Лабораторна робота № 2

Тема: Робота з контейнерами та компонентами Windows Forms.
Мета роботи: отримати практичні навики використання компонентів під час розробки додатків Windows Forms.

Теоретичні відомості

Для організації елементів управління у зв'язані групи існують спеціальні елементи – контейнери. Наприклад, *Panel, FlowLayoutPanel, SplitContainer, GroupBox*. Форму також можна віднести до контейнерів. Використання контейнерів полегшує управління елементами, а також надає формі певний візуальний стиль.

Всі контейнери мають властивість `Controls`, яке містить всі елементи даного контейнера. Коли переносять який-небудь елемент з панелі інструментів на контейнер, наприклад, кнопку, вона автоматично додається в цю колекцію даного контейнера. В цю колекцію також можна додати елемент керування динамічно за допомогою коду.

1.1. Динамічне додавання елементів

Додамо на форму кнопку динамічно. Спочатку додамо подію завантаження форми, в якій буде створюватися новий елемент управління. Це можна зробити або за допомогою коду або за допомогою перетягування елементів з *Панелі Інструментів*. Однак останній спосіб досить обмежений, оскільки дуже часто потрібно динамічно створювати (видаляти) елементи на формі.

Для динамічного додавання елементів створимо обробник події завантаження форми в файлі коду:

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

Тепер додамо в нього код додавання кнопки на форму:

```
private void Form1_Load(object sender, EventArgs e)
{
    Button helloButton = new Button();
    helloButton.BackColor = Color.LightGray;
    helloButton.ForeColor = Color.DarkGray;
    helloButton.Location = new Point(10, 10);
    helloButton.Text = "Привіт";
    this.Controls.Add(helloButton);
}
```

Спочатку створюємо кнопку і задаємо її властивості. Потім, використовуючи метод `Controls.Add`, додаємо її в колекцію елементів форми. Якщо це не зробити, то кнопка не буде видима, оскільки в цьому випадку для нашої форми її просто не буде існувати.

За допомогою методу `Controls.Remove()` можна видалити раніше доданий елемент з форми:

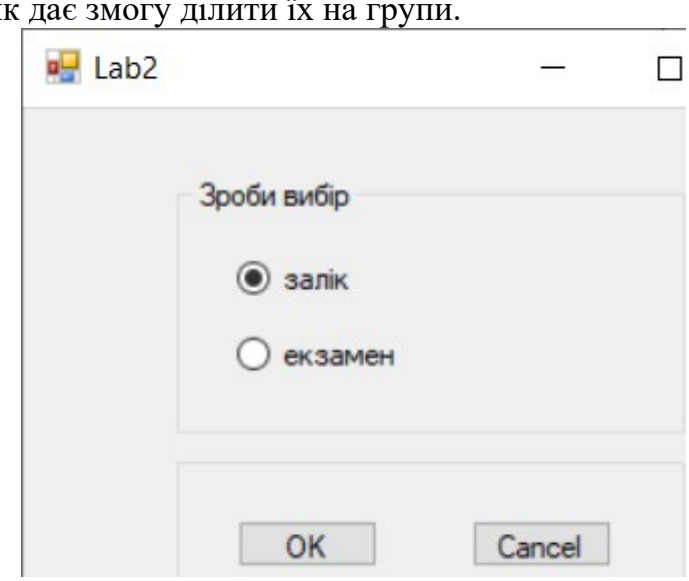
```
this.Controls.Remove(helloButton);
```

Хоча в даному випадку як контейнер використовується форма, але при додаванні і видаленні елементів з будь-якого іншого контейнера, наприклад, `GroupBox`, будуть використовуватись ті ж методи.

1.2. Елементи GroupBox, Panel і FlowLayoutPanel

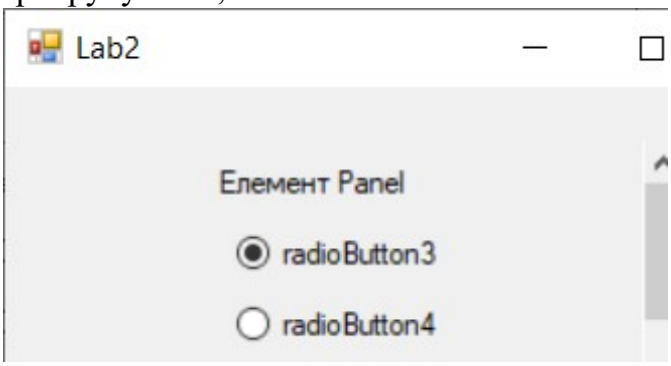
`GroupBox` – спеціальний контейнер, обмежений від решти форми межею. Він має заголовок, який задається властивістю `Text`. Щоб зробити `GroupBox` без заголовка, як значення властивості `Text` просто встановлюється порожній рядок.

Цей елемент часто використовується для групування перемикачів – елементів `RadioButton`, так як дає змогу ділити їх на групи.



Елемент `Panel` відображається як панель і також, як і `GroupBox`, об'єднує елементи в групи. Вона може візуально зливатися з іншою формою, якщо вона має те ж значення кольору фону у властивості `BackColor`, що і форма. Щоб її виділити, крім кольору, за допомогою властивості `BorderStyle`, яка за замовчуванням має значення `None`, тобто відсутність границь, можна задати межі елемента.

Також якщо панель має багато елементів, які виходять за її межі, можна зробити панель, що прокручується, встановивши її властивість `AutoScroll` в `True`.



Компоненти `GroupBox` і `Panel`, як і форма, мають колекції елементів, які також можна динамічно додавати в ці контейнери елементи. Наприклад, на формі є елемент `GroupBox`, який має ім'я `groupBox1`:

```
private void Form1_Load(object sender, EventArgs e)
{
    Button helloButton = new Button();
    helloButton.BackColor = Color.LightGray;
    helloButton.ForeColor = Color.Red;
    helloButton.Location = new Point(30, 30);
    helloButton.Text = "Привіт";
    groupBox1.Controls.Add(helloButton);
}
```

Для вказівки розташування елемента в контейнері використовується структура `Point: new Point(30, 30);`, в конструкторі якої передаємо розміщення по осях `X` та `Y`. Ці координати встановлюються щодо лівого верхнього кута контейнера, в даному випадку елемента `GroupBox`.

При цьому треба врахувати, що контейнером верхнього рівня є форма, а елемент `groupBox1` сам знаходиться в колекції елементів форми. І при бажанні можна його видалити:

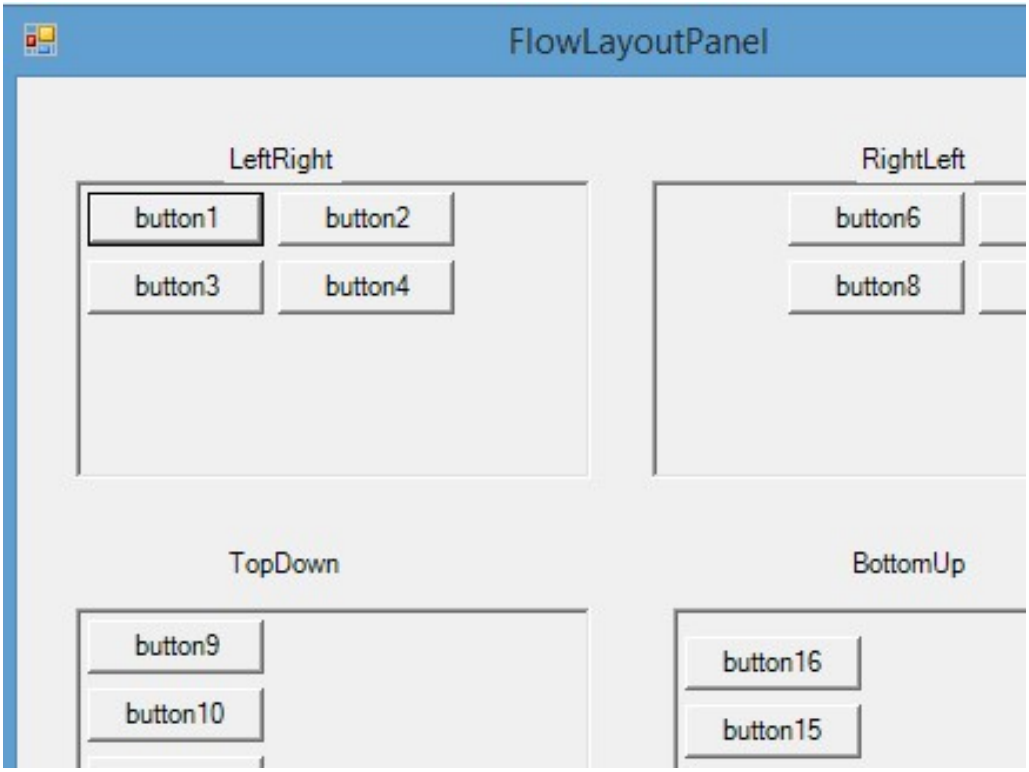
```
this.Controls.Remove(groupBox1);
```

1.2.1. FlowLayoutPanel

Елемент `FlowLayoutPanel` успадкований від класу `Panel`, і тому наслідує всі його властивості. Однак при цьому має додаткову функціональність. Цей елемент дає змогу змінювати позиціонування і компоновку дочірніх елементів при зміні розмірів форми під час виконання програми.

Властивість елемента `FlowDirection` дає змогу задати напрямок, в якому спрямовані дочірні елементи. За замовчуванням має значення `LeftToRight`, тобто елементи будуть розташовуватися починаючи від лівого верхнього краю. Наступні елементи будуть йти вправо. Це властивість також може отримувати такі значення:

- `RightToLeft` - елементи розташовуються від правого верхнього кута в лівий бік;
- `TopDown` - елементи розташовуються від лівого верхнього кута і йдуть вниз;
- `BottomUp` - елементи розташовуються від лівого нижнього кута і йдуть вгору.

