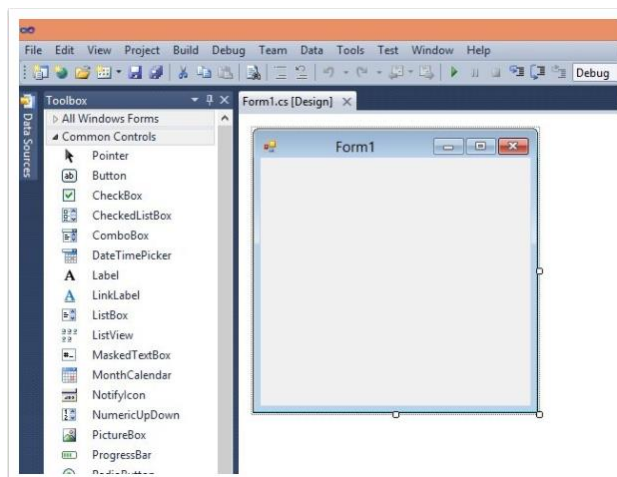


Рисунок 1.2 – Вікно Visual Studio з новим проектом у Windows Forms

Новий проект містить три частини. Центральна частина – це форма з заголовком Form1. Ліворуч – панель інструментів, яка містить усі компоненти, які можна використовувати у цій формі C # Windows Forms. Праворуч – це «Провідник рішень», а під ним – «Панель властивостей».

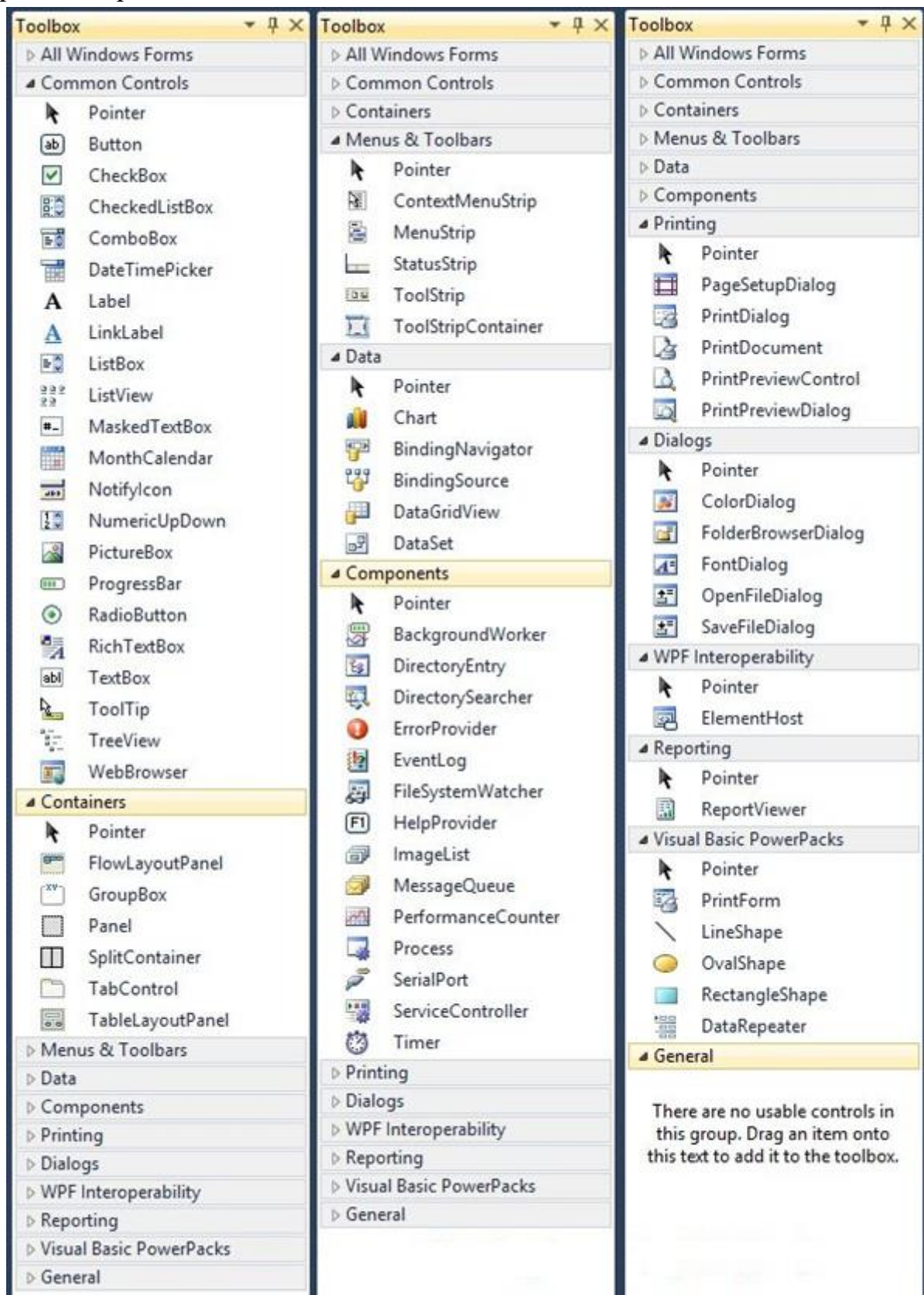


Рисунок 1.3 – Компоненти панелі інструментів C # Windows Forms

Усі елементи керування можна використовувати в C # Windows Forms. Найчастіше використовуються кнопки та текстові поля, які знаходяться у розділі «Загальні елементи керування». На рис. 1.4 наводиться зображення Провідника рішень у формах C # Windows.

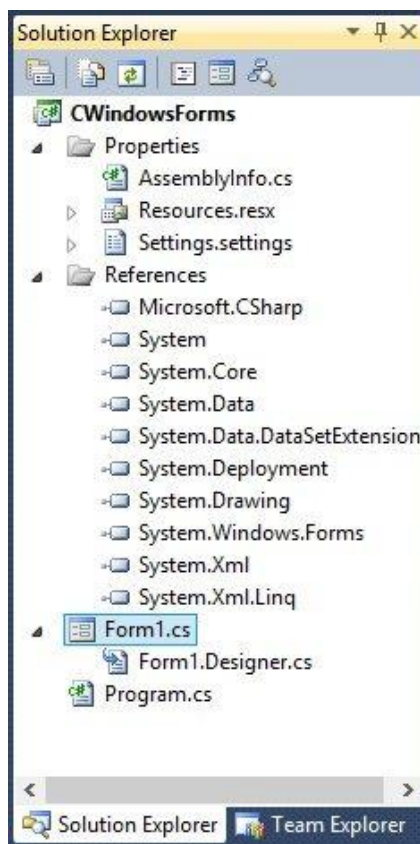


Рисунок 1.4 – Провідник рішень у формах C # Windows

Провідник рішень показує всі файли, які використовуються у вашому C # Windows Forms Project. У папці "Properties" є файл AssemblyInfo.cs, який містить ім'я та авторські права щодо вашого проекту. У розділі "References" містяться всі файли системної бібліотеки. У файлі Form1.Designer.cs знаходиться дизайн Form1. У файлі Program.cs знаходиться код для Form1. Весь код, який буде написаний для проекту C # Windows, буде доданий у файл Program.cs.

На рис. 1.5 наводиться зображення вікна «Властивості» у формах C # Windows. Коли вибирається будь-який компонент у C # Windows Forms, тоді його властивості відкриваються. З цих властивостей можна змінити багато атрибутів вашого проекту.

У атрибуту Text знаходиться назва проекту – Form1. Змінимо її на «Introduction to C # Windows Forms» (рис. 1.6).

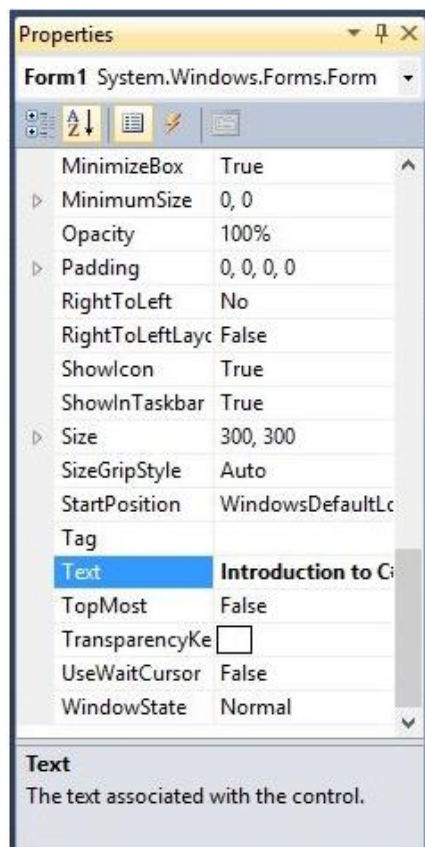


Рисунок 1.5 – Properties panel in C# Windows Forms:

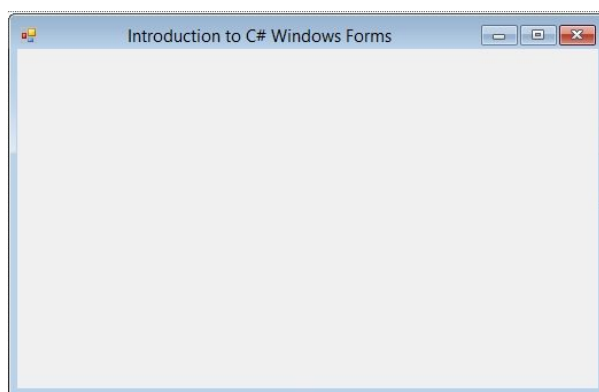


Рисунок 1.6 – Змінений заголовок Windows Forms

Таблиця 1.1. – Основні властивості форми

ВЛАСТИВІСТЬ	ОПИС
Name	Задає ім'я класу <b>Form</b> . Дана властивість задається виключно під час розробки
BackColor	Вказує колір фону форми
Enabled	Вказує чи може форма приймати ввід від користувача. Якщо властивості <b>Enabled</b> задано значення <b>False</b> , усі елементи керування форми також блокуються
ForeColor	Вказує колір переднього плану форми, тобто колір тексту, що виводиться.
FormBorderStyle	Указує вид та поведінку межі та рядка заголовку форми Значення властивості: <b>None</b> - форма не має меж, не може бути мінімізована або розгорнута до максимальних розмірів та в неї немає екранної кнопки керування вікном та кнопки довідки;

ВЛАСТИВІСТЬ	ОПИС
	<ul style="list-style-type: none"> <li>▪ <b>FixedSingle</b> - форма має тонку межу, розміри форми неможна змінювати під час виконання. Форма може бути мінімізована, розгорнута до максимальних розмірів, має кнопку довідку або кнопку керування вікном;</li> <li>▪ <b>Fixed3D</b> - форма має об'ємну межу, розміри форми неможна змінити під час виконання. Форма може бути мінімізована, розгорнута до максимальних розмірів, має кнопку довідки або кнопку керування вікном;</li> <li>▪ <b>FixedDialog</b> - форма має тонку межу, розміри форми неможна змінювати під час виконання. У форми немає екранної кнопки керування вікном, але може бути кнопка довідки, яка визначається іншими властивостями. Форму можна мінімізувати та розгорнути до максимальних розмірів;</li> <li>▪ <b>Sizable</b> - форма має налаштування по замовчуванню, але вони можуть змінюватись користувачем. Форма може бути мінімізована, розгорнута до максимальних розмірів, має кнопку довідки, яка визначається іншими властивостями.</li> <li>▪ <b>FixedToolWindow</b> - форма має тонку межу, розміри форми неможна змінити під час виконання. Форма містить тільки кнопку закриття</li> <li>▪ <b>SizableToolWindow</b> - форма має тонку межу, розміри</li> </ul>
Location	Коли властивості <b>StartPosition</b> задано значення <b>Manual</b> , то властивість указує початкове положення форми відносно верхнього лівого кута екрану
MaximizeBox	Указує, чи є у форми кнопка <b>MaximizeBox</b>
MaximumSize	Встановлює максимальний розмір форми. Якщо задати властивість розмір 0; 0, то у форми не буде верхнього обмеження розміру
MinimizeBox	Указує, чи у форми кнопка <b>MinimizeBox</b>
MinimumSize	Встановлює мінімальний розмір форми, який може задати користувач
Opacity	Встановлює рівень непрозорості або прозорості форми від 0 до 100%. Форма, що має непрозорість 100%, повністю непрозора
Size	Приймає та встановлює початковий розмір форми
StartPosition	Указує на положення форми в момент її першого виведення на екран
Text	Указує заголовок форми
TopMost	Указує, чи завжди форма відображається поверх усіх форм, властивості <b>TopMost</b> яких не задано значення <b>True</b>
Visible	Указує, видима чи форма під час роботи
WindowState	Указує, чи форма є мінімізованою, розгорнутою до максимальних розмірів, або при першій появі їй заданий розмір, що вказаний у властивості <b>Size</b>



## 1.2 Додавання елементів керування C #

Отже, перший C # елемент керування, який будемо використовувати, – це поле Text Box, яке присутнє в категорії **Common Controls** (Загальні елементи керування). Необхідно перетягнути це текстове поле з панелі інструментів до форми Windows, після чого воно буде розміщене у формі Windows, як показано на рис. 1.7.