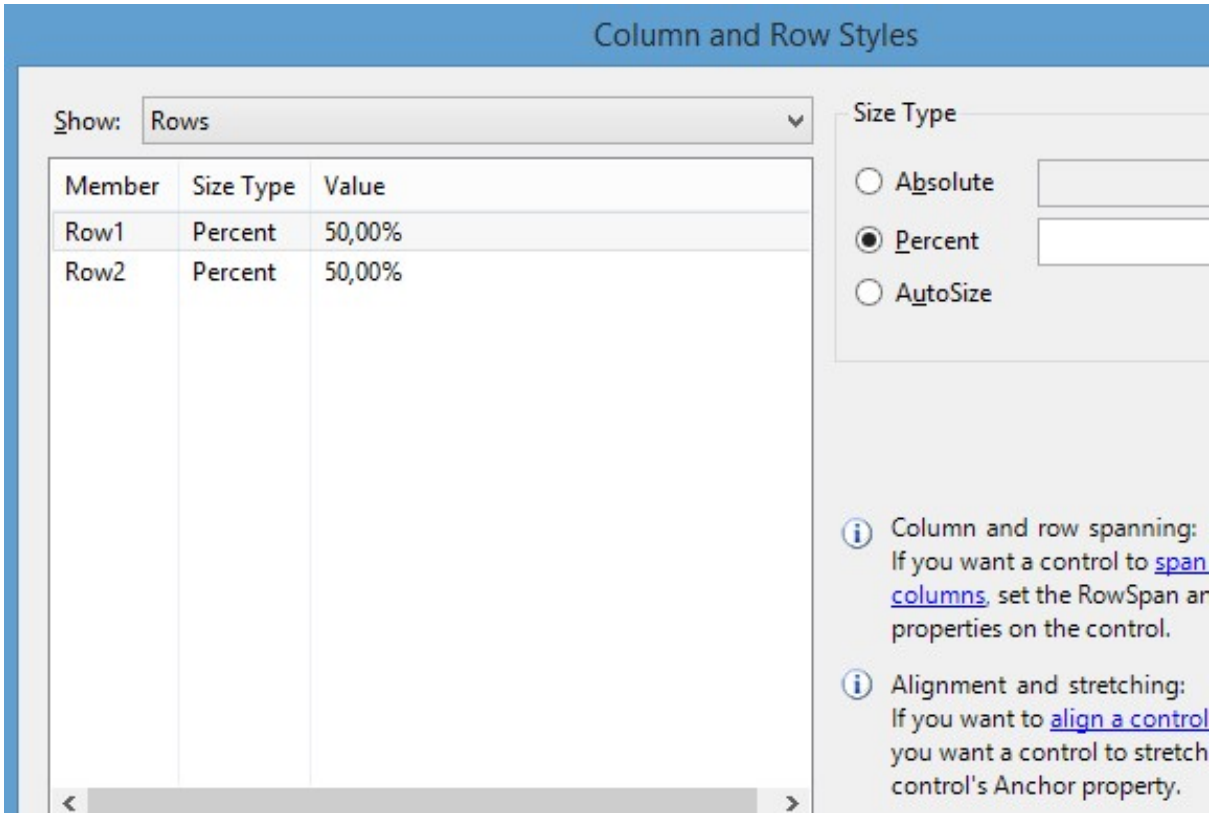


При розташуванні елементів важливу роль відіграє властивість `WrapContents`. За замовчуванням вона має значення `True`. Це дає змогу переносити елементи, які не поміщаються в `FlowLayoutPanel`, у новий рядок або у новий стовпець. Якщо вона має значення `False`, то елементи не переносяться, а до контейнера просто додаються смуги прокрутки, якщо властивість `AutoScroll` дорівнює `True`.

1.3. TableLayoutPanel

Елемент `TableLayoutPanel` також перевизначає панель і має в своєму розпорядженні дочірні елементи управління у вигляді таблиці, де для кожного елемента є своя комірка. Якщо потрібно розташувати в клітинку більше одного елемента, то в цю комірку додається інший компонент `TableLayoutPanel`, в який потім вкладаються інші елементи.

Щоб задати необхідну кількість рядків стовпців таблиці, використовують властивості `Rows` і `Columns` відповідно. Вибравши один з цих пунктів у вікні `Properties` (Властивості), відобразиться наступне вікно для налаштування стовпців і рядків:



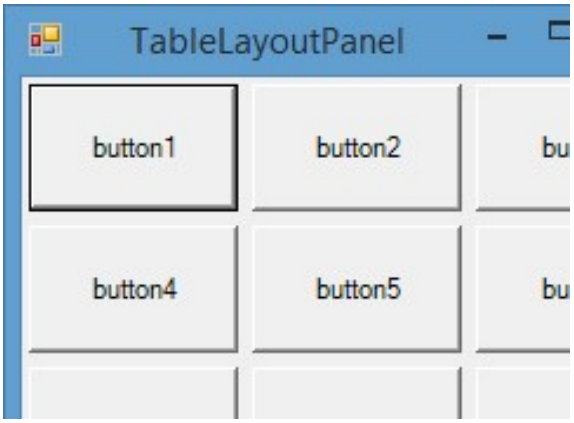
В полі `Size Type` можна вказати розмір стовпців/рядків. Доступні три можливі варіанти:

- `Absolute`: задається абсолютна величина для рядків або стовпців в пікселях;
- `Percent`: задається відносний розмір у відсотках. Якщо необхідно створити гнучкий дизайн форми, щоб її рядки і стовпці, а також елементи управління в комірках таблиці автоматично масштабувались при зміні розмірів форми, то потрібно використовувати саме цю опцію;
- `AutoSize`: висота рядків і ширина стовпців задається автоматично в залежності від розміру найбільшої в рядку або стовпці комірки.

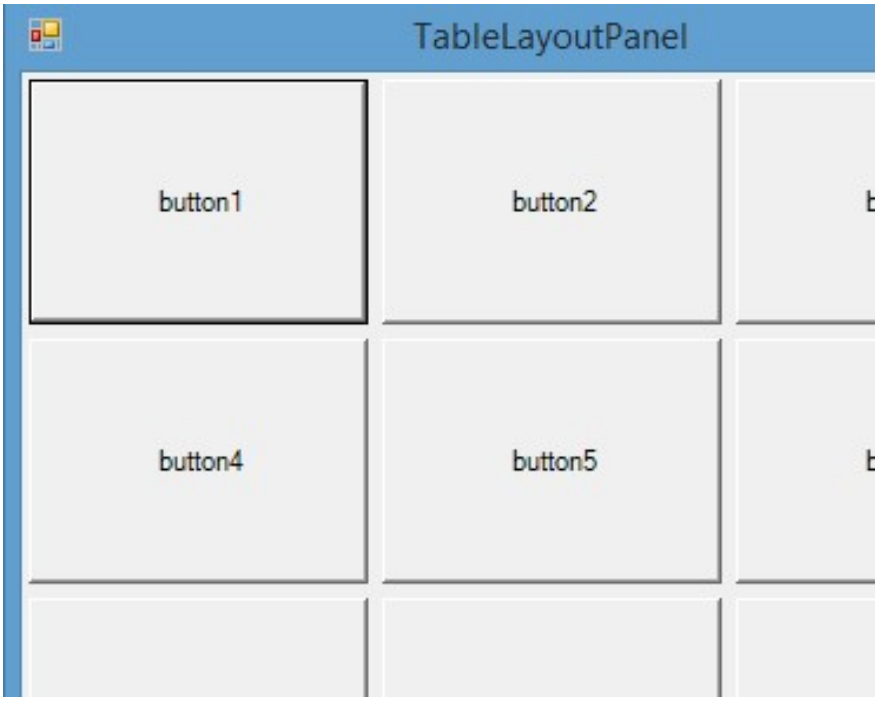
Також можна комбінувати ці значення, наприклад, один стовпець може бути фіксованим з абсолютною шириною, а інші стовпці можуть мати ширину у відсотках.

У цьому діалоговому вікні також можна додати або видалити рядки і стовпці. У той же час графічний дизайнер в *Visual Studio* не завжди відразу відображає зміни в таблиці – додавання або видалення рядків і стовпців, зміну їх розмірів, тому, якщо змін на формі не відно, треба її закрити і потім відкрити в графічному дизайнері знову.

Отже, наприклад, у є три стовпці і три рядки розмір яких однаковий - 33.33%. У кожну клітинку таблиці додано кнопку, у якій встановлено властивість `Dock=Fill`.



Якщо змінити розміри форми, то автоматично змасштабуються і рядки і стовпці разом із вставленими в них кнопками:



Це досить зручно для створення масштабованих інтерфейсів.

У коді значення стовпців і рядків можна змінювати динамічно. Причому всі стовпці представлені типом `ColumnStyle`, а рядки – типом `RowStyle`:

```
tableLayoutPanel1.RowStyles[0].SizeType = SizeType.Percent;
tableLayoutPanel1.RowStyles[0].Height = 40;

tableLayoutPanel1.ColumnStyles[0].SizeType = SizeType.Absolute;
tableLayoutPanel1.ColumnStyles[0].Width = 50;
```

Щоб установити тривалість в `ColumnStyle` і `RowStyle` визначено властивість `SizeType`, яка приймає одне із значень переліку `SizeType`

Додавання елемента в контейнер `TableLayoutPanel` має свої особливості. Додамо його в наступну вільну комірку або явно вказавши елемент таблиці:

```
Button saveButton = new Button();
// додаємо кнопку в наступну вільну комірку
```

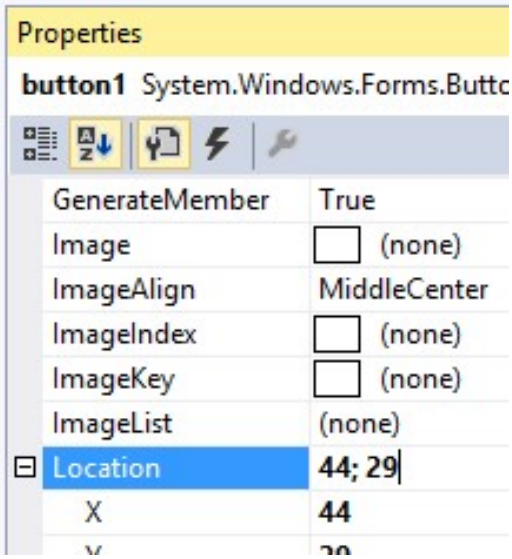
```
tableLayoutPanel1.Controls.Add(saveButton);
// додаємо кнопку в комірку (2,2)
tableLayoutPanel1.Controls.Add(saveButton, 2, 2);
```

В даному випадку додаємо кнопку в комірку, утворену на перетині третього стовпця і третього рядка. Якщо немає стільки рядків і стовпців, то система автоматично вибере потрібну комірку для додавання.

1.4. Розміри елементів і їх позиціонування в контейнері

1.4.1. Позиціонування

Для кожного елемента управління можна визначити властивість `Location`, яка задає координати верхнього лівого кута елемента щодо контейнера. При перенесенні елемента з панелі інструментів на форму ця властивість задається автоматично. Однак потім у вікні *Властивостей* можна вручну виправити координати розміщення елемента:

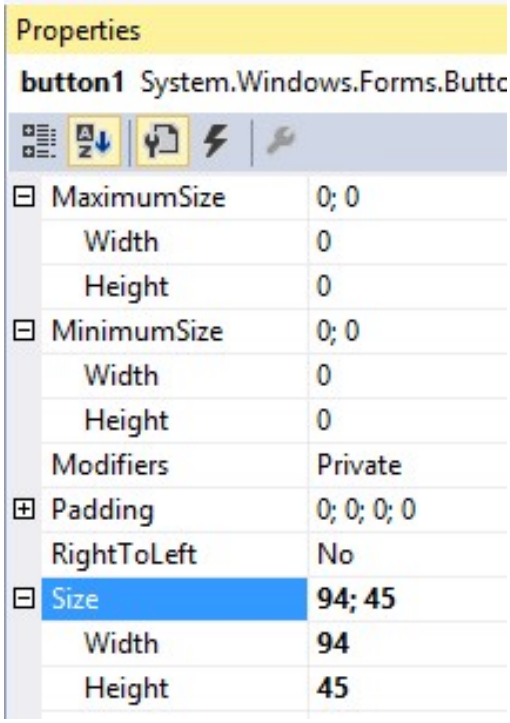


Також можна встановити позицію елемента в коді:

```
private void Form1_Load(object sender, EventArgs e)
{
    button1.Location = new Point(50, 50);
}
```

1.4.2. Встановлення розмірів

За допомогою властивості `Size` можна задати розміри елемента:



Додаткові властивості `MaximumSize` і `MinimumSize` дають змогу обмежити мінімальний і максимальний розміри.