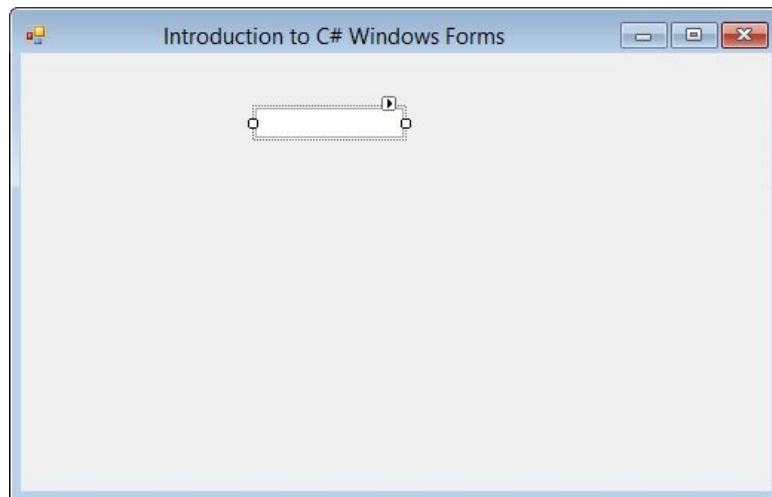


Рисунок 1.7 –Розташування елементу Text Box на Windows Forms

Якщо вибрати це текстове поле, тоді в розділі властивостей відкриються його властивості. З його властивостей ви можете змінити багато його атрибутів. Два основні атрибути у властивостях будь-яких елементів керування C # - це ім'я (Name) та текст (Text).

Ім'я – це ім'я цього елемента управління, яким ми його називаємо, коли пишемо код для цього елемента управління.

Текст – це текст, який з'являється на цьому контролі для перегляду користувачем.

Перетягнемо кнопку з панелі інструментів і розмістимо її на формі Windows, як показано на рис. 1.8.

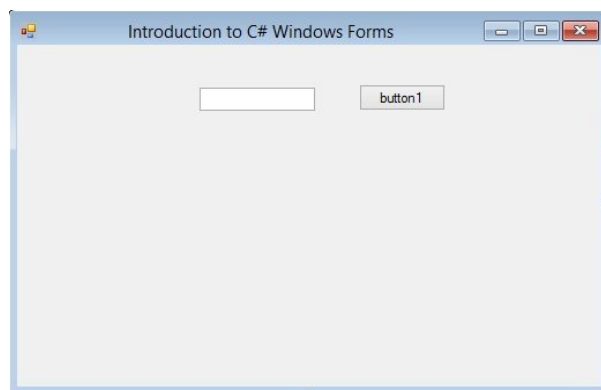


Рисунок 1.8 – Розташування елементу Button на Windows Forms

У розділі "Властивості" для Button змінюємо її текст на «Click Here», як показано на рис. 1.9.

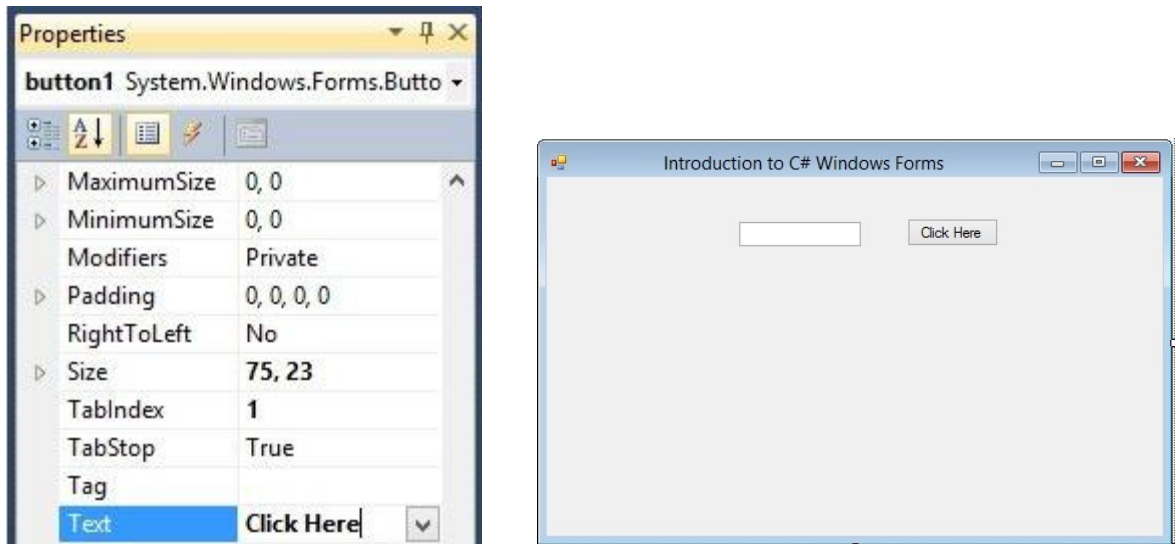


Рисунок 1.9 – Зміна тексту елементу Button на Windows Forms

### 1.3 Робота з кнопкою у формі C # Windows

Створюємо новий Проект за зразком (рис. 1.10). буде виглядати приблизно так:

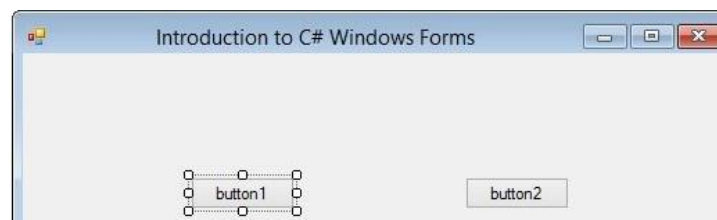


Рисунок 1.10 – Форма C # для Windows з 2 кнопками

Змінюємо текст Button1 на Click Here та текст Button2 на Submit. Також змінюємо назву цих кнопок, тож змінюємо Name першої кнопки на ClickHere, а другої кнопки - Submit.