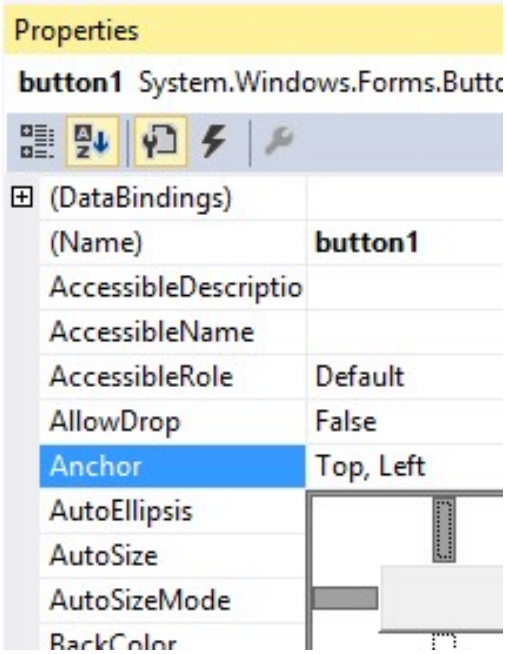
Встановлення властивостей в коді:

```
button1.Size = new Size { Width = 50, Height = 25 };
// окреме встановлення властивостей
button1.Width = 100;
button1.Height = 35;
```

1.4.3. Властивість Anchor

Додаткові можливості по позиціонуванню елемента дають можливість визначити властивість `Anchor`, яка визначає відстань між однією зі сторін елемента і стороною контейнера. Якщо при роботі з контейнером будемо його розтягувати, то разом з ним буде розтягуватися і вкладений елемент.

Автоматично у кожного елемента елемента, що додається, ця властивість дорівнює `Top`, `Left`:



Це означає, що при розтягуванні форми вліво або вгору, елемент збереже відстань від лівої і верхньої межі елемента до меж контейнера, яким є форма.

Можна задати чотири можливих значення для цієї властивості або їх комбінацію:

- `Top`
- `Bottom`
- `Left`
- `Right`

Наприклад, якщо змінити значення цієї властивості на протилежне - `Bottom`, `Right`, тоді відстань між правою і нижньою стороною елемента і формою буде незмінною.

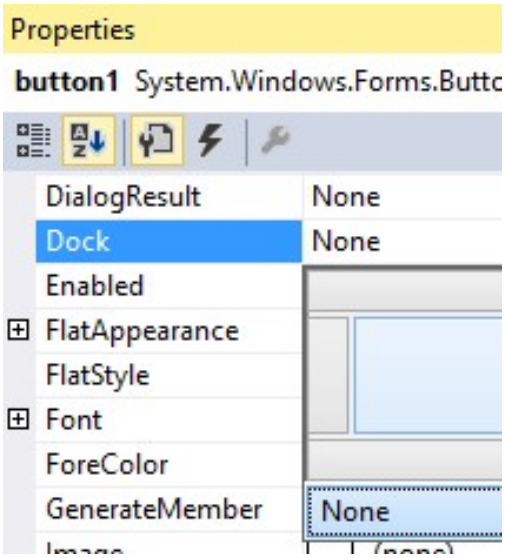
При цьому треба зазначити, що дана властивість враховує відстань до меж контейнера, а не форми. Тобто якщо на формі є елемент `Panel`, а на `Panel` розташована кнопка, то на кнопку впливатиме зміна меж `Panel`, а не форми. Розтягування форми у цьому випадку буде впливати тільки, якщо воно впливає на контейнер `Panel`.

Щоб задати цю властивість в коді, треба використати перелік `AnchorStyles`:

```
button1.Anchor = AnchorStyles.Left;
// задаємо комбінацію значень
button1.Anchor = AnchorStyles.Left | AnchorStyles.Top;
```

1.4.4. Властивість Dock

Властивість `Dock` дає змогу прикріпити елемент до певної сторони контейнера. За замовчуванням вона має значення `None`, але також дає змогу задати ще п’ять значень:



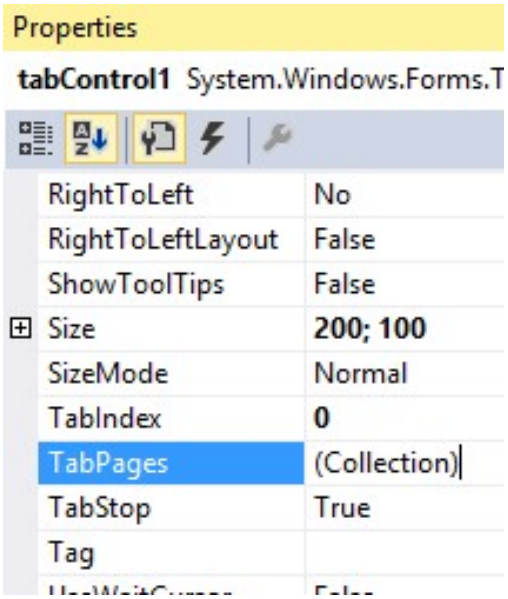
- `Top`: елемент притискається до верхньої межі контейнера;
- `Bottom`: елемент притискається до нижньої межі контейнера;
- `Left`: елемент притискається до лівого боку контейнера;
- `Right`: елемент прикріплюється до правого боку контейнера;
- `Fill`: елемент заповнює весь простір контейнера.

1.5. Панель вкладок TabControl і SplitContainer

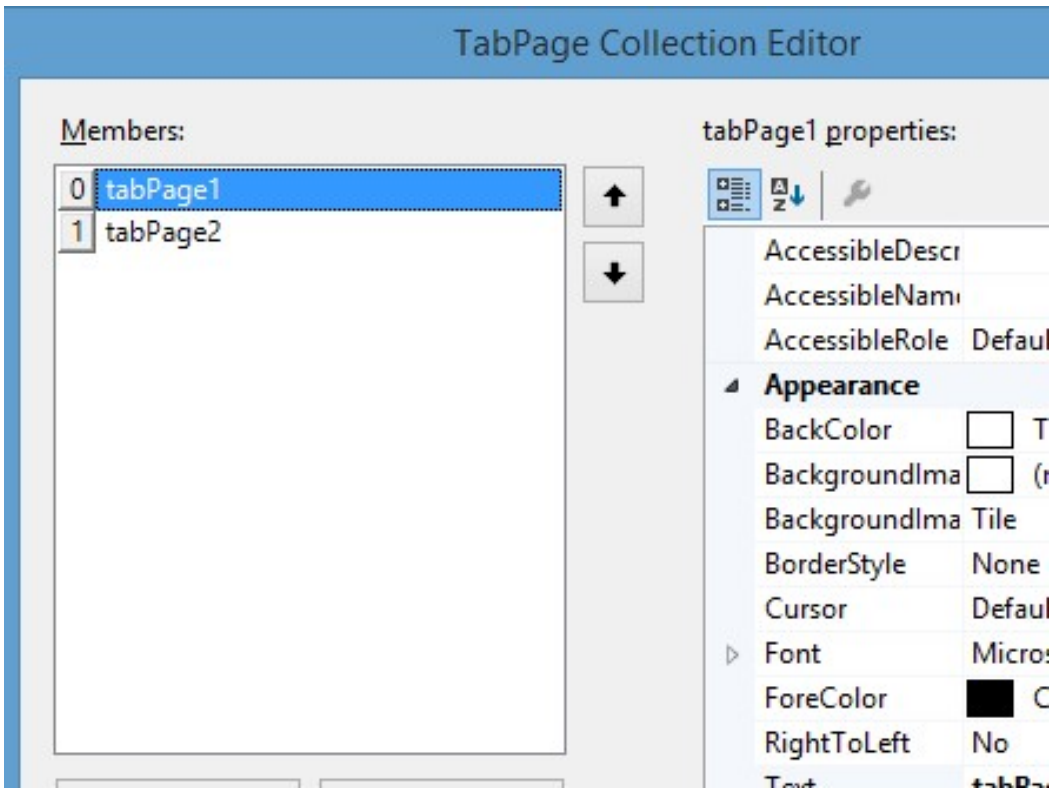
1.5.1. TabControl

Елемент `TabControl` дає змогу створити елемент управління з кількома вкладками. Кожна вкладка представляється класом `TabPage` і буде зберігати певний набір елементів управління – кнопки, текстові поля тощо.

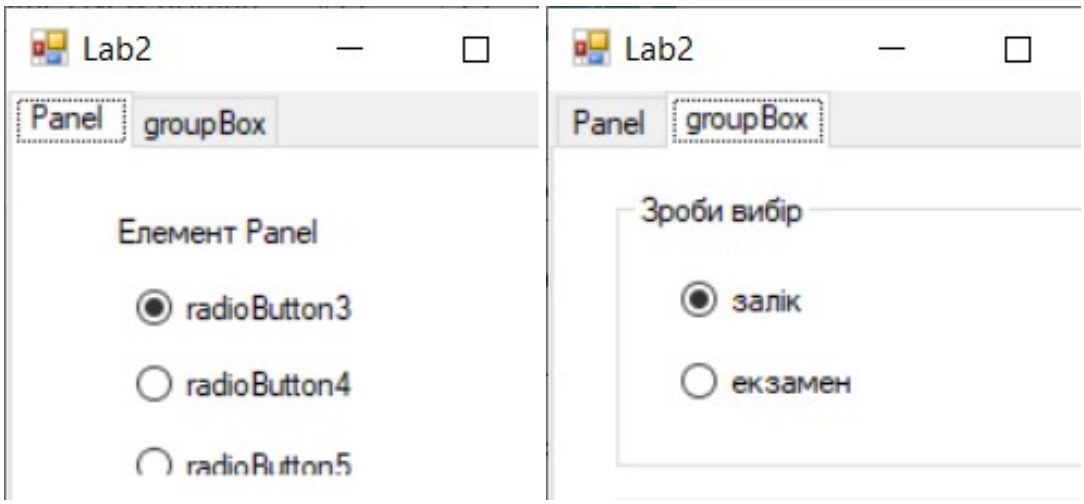
Щоб налаштувати вкладки елемента `TabControl` використаємо властивість `TabPages`. При перенесенні елемента `TabControl` з панелі інструментів на форму за замовчуванням створюються дві вкладки – `tabPage1` та `tabPage2`. Змінимо їх відображення за допомогою властивості `TabPages`:



Відкриється вікно редагування/додавання і видалення вкладок:



Кожна вкладка представляє свого роду панель, на яку можна додати інші елементи управління, а також заголовок, за допомогою якого можна переходити по вкладках. Текст заголовка задається за допомогою властивості `Text`.



Для програмного додавання нової вкладки спочатку потрібно її створити, а потім додати в колекцію `tabControl1.TabPages` за допомогою методу `Add`:

```
// додавання вкладки
TabPage newTabPage = new TabPage();
newTabPage.Text = "Тварини";
tabControl1.TabPages.Add(newTabPage);
```

Видалення вкладки:

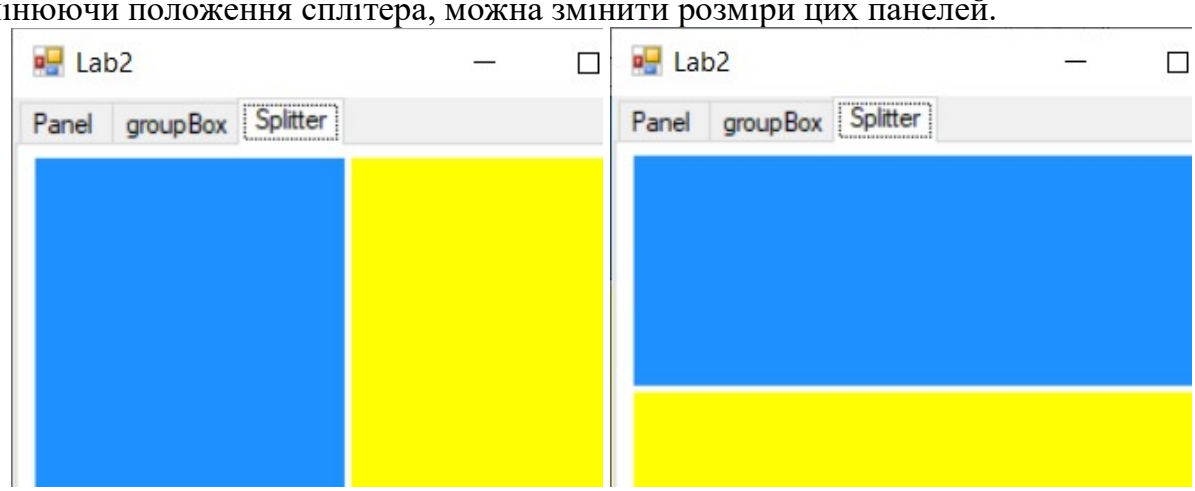
```
// видалення вкладки
// за індексом
tabControl1.TabPages.RemoveAt(0);
// по об'єкту
tabControl1.TabPages.Remove(newTabPage);
```

Вкладкою можна легко маніпулювати, звертаючись в колекції `tabControl1.TabPages` до вкладки за індексом:

```
// зміна властивостей
tabControl1.TabPages[0].Text = "Перша вкладка";
```

1.5.2. SplitContainer

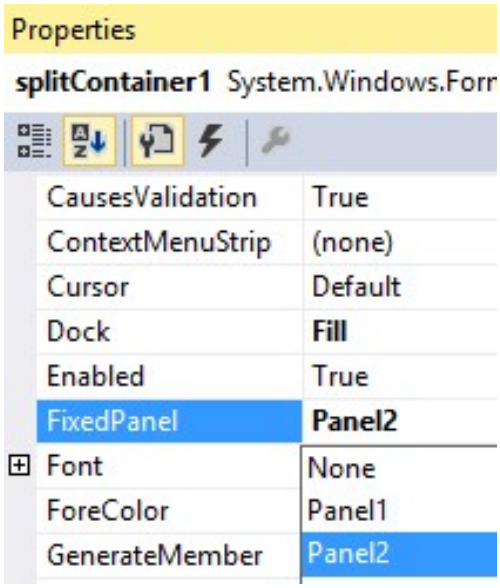
Елемент `SplitContainer` дає змогу створювати дві візуально розділені панелі. Змінюючи положення сплітера, можна змінити розміри цих панелей.



Використовуючи властивість `Orientation`, можна задати горизонтальне або вертикальне відображення сплітера на формі. В даному випадку ця властивість приймає значення `Horisontal` та `Vertical` відповідно.

Якщо слід заборонити зміну положення спліттера, можна присвоїти властивості `IsSplitterFixed` значення `true`. Таким чином, сплітер стане фіксованим, і змінити його положення вже не буде змоги.

За замовчуванням при розтягуванні форми або її звуженні також буде змінюватися розмір обох панелей сплітконтейнера. Однак можна закріпити за однією панеллю фіксовану ширину (при вертикальній орієнтації спліттера) або висоту (при горизонтальній орієнтації спліттера). Для цього потрібно використати властивість `FixedPanel` елемента `SplitContainer`. Як значення вона приймає панель, яку треба зафіксувати:



Щоб змінити положення сплітера в коді, можна використати властивість `SplitterDistance`, яке встановлює положення спліттера в пікселях від лівого або верхнього краю елемента `SplitContainer`. А за допомогою властивості `SplitterIncrement` можна задати крок, на який буде переміщатися спліттер за допомогою клавіш-стрілок.

Щоб приховати одну з двох панелей, можна встановити властивість `Panel1Collapsed` або `Panel2Collapsed` в `true`.

Елементи управління

Елементи управління є візуальними класами, які отримують введені користувачем дані і можуть ініціювати різні події. Всі елементи управління успадковуються від класу `Control` і тому мають ряд спільних властивостей:

- `Anchor`: визначає, як елемент буде розтягуватися;
- `BackColor`: визначає фоновий колір елемента;
- `BackgroundImage`: визначає фонове зображення елемента;

- `ContextMenu`: контекстне меню, яке відкривається при натисканні на елемент правою кнопкою миші. Здається за допомогою елемента `ContextMenu`;
- `Cursor`: представляє, як буде відображатися курсор миші при наведенні на елемент;
- `Dock`: задає розташування елемента на формі;
- `Enabled`: визначає, чи буде доступний елемент для використання. Якщо ця властивість має значення `False`, то елемент блокується;
- `Font`: встановлює шрифт тексту для елемента;
- `ForeColor`: визначає колір шрифту;
- `Location`: визначає координати верхнього лівого кута елементу управління;
- `Name`: ім'я елемента керування;
- `Size`: визначає розмір елемента;
- `Width`: ширина елемента;
- `Height`: висота елемента;
- `TabIndex`: визначає порядок обходу елемента після натискання на клавішу *Tab*;
- `Tag`: дає змогу зберігати значення, асоційоване з цим елементом управління.

1.6. Кнопка

Найбільш часто елементом управління, що використовується, є кнопка. Опрацьовуючи подія натискання кнопки, можна виконувати різні дії.

При натисканні на кнопку на формі в редакторі *Visual Studio* за замовчуванням відкривається код обробника події `Click`, який буде виконуватися при натисканні цієї кнопки:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World");
}
```

1.6.1. Оформлення кнопки

Щоб керувати зовнішнім відображенням кнопки, можна використовувати властивість `FlatStyle`, яка може мати такі значення:

- `Flat` - кнопка має плоский вигляд;
- `Popup` - кнопка має об'ємний вигляд при наведенні на неї вказівника, в інших випадках вона має плоский вигляд;
- `Standard` - кнопка має об'ємний вигляд (використовується за замовчуванням);
- `System` - вид кнопки залежить від налаштувань операційної системи.

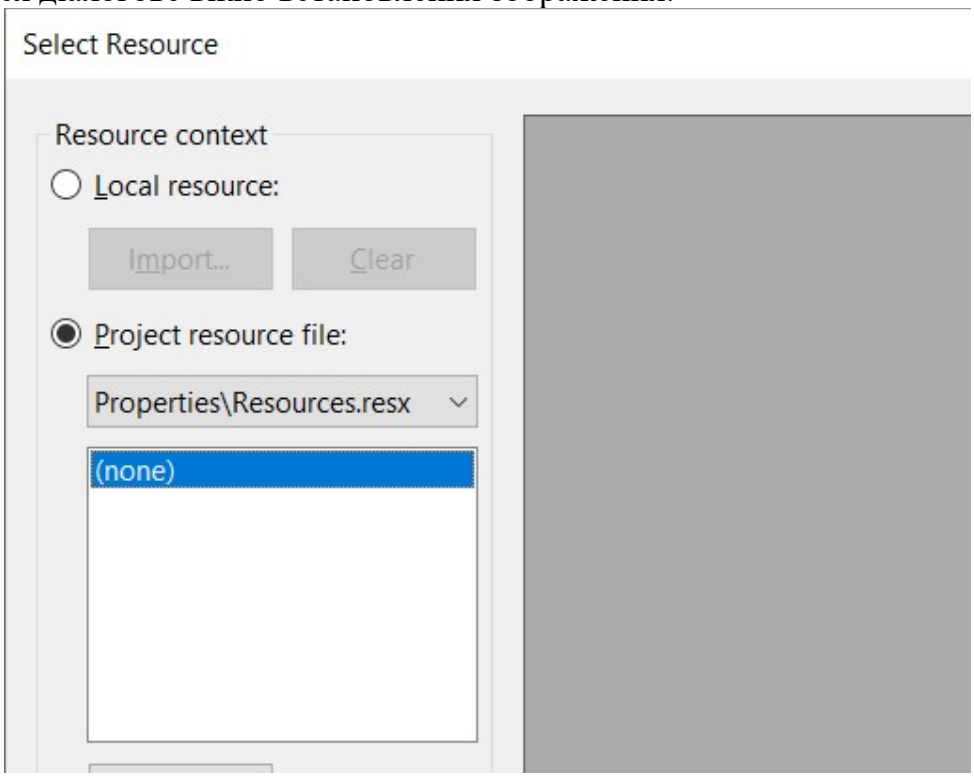
1.6.2. Зображення на кнопці

Як і для багатьох елементів управління, для кнопки можна задавати зображення за допомогою властивості `BackgroundImage`. Однак можна також

керувати розміщенням тексту і зображенням на кнопки. Для цього використовується властивість `TextImageRelation`, яка набуває таких значень:

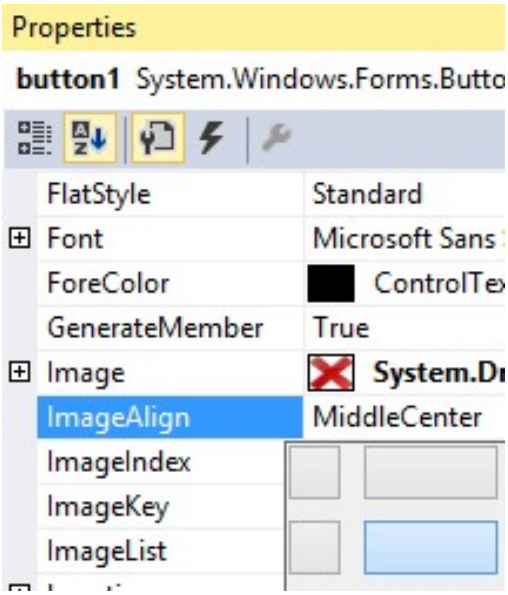
- `Overlay`: текст накладається на зображення;
- `ImageAboveText`: зображення розташовується над текстом;
- `TextAboveImage`: текст розташовується над зображенням;
- `ImageBeforeText`: зображення розташовується перед текстом;
- `TextBeforeImage`: текст розташовується перед зображенням.

Наприклад, встановимо для кнопки зображення. Для цього виберемо кнопку і у вікні *Властивостей* натиснемо на поле `Image` (не плутати з `BackgroundImage`). Відкриється діалогове вікно встановлення зображення:



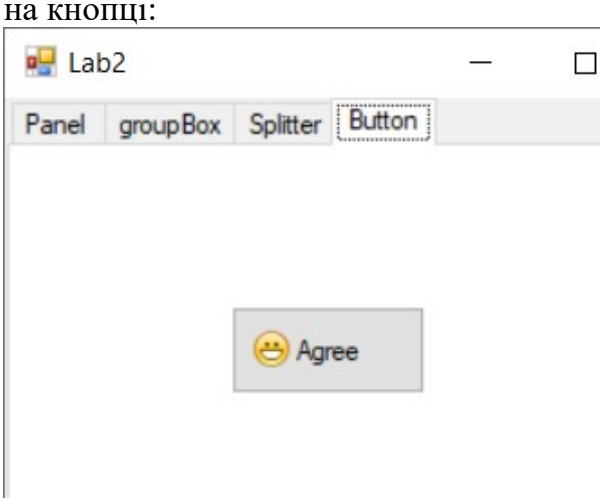
У цьому вікні виберемо опцію `Local Resource` і натиснемо на кнопку `Import`, після чого відкриється діалогове вікно вибору файла зображення.

Після вибору зображення можна встановити властивість `ImageAlign`, яка керує позиціюванням зображення на кнопці:



Доступні 9 варіантів, за допомогою яких можна прикріпити зображення до певної сторони кнопки. Залишимо значення за замовчуванням - `MiddleCenter`, тобто позиціонування по центру.

Потім перейдемо до властивості `TextImageRelation` і встановимо для неї значення `ImageBeforeText`. У підсумку отримаємо кнопку, для якої після зображення йде напис на кнопці:



1.6.3. Клавіші швидкого доступу

При роботі з формами при використанні клавіатури дуже зручно користуватися клавішами швидкого доступу. При натисканні на клавіатурі комбінації клавіш *Alt + деякий символ*, буде викликатися певна кнопка. Наприклад, задамо для деякої кнопки властивість `Text` рівне `&Main`. Перший знак - амперсанд - визначає букву, яка буде підкреслена. В даному випадку напис буде виглядати як *Main*. Тепер, щоб викликати подію `Click`, достатньо натиснути на комбінацію клавіш *Alt + M*.

1.6.4. Кнопки за замовчуванням

Форма, на якій розміщуються всі елементи управління, має властивості, що дають змогу призначати кнопку за замовчуванням і кнопку скасування.

Так, властивість форми `AcceptButton` дає змогу призначати кнопку за замовчуванням, яка буде спрацьовувати після натискання на клавіші `Enter`.

Аналогічно працює властивість форми `CancelButton`, яка визначає кнопку скасування. Призначивши таку кнопку, можна викликати її натискання, натиснувши на клавішу `Esc`.

1.7. Мітки та покликання

1.7.1. Label

Для відображення простого тексту на формі, доступного тільки для читання, призначений елемент `Label`. Щоб задати відображення тексту мітки, треба встановити властивість `Text` елемента.

1.7.2. LinkLabel

Особливий тип міток представляють елементи `LinkLabel`, які призначені для виведення покликань, що аналогічні покликанням, розміщеним на стандартних веб-сторінках.

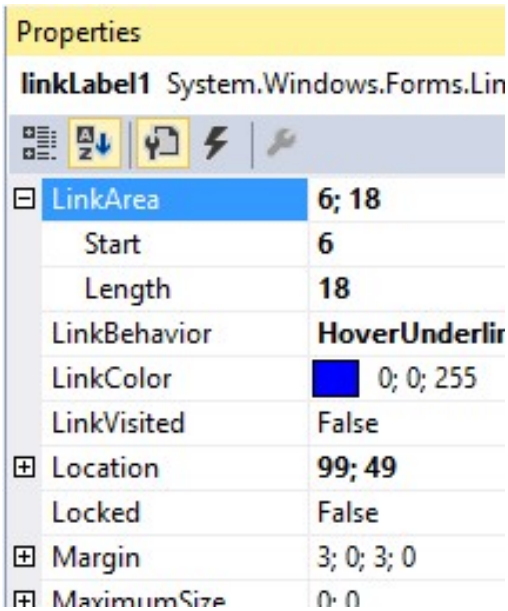
Як і зі звичайними покликаннями на веб-сторінках, щодо елемента можна визначити три кольори:

- властивість `ActiveLinkColor` задає колір покликання при натисканні;
- властивість `LinkColor` задає колір покликання до натискання, по якому ще не було переходів;
- властивість `VisitedLinkColor` задає колір покликання, по якому вже були переходи.

Крім кольору покликання для даного елемента можна встановити властивість `LinkBehavior`, яка керує поведінкою покликання. Ця властивість приймає чотири можливих значення:

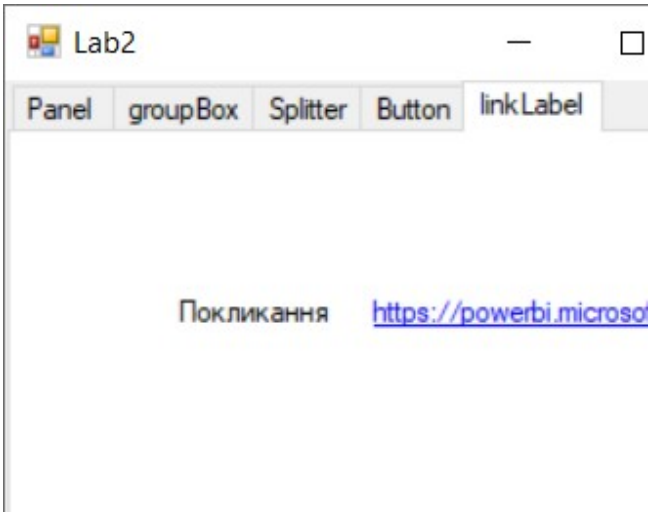
- `SystemDefault`: для покликання встановлюються системні налаштування;
- `AlwaysUnderline`: покликання завжди підкреслюється;
- `HoverUnderline`: покликання підкреслюється тільки при наведенні на нього курсора миші;
- `NeverUnderline`: покликання ніколи не підкреслюється.

За замовчуванням весь текст на даному елементі вважається покликанням. Однак за допомогою властивості `LinkArea` можна змінити область покликання. Наприклад, щоб не включати в покликання перші шість символів, використаємо властивість `Start`:



Щоб виконати перехід за покликанням після натискання на нього, потрібно додатково написати код для опрацювання події `LinkClicked` елемента `LinkLabel`.

Наприклад, нехай на формі є елемент покликання, який називається `linkLabel1` і який містить деяке посилання:

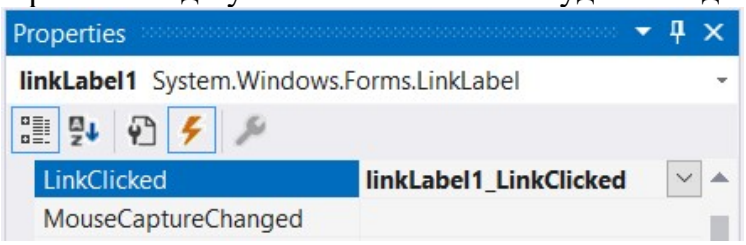


Щоб перейти за покликанням, створимо обробник `LinkClicked`:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        // задаємо обробник події
        linkLabel1.LinkClicked += linkLabel1_LinkClicked;
    }

    private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs
e)
```

```
    {
        System.Diagnostics.Process.Start("https://powerbi.microsoft.com/ ");
    }
}
```