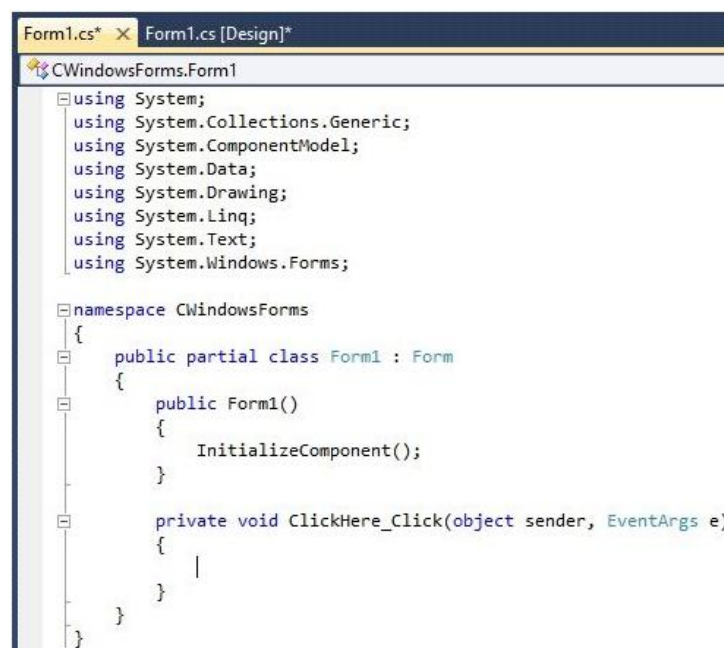


Рисунок 1.11 – Код обробки події Click для кнопки ClickHere

Додамо події Click для кожної з цих кнопок. Подія Click – це подія, яка повинна відбутися при натисканні на кнопку. Двічі клацніть першу кнопку, і вона відкриє код, як показано на рис. 1.11.

У наведеному коді є функція з назвою **ClickHere\_Click**, яка є функцією події клацання для першої кнопки. **ClickHere** - це назва кнопки, яку змінили вище, а **\_Click** - подія. Простими словами, ця подія відбудеться після натискання кнопки ClickHere.

У фігурних дужках {} вставляється код для події клацання цієї кнопки. Додаємо код у ці дужки.

```
MessageBox.Show ("It's Click Here Button");
```

Коли буде натиснута кнопка «Click Here», відкриється вікно із повідомленням (рис. 1.12)

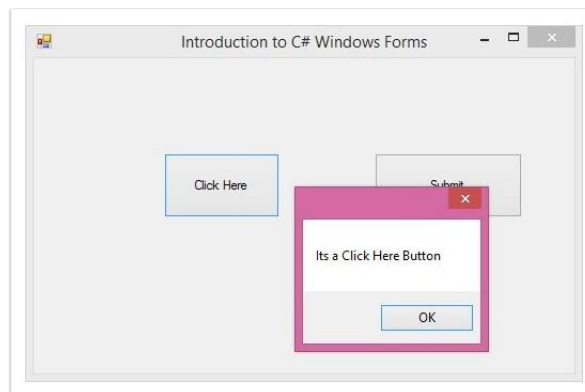


Рисунок 1.12 – Вікно повідомлення, що появляється після натиснення кнопка «Click Here»

Аналогічно створюємо обробку події для кнопки «Submit». Результат наведений на рис. 1.13

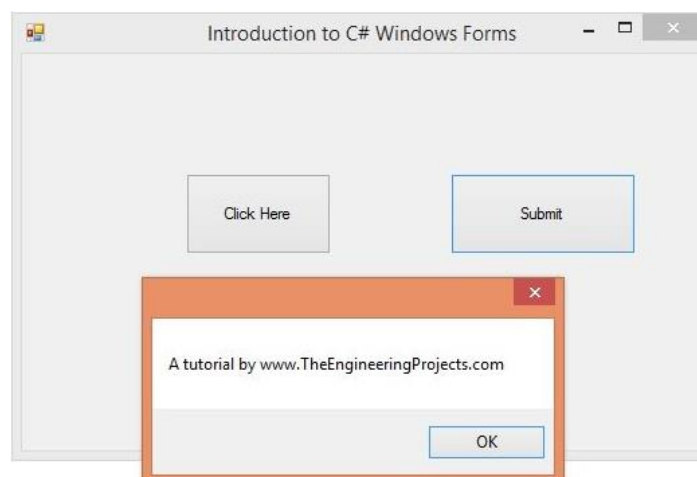


Рисунок 1.13 – Вікно повідомлення, що появляється після натиснення кнопка «Submit»



## 1.4 Керування кнопками C #

Якщо необхідно змінити текст кнопки під час виконання або динамічно, тоді використовується код

```
button1.Text = "TEP Button Example Text";
```

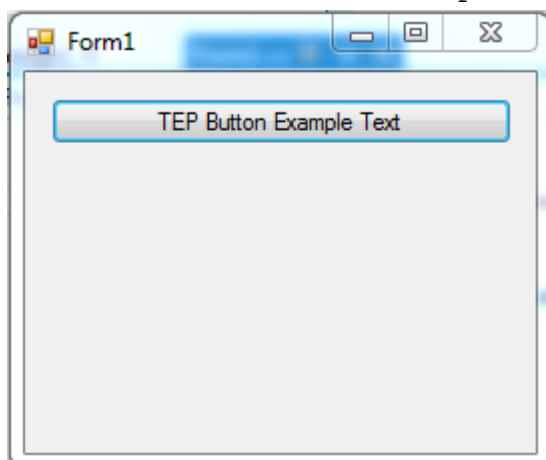


Рисунок 1.14 – Зміна тексту кнопки

Якщо потрібно показати зображення всередині кнопки, тоді використовується властивість **Image**. **FromFile** використовується для встановлення шляху зображення, яке ви хочете встановити.

```
button1.Image = Image.FromFile("C:\\Users\\Jade\\Pictures\\brownImage.jpg");
```

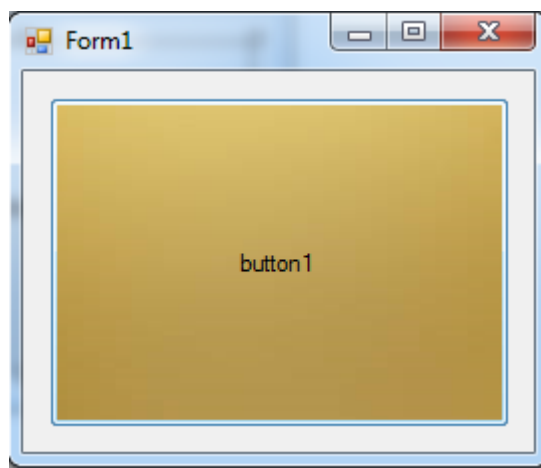


Рисунок 1.15 – Результат встановлення зображення brownImage.jpg для кнопки

Для встановлення кольору тла кнопки, то використовують властивість **BackColor**.