При цьому обробник події у вікні властивостей буде вигладати так:



Метод `System.Diagnostics.Process.Start()` відкриє дане покликання у веб-браузері, який є браузером за замовчуванням.

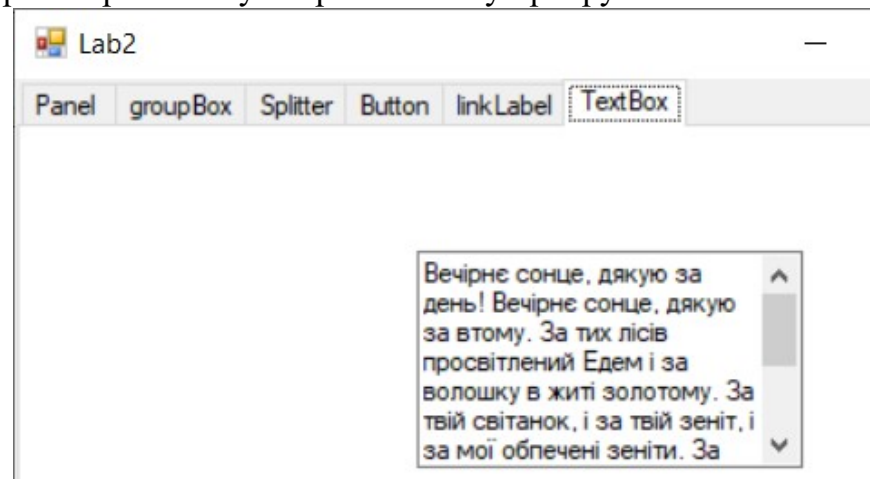
1.8. Текстове поле TextBox

Для введення і редагування тексту призначені текстове поле – елемент `TextBox`. Текст елемента `TextBox` можна встановити або отримати за допомогою властивості `Text`, як і в елемента `Label`.

За замовчуванням при перенесенні елемента з панелі інструментів створюється однорядкове текстове поле. Для відображення великих обсягів інформації в текстовому полі потрібно використовувати його властивості `Multiline` та `ScrollBars`. При встановленні значення `true` для властивості `Multiline`, всі надлишкові символи, що виходять за межі поля, будуть переноситися на новий рядок.

Крім того, можна зробити прокрутку текстового поля, встановивши для його властивості `ScrollBars` одне з таких значень:

- `None`: без прокручувань (за замовчуванням);
- `Horizontal`: створює горизонтальну прокрутку при довжині рядка, що перевищує ширину текстового поля;
- `Vertical`: створює вертикальну прокрутку, якщо рядки не поміщаються в текстовому полі;
- `Both`: створює вертикальну і горизонтальну прокрутки.



1.8.1. Автозаповнення текстового поля

Елемент `TextBox` володіє достатніми можливостями для створення автодоповнюваного поля. Для цього потрібно властивість `AutoCompleteCustomSource` елемента `TextBox` прив'язати до деякої колекції, з якої беруться дані для заповнення поля.

Режим автодоповнення представлений властивістю `AutoCompleteMode`, яка має такі значення:

- `None`: відсутність автодоповнення;
- `Suggest`: пропонує варіанти для введення, але не доповнює;
- `Append`: доповнює введене значення до рядка зі списку, але не пропонує варіанти для вибору;
- `SuggestAppend`: одночасно пропонує варіанти для автодоповнення і доповнює введене користувачем значення.

Отже, додамо на форму текстове поле і допишемо в код події завантаження форми наступні рядки:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        AutoCompleteStringCollection vCustomCollection = new
AutoCompleteStringCollection()
        {
            "Франко",
            "Фарбований",
            "Фаренгейт",
            "Костенко"
        };
        textBox1.AutoCompleteCustomSource = vCustomCollection;
        textBox1.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
        textBox1.AutoCompleteSource = AutoCompleteSource.CustomSource;
    }
}
```

1.8.2. Перенесення по словах

Щоб текст в елементі `TextBox` переносився по словах, властивість `WordWrap` треба встановити рівною `true`. Тобто якщо одне слово не вміщається у рядок, то воно переноситься на наступний. Дана властивість буде працювати тільки для багаторядкових текстових полів.

1.8.3. Введення пароля

Також даний елемент має властивості `PasswordChar` та `UseSystemPasswordChar`, що дають змогу зробити з нього поле для введення пароля.

Властивість `PasswordChar` за замовчуванням не має значення, якщо встановити певний символ, то цей символ буде відображатися при введенні будь-яких символів у текстове поле.

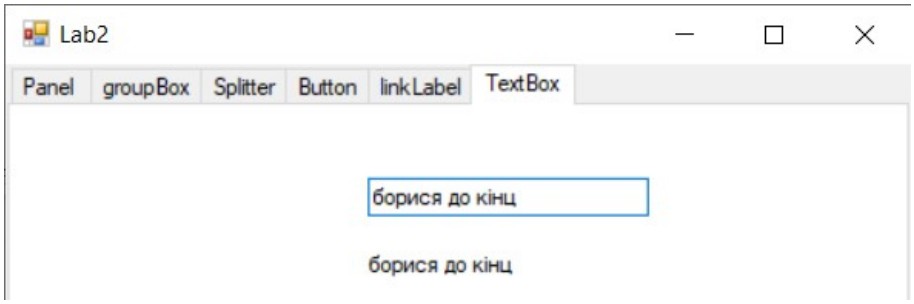
Властивість `UseSystemPasswordChar` має схожу дію. Якщо встановити її значення в `true`, то замість введених символів у текстовому полі буде відображатися знак пароля, прийнятий у системі, наприклад, зірочка.

1.8.4. Подія TextChanged

З усіх подій елемента `TextBox` слід розглянути подію `TextChanged`, яка спрацьовує при зміні тексту в елементі.

Наприклад, помістимо на форму, крім текстового поля, мітку і зробимо так, щоб при зміні тексту в текстовому полі також змінювався текст на мітці:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        textBox1.TextChanged += textBox1_TextChanged;
    }

    private void textBox1_TextChanged(object sender, EventArgs e)
    {
        label1.Text = textBox1.Text;
    }
}
```



1.9. Елемент MaskedTextBox

Елемент `MaskedTextBox` фактично є звичайним текстовим полем. Однак він дає змогу контролювати введення значень і автоматично перевіряти їх на наявність помилок.

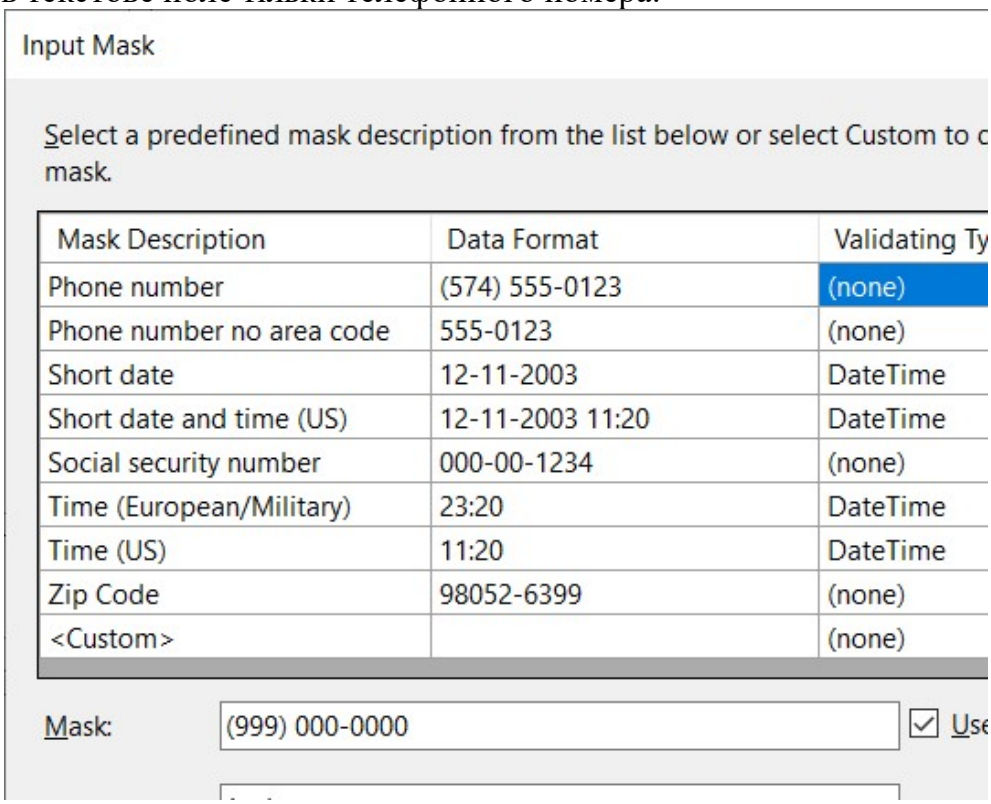
Щоб контролювати символи, які вводяться в поле, потрібно задати маску. Для встановлення маски можна використати такі символи:

- `0`: дає змогу вводити тільки цифри;
- `9`: дає змогу вводити цифри та пробіли;
- `#`: дає змогу вводити цифри, пробіли та знаки '+' і '-';
- `L`: дає змогу вводити тільки літерні символи;

- `?`: дає змогу вводити додаткові необов'язкові літерні символи;
- `A`: дає змогу вводити літерні і цифрові символи;
- `.`: задає позицію роздільника цілої та дробової частини;
- `,`: використовується для поділу розрядів в цілій частині числа;
- `:`: використовується в часових проміжках – розділяє години, хвилини і секунди;
- `/`: використовується для поділу дат;
- `$`: використовується як символ валюти.

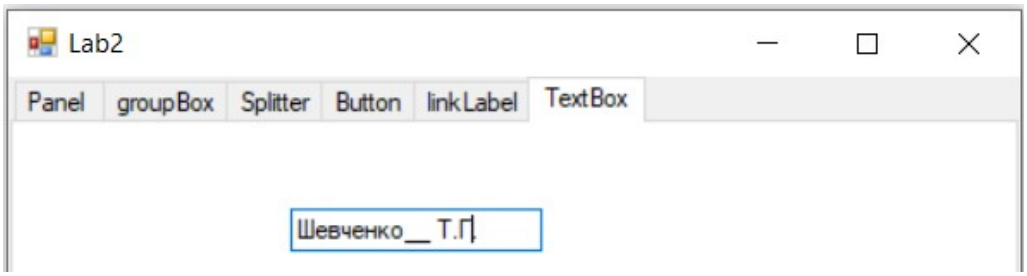
Щоб встановити маску, треба використати властивість `Mask` елемента. При виборі цієї властивості відкриється вікно для встановлення одного зі стандартних шаблонів маски.

Зокрема можна вибрати `Phone number` (Телефонний номер), який має на увазі введення в текстове поле тільки телефонного номера:



Таким чином в текстове поле можна ввести тільки цифри, отримавши в підсумку телефонний номер.

Створимо власну маску. Наприклад, створимо маску для введення прізвища обмеженої довжини, ініціалів імені та по батькові. Для цього вкажемо у властивості `Mask` значення `L????????? L.L.`. Тоді введення в текстове поле буде виглядати наступним чином:



Даний елемент також має ряд властивостей, які можна використовувати для управління введенням. Наприклад, властивість `BeepOnError` при встановленні значення `true` подає звуковий сигнал при введенні некоректного символу.

Властивість `HidePromptOnLeave` зі значенням `true` при втраті текстовим полем фокусу приховує символи, вказані в `PromptChar`.

Властивість `PromptChar` вказує на символ, який відображається в полі на місці введення символів. За замовчуванням стоїть знак підкреслення.

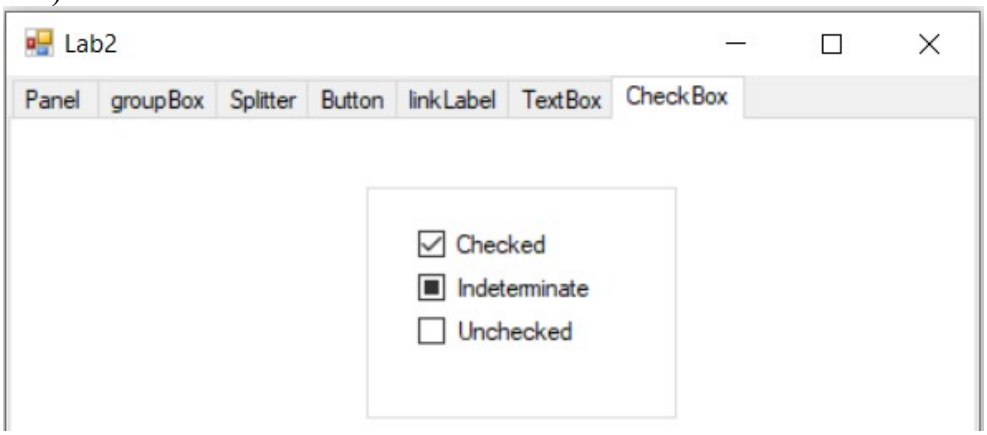
Властивість `AsciiOnly` при значенні `true` дає змогу вводити тільки ASCI-символи, тобто символи з діапазону A-Z і a-z.

1.10. Елементи Radiobutton і CheckBox

1.10.1. CheckBox

Елемент `CheckBox` (прапорець) призначений для установки одного з двох значень: *включений* або *виключений*. Щоб включити прапорець, треба встановити у його властивості `Checked` значення `true`.

Крім властивості `Checked`, у елемента `CheckBox` є властивість `CheckState`, яка дає змогу задати для прапорця одне з трьох станів - `Checked` (включений), `Indeterminate` (прапорець знаходиться в неактивному стані) і `Unchecked` (виключений).



Розглянемо властивість `AutoCheck`. Якщо вона має значення `false`, то стан прапорця змінювати не можна. За замовчуванням вона має значення `true`.

При зміні стану прапорця він генерує подію `CheckedChanged`. Опрацьовуючи цю подію, можна виконувати певні дії:

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
```

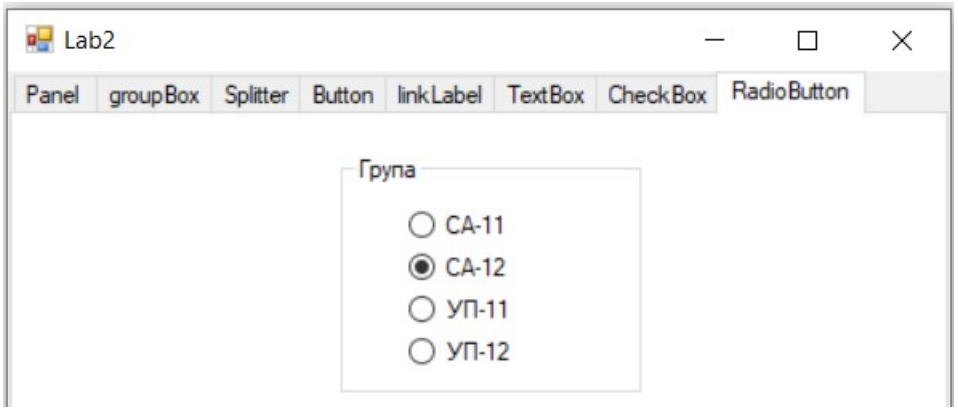
```
    // приводимо відправника до елементу типу CheckBox
    CheckBox checkBox = (CheckBox)sender;
    if (checkBox.Checked == true)
    {
        MessageBox.Show("Прапорець " + checkBox.Text + " включений");
    }
    else
    {
        MessageBox.Show("Прапорець " + checkBox.Text + " виключений");
    }
}
```

1.10.2. Radiobutton

На елемент `CheckBox` схожий елемент `RadioButton` (перемикач). Перемикачі розташовуються групами, і включення одного перемикача означає відключення всіх інших.

Щоб встановити у перемикача включений стан, треба присвоїти його властивості `Checked` значення `true`.

Для створення групи перемикачів, з яких можна було б вибирати, треба помістити кілька перемикачів в певний контейнер, наприклад, в елемент `GroupBox` або `Panel`. Перемикачі, що знаходяться в різних контейнерах, будуть належати до різних груп:



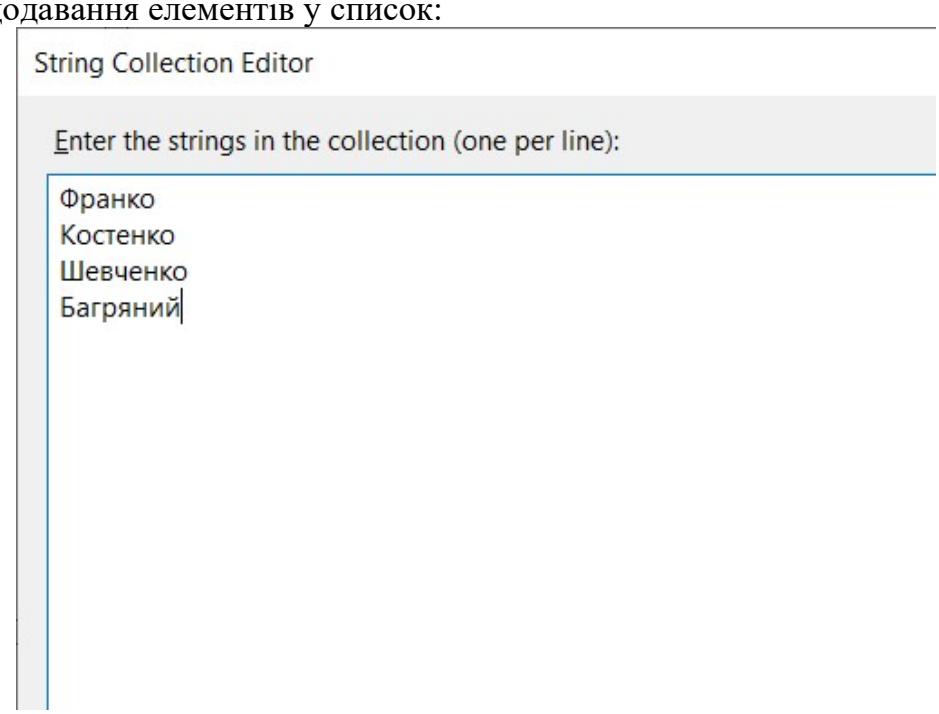
Схожим чином можна перехоплювати перемикання перемикачів у групі, опрацьовуючи подію `CheckedChanged`. Зв'язавши кожний перемикач групи з одним обробником цієї події, можна отримати той перемикач, який в даний момент обраний:

```
private void radioButton_CheckedChanged(object sender, EventArgs e)
{
    // приводимо відправника до елементу типу RadioButton
    RadioButton radioButton = (RadioButton)sender;
    if (radioButton.Checked)
    {
        MessageBox.Show("Ви вибрали " + radioButton.Text);
    }
}
```

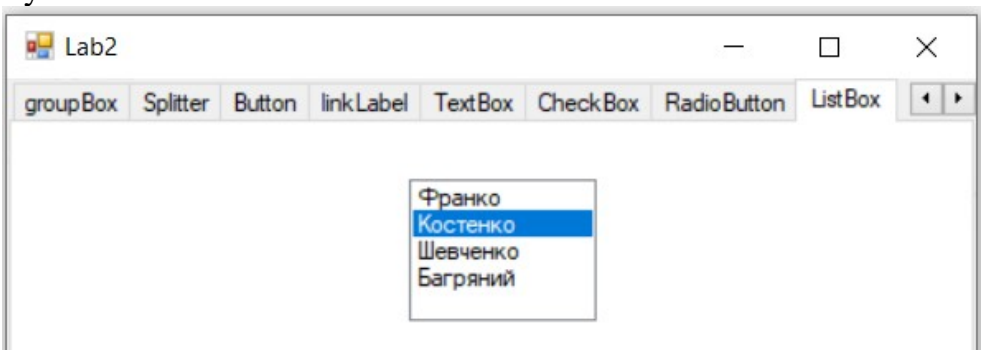
1.11. ListBox

Елемент `ListBox` є простим списком, ключовою властивістю якого є властивість `Items`, яке зберігає набір усіх елементів списку.

Елементи можуть додаватися в список як під час проєктування, так і програмним способом. Після подвійного клацання на властивість відображається вікно для додавання елементів у список:



У порожнє поле вводимо по одному елементу списку - по одному у кожному рядку. Після цього всі додані нами елементи опиняться в списку, і ми зможемо їх використовувати:



1.11.1. Програмне управління елементами в ListBox

1.11.1.1. Додавання елементів

Отже, всі елементи списку входять у властивість `Items`, яка представляє собою колекцію. Для додавання нового елемента в цю колекцію, а значить і в список, використовується метод `Add`, наприклад: `listBox1.Items.Add` ("*Новий*

елемент");. При використанні цього методу кожен елемент додається в кінець списку.

Можна додати відразу кілька елементів, наприклад, масив. Для цього використовується метод `AddRange`:

```
string[] vWriters = { "Франко", "Костенко", "Шевченко", "Багряний"};
listBox1.Items.AddRange(vWriters);
```

1.11.1.2. Вставка елементів

На відміну від простого додавання вставка проводиться за певним індексом списку за допомогою методу `Insert`:

```
listBox1.Items.Insert(1, "Нестайко");
```

У даному випадку вставляємо елемент на другу позицію в списку, так як відлік позицій починається з нуля.

1.11.1.3. Видалення елементів

Для видалення елемента по його тексту використовується метод `Remove`:

```
listBox1.Items.Remove("Костенко");
```

Щоб видалити елемент по його індексу в списку, використовується метод `RemoveAt`:

```
listBox1.Items.RemoveAt(1);
```

Крім того, можна очистити відразу весь список, застосувавши метод `Clear`:

```
listBox1.Items.Clear();
```

1.11.1.4. Доступ до елементів списку

Використовуючи індекс елемента, можна отримати значення цього елемента списку. Наприклад, отримаємо перший елемент списку:

```
string firstElement = listBox1.Items[0];
```

Метод `Count` дає змогу визначити кількість елементів у списку:

```
int number = listBox1.Items.Count();
```

1.11.1.5. Виділення елементів списку

При виділенні елементів списку ними можна керувати як через індекс, так і через сам виділений елемент. Отримати виділені елементи можна за допомогою наступних властивостей елемента `ListBox`:

- `SelectedIndex`: повертає або встановлює номер виділеного елемента списку. Якщо виділені елементи відсутні, тоді властивість має значення -1;
- `SelectedIndices`: повертає або встановлює колекцію виділених елементів у вигляді набору їх індексів;
- `SelectedItem`: повертає або встановлює текст виділеного елемента;
- `SelectedItems`: повертає або встановлює виділені елементи у вигляді колекції.

За замовчуванням список підтримує виділення одного елемента. Щоб додати можливість виділення декількох елементів, треба встановити у його властивості `SelectionMode` значення `MultiSimple`.

Щоб виділити елемент програмно, треба застосувати метод `SetSelected` (`int index, bool value`), де `index` - номер виділеного елемента. Якщо другий параметр - `value` має значення `true`, то елемент за вказаною індексу виділяється, якщо `false`, то виділення навпаки ховається:

```
listBox1.SetSelected(2, true); // буде виділено третій елемент
```

Щоб зняти виділення з усіх виділених елементів, використовується метод `ClearSelected`.

1.11.2. Подія SelectedIndexChanged

З усіх подій елемента `ListBox` у першу чергу треба розглянути подію `SelectedIndexChanged`, яка виникає при зміні виділеного елемента:

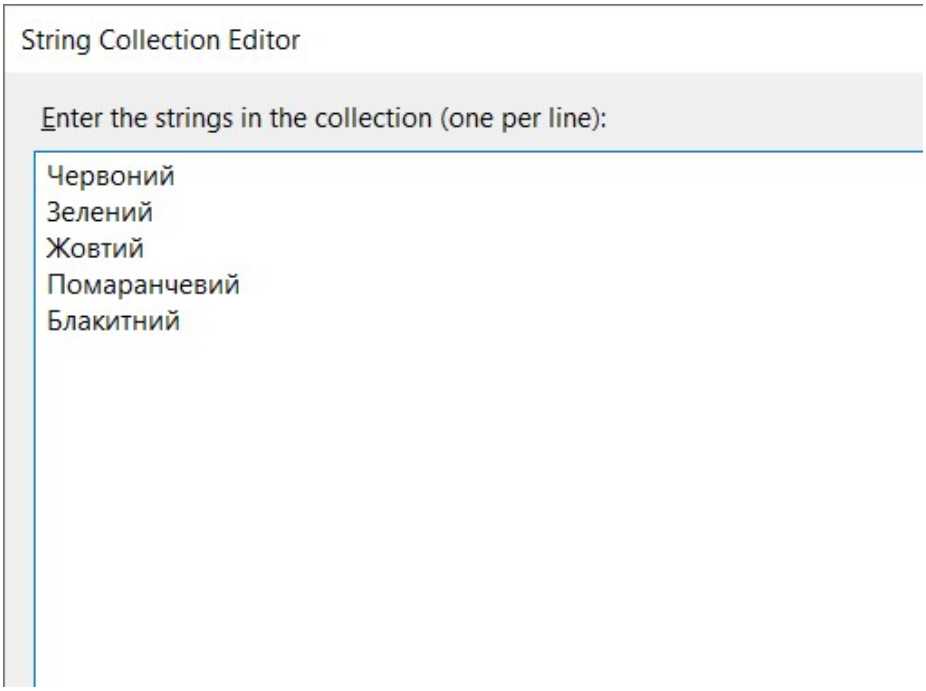
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        string[] vWriters = { "Франко", "Костенко", "Шевченко", "Багряний"};
        listBox1.Items.AddRange(vWriters);

        listBox1.SelectedIndexChanged += listBox1_SelectedIndexChanged;
    }

    void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string selectedWriters = listBox1.SelectedItem.ToString();
        MessageBox.Show(selectedWriters);
    }
}
```

У даному випадку за вибором елемента списку буде відображатися повідомлення з виділеним елементом.

1.12. Елемент ComboBox

Елемент `ComboBox` утворює список, що випадає, і поєднує функціональність компонентів `ListBox` та `TextBox`. Для зберігання елементів списку в `ComboBox` також призначена властивість `Items`.

Аналогічно до елемента `ListBox`, можна додавати елементи `ComboBox`:



Також можна програмно керувати елементами.

Додавання елементів:

```
// додаємо один елемент
comboBox1.Items.Add("Білий");
// додаємо набір елементів
comboBox1.Items.AddRange(new string[] { "Жовтий", "Чорний" });
// додаємо один елемент на певну позицію
comboBox1.Items.Insert(1, "Коричневий");
```

При додаванні за допомогою методів `Add/AddRange` всі нові елементи розташовуються в кінці списку. Однак якщо у `ComboBox` встановити властивість `Sorted` рівною `true`, тоді при додаванні буде автоматично виконуватись сортування.

Видалення елементів:

```
// видаляємо один елемент
comboBox1.Items.Remove("Зелений");
// видаляємо елемент за індексом
comboBox1.Items.RemoveAt(1);
// видаляємо всі елементи
comboBox1.Items.Clear();
```

Також можна отримати елемент за індексом і виконувати з ним різні дії. Наприклад, змінити його:

```
comboBox1.Items[0] = "Червоний";
```

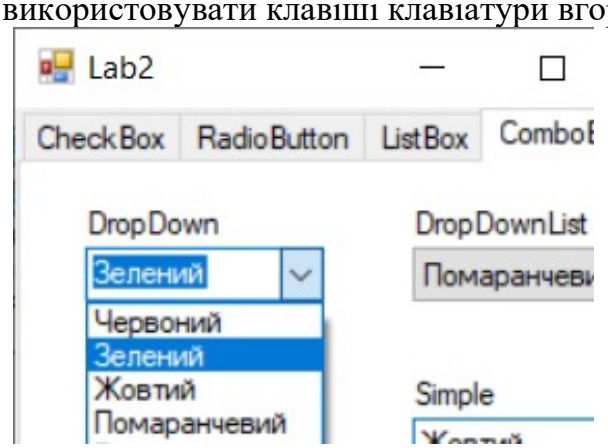
1.12.1. Налаштування оформлення ComboBox

За допомогою ряду властивостей можна налаштувати стиль оформлення компонента. Так, властивість `DropDownWidth` задає ширину списку. За допомогою властивості `DropDownHeight` можна встановити висоту списку.