Можна використовувати вбудовані атрибути кольору для встановлення кольорів.

```
button1.BackColor = Color.Aqua;
```

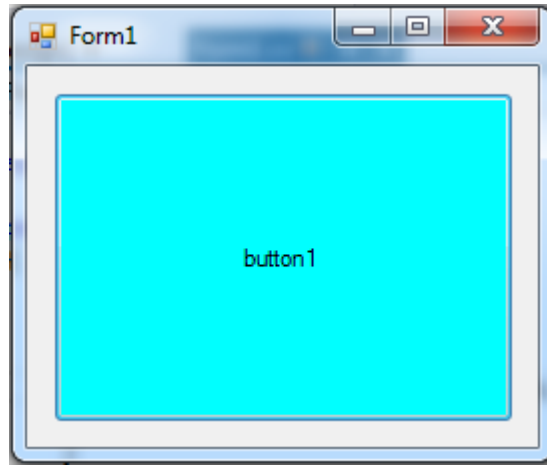


Рисунок 1.16 – Результат встановлення фону кнопки

Властивість **ForeColor** використовується для встановлення кольору тексту. Приклад коду, який змінить колір переднього плану на чорний, та встановить колір тексту білий:

```
button1.ForeColor = Color.White;  
button1.BackColor = Color.Black;
```

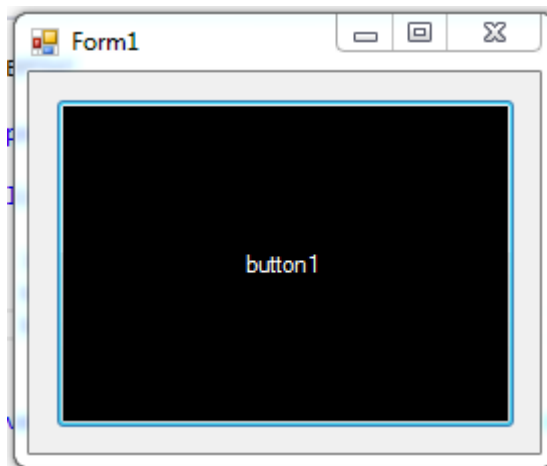


Рисунок 1.17 – Результат встановлення властивостей кольору фону та тексту кнопки

Властивість **Font** дозволяє змінити розмір тексту кнопки. Створюємо **new Font object** і передаємо **button.font.fontfamily** і після коми ставиться розмір тексту кнопки.

```
button1.Font = new Font(button1.Font.FontFamily, 33);
```

або

```
int newSize = 33;  
button1.ForeColor = Color.White;  
button1.BackColor = Color.Black;  
button1.Font = new Font(button1.Font.FontFamily, newSize);
```

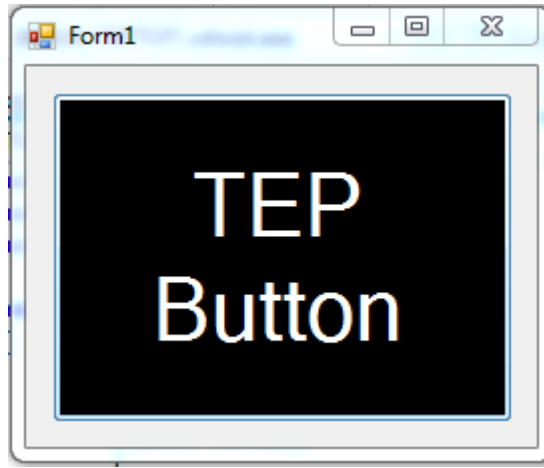


Рисунок 1.18 – Результат встановлення розміру тексту кнопки

Є кілька подій, які можемо використовувати для кнопок. Події - це деякі умовні перевірки на виконання конкретних функцій, коли користувач виконує щось конкретне. Перелік поширених подій, які використовуються для кнопки C#.

- Click Event
- Text Changed Event
- MouseHover Event
- MouseLeave Event

Подія Click відбудеться, коли кінцевий користувач один раз натисне кнопку. Для використання події **Click** для обробки цієї функції після назви кнопки необхідно написати **\_Click**. Наприклад для кнопки з ім'ям **button1** доведеться створити метод з іменем **button1\_Click**:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("C# Button Click Event Executed");
}
```

Подія Text Changed відбувається при зміні тексту кнопки. Це вбудований метод з ім'ям **\_TextChanged**. У наступному коді є подія **button1\_TextChanged**, яка генерує спливаюче повідомлення при зміні тексту кнопки. Подія **Button1\_Click** використовується для зміни тексту, коли користувач натисне кнопку. Логіка – це коли користувач натисне кнопку, текст кнопки буде змінено і **button1\_TextChanged** виконується.

```
private void button1_Click(object sender, EventArgs e)
{
    button1.Text = "New Text";
}
private void button1_TextChanged(object sender, EventArgs e)
{
    MessageBox.Show("C# Button TextChanged Event");
}
```

Подія **MouseHover** відбувається, коли користувач наводить курсор миші на кнопку. Це вбудована подія з ім'ям **\_MouseHover** після імені кнопки. У коді використовується типова назва кнопки. Коли користувач наведе курсор миші на кнопку `button1`, він створить спливаюче повідомлення.

```
private void button1_MouseHover(object sender, EventArgs e)
{
    MessageBox.Show("C# Button MouseHover Event");
}
```

Подія **MouseLeave** відбувається, коли користувач залишить кнопку або перемістить курсор з меж кнопки. У наступному коді створена подія **\_MouseLeave** із назвою кнопки за замовчуванням. Коли користувач забере курсор миші з кнопки генерує спливаюче повідомлення.

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

namespace TEPTUT
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void button1_MouseLeave(object sender, EventArgs e)
        {
            MessageBox.Show("C# Button MouseLeave Event");
        }
    }
}
```

### *1.5 C# керування мітками*

**Label** (мітка) також використовується для відображення описового тексту, такого як примітки та попередження про використання програмного забезпечення. Щоб додати мітку з панелі інструментів її потрібно перетягнути з вкладки дизайнера форми.

`Label1` – це ім'я за замовчуванням першої мітки, яку використовують у програмі. Можна регулювати розмір форми, розтягуючи координати. Можна змінити текст на правій панелі під міткою властивості. Також можна змінити текст, використовуючи наведений код (рис. 1.19):

```

namespace Lab1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = "ТЕР C# Label Control Example Text";
        }
    }
}

```

Рисунок 1.19 – Зміна тексту мітки Label1 при завантаженні форми

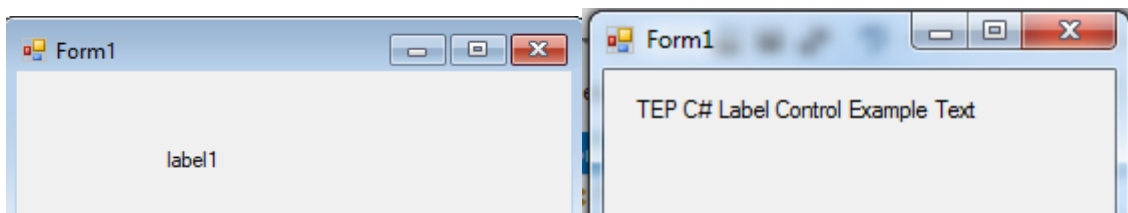


Рисунок 1.20 – Результат зміни тексту Label1 при завантаженні форми

Властивість `BackColor` використовується для зміни кольору фону мітки, наприклад (рис. 1.21):

```
label1.BackColor = Color.Aqua;
```

Ось результат коду:

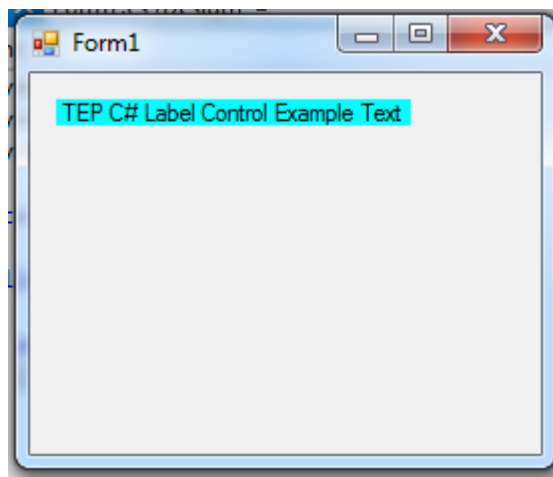


Рисунок 1.21 – Встановлення фону кольору Label1

Властивість `ForeColor` використовується для зміни кольору тексту, наприклад (рис. 1.22):

```
label1.ForeColor = Color.BlueViolet;
```



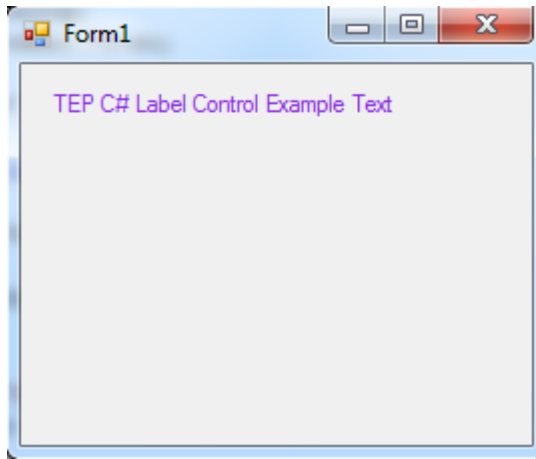


Рисунок 1.22 – Встановлення кольору тексту Label1

Для встановлення зображення як фон мітки, то потрібно використовувати такий код:

```
label1.Image = Image.FromFile("C:\\Pictures\\brownImage.jpg");
```

Метод `Image.FromFile` використовується для вказання шляху зображення, яке ви хочете встановити. У подвійні лапки пишуть шлях і використовують "\\" для розділення каталогів. Фрагмент коду, що змінює властивості Label1, та результат його виконання наведений на рис. 1.23.

```
private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "TEP C# Label Control Example Text";
    //label1.BackColor = Color.Aqua;
    label1.ForeColor = Color.BlueViolet;
    //label1.Image = Image.FromFile("C:\\Users\\Public\\Pictures\\img016.jpg");
}
```

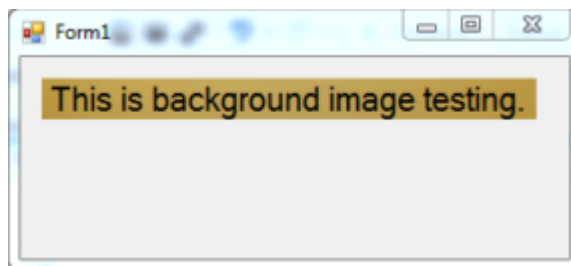


Рисунок 1.23 – Результат виконання фрагменту коду, що змінює властивості Label1

Є кілька подій, які можна використовувати для міток (рис. 1.24):

- Click Event
- Double Click Event
- Text Changed Event
- MouseHover Event
- MouseLeave Event

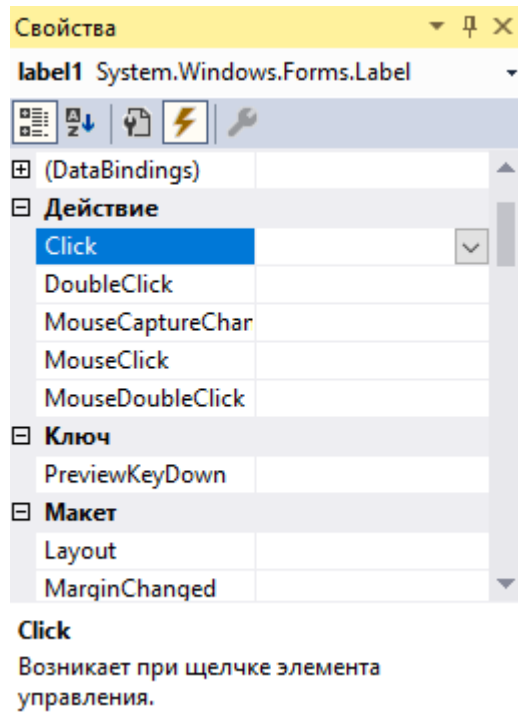


Рисунок 1.24 – Події для елемента Label1

Подія Click виконуватися, коли користувач натисне на мітку.

```
private void label1_Click(object sender, EventArgs e)
{
    label1.Text = "Text Changed";
}
```

Подія Double Click відбудеться, коли користувач двічі натисне на label.

```
private void label1_Click(object sender, EventArgs e)
{
    label1.Text = "Text Changed";
}

private void label1_DoubleClick(object sender, EventArgs e)
{
    label1.Text = "Text New Changed";
}
```

Подія Text Changed (рис. 1.25) відбудеться при зміні тексту мітки. Розглянемо це на прикладі. Використовуються дві мітки. Тепер логіка полягає в тому, що текст однієї мітки змінюється, тоді інший текст іншої мітки повинен змінюватися автоматично.

Для першої мітки також використовується **\_ClickEvent** та **\_TextChanged Event**.

Отже, тепер, коли користувач натисне на мітку, текст мітки зміниться через **\_Click Event**. Тепер, коли текст буде змінено, **відбудеться** **\_TextChanged Event** і текст другої мітки також зміниться.

Ось код, який слід спробувати.

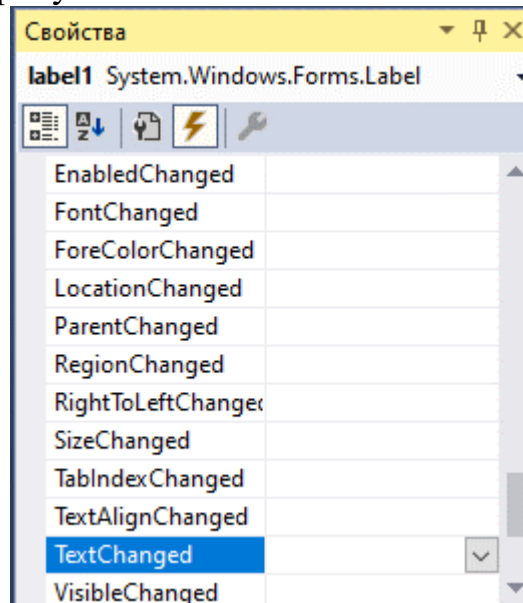


Рисунок 1.25 – Вибір події Text Changed для Label1

```
private void Form1_Load(object sender, EventArgs e)
{
    //label1.Text = "ТЕР C# Label Control Example Text";
    //label1.BackColor = Color.Aqua;
    label1.ForeColor = Color.BlueViolet;
    //label1.Image = Image.FromFile("C:\\Users\\Public\\Pictures\\img016.jpg");
}

private void label1_Click(object sender, EventArgs e)
{
    label1.Text = "Text Changed";
}

private void label1_DoubleClick(object sender, EventArgs e)
{
    label1.Text = "Text New Changed";
}

private void label1_TextChanged(object sender, EventArgs e)
{
    label2.Text = "2nd Text Changed";
}
}
```

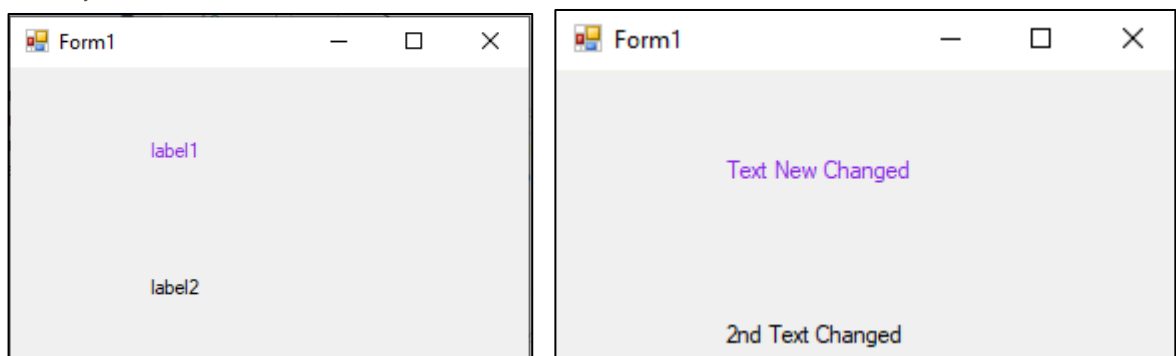


Рисунок 1.26 – Результат події Click та Text Changed для Label1 та Label2

Подія `MouseHover` відбудеться, коли користувач наведе курсор миші на мітку. Припустимо, ви хочете змінити текст, коли користувач наведе курсор миші на мітку. Ось код для цього:

```
private void label1_MouseHover(object sender, EventArgs e)
{
    label1.ForeColor = Color.Red;
    label1.Text = "Text Too Changed";
}
```

Подія `MouseLeave` відбудеться, коли курсор покине мітку. Припустимо, ви хочете змінити текст мітки, коли курсор миші покине мітку. Ось код, який буде виконувати це:

```
private void label1_MouseLeave(object sender, EventArgs e)
{
    label1.Text = "again Text Changed";
}
```

## 2. Практичне завдання



У процесі виконання завдань лабораторної роботи необхідно формувати набори тестових даних для перевірки правильності виконання програмного коду. Створений код і результати перевірки його роботи потрібно помістити у звіт. Тестувати роботу програми рекомендується після додання чи зміни кожного оператора виведення.

**Завдання 1. Створення проект для визначення віку за даними навчання на відповідному курсі університету.**

1. Для цього на форму додати 2 кнопки з текстом «ClickHere» та «OK», поле вводу `TextBox`, `Label 1`, `Label2` (рис. 2.1).

2. Задати властивості, щоб при запуску форми елементи `Label2`, `TextBox`, `Button2` були приховані. За замовчуванням значення `TextBox` дорівнює 0.

3. Задати колір тексту `Label 1` – `DarkBlue`, `Label 2` – `DarkOrange`; розмір шрифту 14. `Label 1` повинна виводити текст «На якому курсі Ви навчаєтесь? Введіть число після натиснення кнопки» у 2 рядки.

4. За натисненням кнопки «ClickHere» з'являється поле `TextBox` та кнопка «OK».

5. Після введення числа у поле `TextBox` та натиснення кнопки «OK» виводиться вікно з повідомленням про Ваш вік. Для цього попередньо потрібно зчитати значення числа, що введено у поле `TextBox` та додати до нього 18.

6. Після закриття вікна повідомлення приховати `Label 1` та вивести у `Label2` повідомлення про Ваш вік (рис. 2.2).

Form1

Click Here

label1

label2

0

OK

Form1

Click Here

На якому курсі Ви навчаєтесь?  
Введіть число після натиснення кнопки

Form1

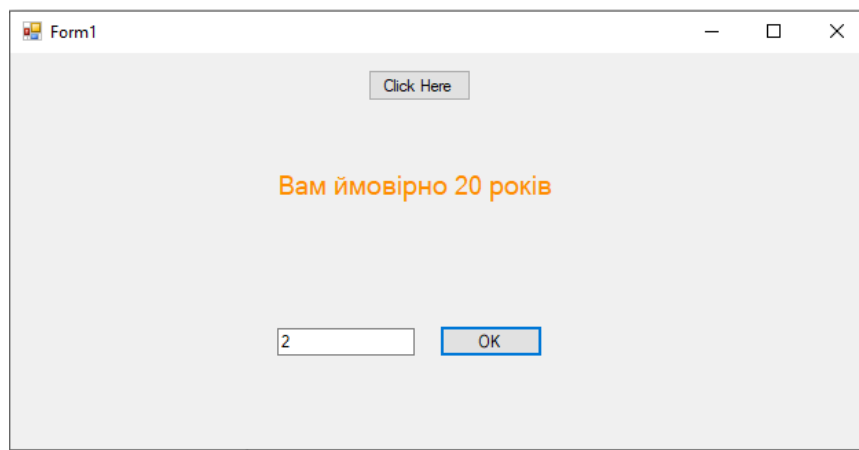
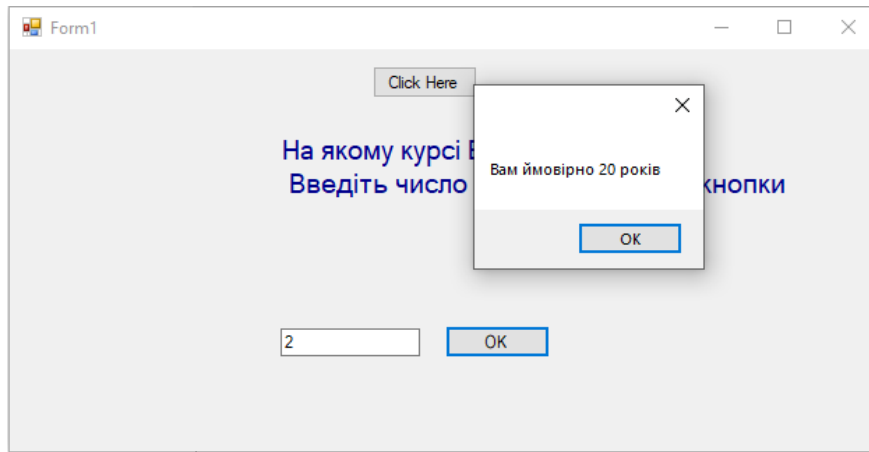
Click Here

На якому курсі Ви навчаєтесь?  
Введіть число після натиснення кнопки

2

OK





```
namespace Lab2
{
    public partial class Form1 : Form
    {
        int c;
        public Form1()
        {
            InitializeComponent();
            textBox1.Visible = false;
            label2.Visible = false;
            button2.Visible = false;
            label2.Font = new Font(label1.Font.FontFamily, 14);
            label2.ForeColor = Color.DarkOrange;
            label1.Font = new Font(label1.Font.FontFamily, 14);
            label1.ForeColor = Color.DarkBlue;
            label1.Text = "На якому курсі Ви навчаєтесь?\n Введіть число після натиснення кнопки";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            textBox1.Visible = true;
            button2.Visible = true;
        }
        private void button2_Click(object sender, EventArgs e)
        {
            c = Convert.ToInt16(textBox1.Text);
            MessageBox.Show("Вам ймовірно " + Convert.ToString(18+c) + " років");
            label2.Text = "Вам ймовірно " + Convert.ToString(18 + c) + " років";
            label1.Visible = false;
            label2.Visible = true;
        }
    }
}
```

## Завдання 2. Скласти програму «Стрибаюча кнопка».

1. Створіть новий проект. Помістіть на форму компонент *Label* і кнопку *Спіймай мене!*.
2. Створіть для кнопки обробник події *Click*. Опишіть змінну *k* для збереження кількості клацань кнопки:

*int k;*

У код процедури запишіть оператори:

*k = k + 1;*

*Label1.Text = Convert.ToString(k);*

3. Додайте на форму компонент *Timer* (Таймер). Властивість таймера *Interval* встановіть 1000 (1000 мс = 1 с). Властивість *Enable* переведіть у положення *True*.
4. Створіть обробник події «*Tick*», в якому запрограмуйте випадкову зміну властивостей *Left* і *Top* кнопки.

```
private void timer1_Tick(object sender, EventArgs e)
{
    Random rand = new Random();
    button1.Left = rand.Next(Width-button1.Width);
    button1.Top = rand.Next(Height-button1.Height);
}
```

Кнопка не має «вистрибнути» за межі форми, тому потрібно підключити розміри робочої поверхні форми *Width* і *Height*.

5. Запустіть програму на виконання. Чи вдається спіймати кнопку? Якщо ні, зупиніть виконання і збільшіть значення властивості таймера *Interval*.
6. Додайте на форму кнопку *Button2*, в програмному коді якої запрограмуйте збільшення значення *Interval*:  
*Timer1.Interval = Timer1.Interval + 100;*
7. Додайте на форму кнопку *Button3*, призначену для зменшення значення *Interval*. Перевірте дію кнопок.

## 3. Контрольні запитання

1. У чому полягає принцип швидкої розробки застосувань?
2. На які частини умовно розділяють вікно середовища розробки MS Visual Studio?
3. Що таке проект в MS Visual Studio?
4. Як створити проект з користувацькими вікнами (формами)?
5. Які основні файли проекту?
6. Як створити і переглянути процедури обробки подій форми і окремого елемента керування?
7. В якому файлі міститься головна програма у віконному проекті?

## Література

1. Троелсен Э. С# и платформа.Net / Пер. с англ. – СПб.: Питер, 2007. – 796 с.
2. Троелсен Э. Язык программирования С# 2005 и платформа .Net / Пер. с англ. – М.: Изд. дом "Вильямс", 2007. – 1168 с.
3. Шилдт Г. Полный справочник по С#. / Пер. с англ. – М.: Изд. дом "Вильямс", 2004. – 752 с.
4. Нейгел К. С# 2005 для профессионалов / К. Нейген, Б. Ивсен, Дж. Глинн; / [Пер. с англ. – М.: Изд. дом "Вильямс", 2006. – 1376 с.
5. [www.c-sharpcorner.com](http://www.c-sharpcorner.com) – матеріали з С#.
6. [www.functionx.com](http://www.functionx.com) – матеріали з С# та інших мов програмування.
7. [www.rsdn.ru](http://www.rsdn.ru) – матеріали з С# й інших мов програмування.