Ще одна властивість `MaxDropDownItems` дає змогу задати число видимих елементів списку - від 1 до 100. За замовчуванням це число дорівнює 8.

Інша властивість `DropDownStyle` задає стиль `ComboBox`. Вона може приймати три можливих значення:

- `DropDown`: використовується за замовчанням. Можна відкрити список варіантів, що випадає, при введенні значення в текстове поле або натиснувши на кнопку зі стрілкою в правій частині елемента; відобразиться список, що випадає, в якому можна вибрати можливий варіант;
- `DropDownList`: щоб відкрити список, що випадає, треба натиснути на кнопку зі стрілкою в правій стороні елемента;
- `Simple`: ComboBox є простим текстовим полем, в якому для переходу між елементами можна використовувати клавіші клавіатури вгору/вниз.



1.12.2. Подія SelectedIndexChanged

Найбільш важливою подією для `ComboBox` є подія `SelectedIndexChanged`, що дає змогу відстежити вибір елемента в списку:

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

        comboBox1.SelectedIndexChanged += comboBox1_SelectedIndexChanged;
    }

    void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string selectedState = comboBox1.SelectedItem.ToString();
        MessageBox.Show(selectedState);
    }
```

```
}
```

Тут також властивість `SelectedItem` буде посилатися на вибраний елемент.

1.13. Прив'язка даних в ListBox і ComboBox

Крім прямого додавання елементів в колекцію `Items` компонентів `ListBox` і `ComboBox`, також можна використовувати механізм прив'язки даних.

Прив'язка даних в `ListBox` та `ComboBox` реалізовується за допомогою наступних властивостей:

- `DataSource`: джерело даних – будь-який масив або колекція об'єктів;
- `DisplayMember`: властивість об'єкта, яка буде використовуватись для відображення в ListBox/ComboBox;
- `ValueMember`: властивість об'єкта, що буде використовуватись як його значення.

Розглянемо приклад.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        List<Phone> phones = new List<Phone>
        {
            new Phone { Id=11, Name="Samsung Galaxy A30", Year=2023},
            new Phone { Id=12, Name="Samsung Galaxy M35", Year=2024},
            new Phone { Id=13, Name="iPhone 15", Year=2023},
            new Phone { Id=14, Name="Motorola G54", Year=2024},
            new Phone { Id=15, Name="Xiaomi Redmi Note 15", Year=2024}
        };

        listBox1.DataSource = phones;
        listBox1.DisplayMember = "Name";
        listBox1.ValueMember = "Id";

        listBox1.SelectedIndexChanged += listBox1_SelectedIndexChanged;
    }

    void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        // отримуємо id виділеного об'єкта
        int id = (int)listBox1.SelectedValue;

        // отримуємо весь виділений об'єкт
        Phone phone = (Phone)listBox1.SelectedItem;
        MessageBox.Show(id.ToString() + ". " + phone.Name);
    }
}
class Phone
{
```

```
    public int Id{ get; set; }
    public string Name{ get; set; }
    public int Year{ get; set; }
}
```

Отже, на формі є список `ListBox` з ім'ям `listBox1`. У коді є клас `Phone` з трьома властивостями, об'єкти якого потрібно виводити в список. Ця задача відрізняється від попередніх задач тим, що вона складніша, так як раніше ми виводили звичайні рядки, а тут складні об'єкти.

Для виведення використаємо механізм прив'язки. Спочатку встановимо список телефонів як джерело даних:

```
listBox1.DataSource = phones;
```

Потім встановимо як відображувану властивість властивість `Name` класу `Phone`, а як властивості значення - властивість `Id`:

```
listBox1.DisplayMember = "Name";
listBox1.ValueMember = "Id";
```

Значення відображуваної властивості потім побачимо в списку. Воно буде представляти кожен окремий об'єкт `Phone`.

За допомогою властивості значення, яким є властивість `Id`, можна спростити роботу з джерелом даних. В даному випадку воно не грає великої ролі. Але якби ми використовували як джерело даних певний набір об'єктів з бази даних, то з допомогою `Id` було б простіше видаляти, оновлювати і взаємодіяти з базою даних.

Тепер якщо виділити певний об'єкт, то властивість `SelectedItem` елемента `ListBox` буде містити об'єкт `Phone`, з якого можна отримати значення властивостей:

```
Phone phone = (Phone)listBox1.SelectedItem;
string name = phone.Name;
```

А виділене значення, тобто значення властивості `Id` виділеного телефону, буде знаходитися у властивості `SelectedValue`.

Все це характерно і для елемента `ComboBox`. Нехай крім `ListBox` на формі є `ComboBox`:

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

        List<Phone> phones = new List<Phone>
        {
            new Phone { Id=11, Name="Samsung Galaxy A30", Year=2023},
            new Phone { Id=12, Name="Samsung Galaxy M35", Year=2024},
            new Phone { Id=13, Name="iPhone 15", Year=2023},
            new Phone { Id=14, Name="Motorola G54", Year=2024},
            new Phone { Id=15, Name="Xiaomi Redmi Note 15", Year=2024}
        };

        comboBox1.DataSource = phones;
        comboBox1.DisplayMember = "Name";
```

```
        comboBox1.ValueMember = "Id";

        comboBox1.SelectedIndexChanged += comboBox1_SelectedIndexChanged;

        listBox1.DisplayMember = "Name";
        listBox1.ValueMember = "Id";
    }

    void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        Phone phone = (Phone)comboBox1.SelectedItem;
        listBox1.Items.Add(phone);
    }
}

class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Year { get; set; }
}
```

Тут також для `ComboBox` встановлюється прив'язка, а також відображається властивість і властивість значення. Крім того, опрацьовується подія вибору елемента в `ComboBox` так, щоб обраний елемент потрапляв в `ListBox`.

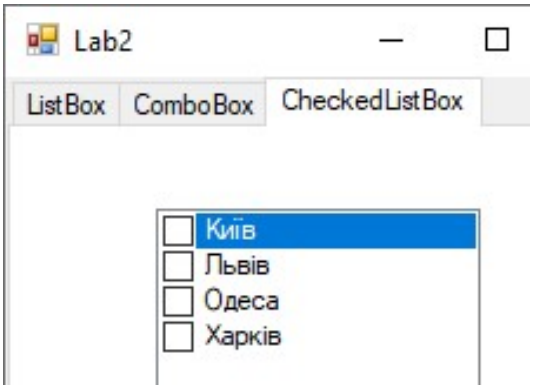
На відміну від `ListBox` елемент `ComboBox` має три властивості для опрацювання виділеного елементу:

- `SelectedItem`: вибраний елемент;
- `SelectedValue`: значення властивості значення, в даному випадку властивість `Id`;
- `SelectedText`: значення властивості відображення, в даному випадку властивість `Name` класу `Phone`.

1.14. Елемент CheckedListBox

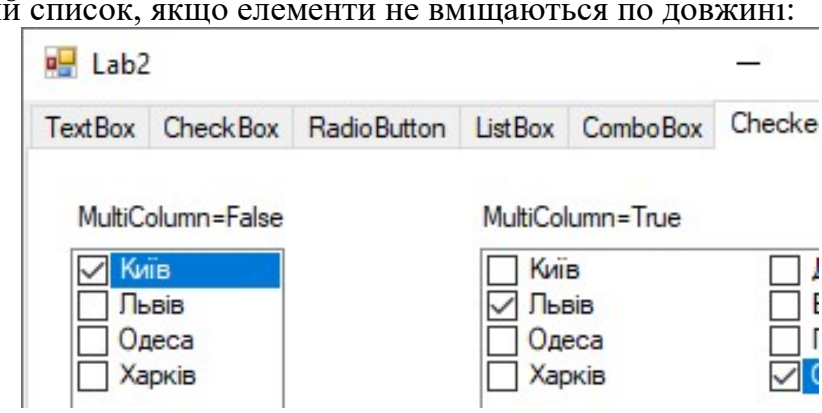
Елемент `CheckedListBox` представляє комбінацію компонентів `ListBox` та `CheckBox`. Для кожного елемента такого списку визначено спеціальне поле `CheckBox`, яке можна відмітити (включити, активувати).

Всі елементи в `CheckedListBox` задаються у властивості `Items`. Також, як і для елементів `ListBox` і `ComboBox`, можна задати набір елементів. За замовчуванням для кожного нового доданого елемента прапорець не встановлено:



Щоб поставити позначку в `checkBox` поруч з елементом у списку, потрібно спочатку виділити елемент і додатковим клацанням встановити прапорець. Однак це не завжди зручно, і за допомогою властивості `CheckOnClick` та встановлення для нього значення `true` можна визначити відразу вибір елемента і установку для нього прапорця одним натисканням.

Інша властивість `MultiColumn` при значенні `true` дає змогу зробити багатоколонний список, якщо елементи не вміщаються по довжині:



Виділений елемент також можна отримати за допомогою властивості `SelectedItem`, а його індекс - за допомогою свойства `SelectedIndex`. Але це можливо тільки, якщо для властивості `SelectionMode` встановлено значення `One`, що означає виділення тільки одного елемента.

При встановленні для властивості `SelectionMode` значень `MultiSmple` і `MultiExtended` можна вибрати одразу кілька елементів, і тоді всі вибрані елементи будуть доступні в властивості `SelectedItems`, а їх індекси - у властивості `SelectedIndeces`.

І оскільки можна поставити позначку не для всіх обраних елементів, то щоб окремо отримати відмічені елементи, у `CheckedListBox` є властивості `CheckedItems` і `CheckedIndices`.

Для додавання і видалення елементів в `CheckedListBox` визначені всі ті ж методи, що і в `ListBox`:

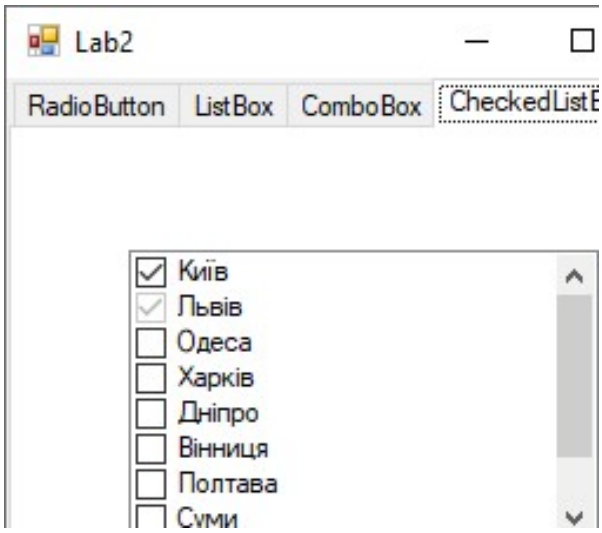
- `Add(item)`: додає один елемент;
- `AddRange(array)`: додає у список масив елементів;
- `Insert(index, item)`: додає елемент за певним індексом;

- `Remove(item)`: видаляє елемент;
- `RemoveAt(index)`: видаляє елемент за певним індексом;
- `Clear()`: повністю очищає список.

1.14.1. SetItemChecked і SetItemCheckState

До особливостей елемента можна віднести методи `SetItemChecked` і `SetItemCheckState`. Метод `SetItemChecked` дає змогу встановити або скинути позначку на одному з елементів. А метод `SetItemCheckState` дає змогу встановити прапорець в один з трьох станів: `Checked` (відмічено), `Unchecked` (не відмічено) і `Indeterminate` (проміжний стан, не визначено):

```
checkedListBox1.SetItemChecked(0, true);
checkedListBox1.SetItemCheckState(1, CheckState.Indeterminate);
```

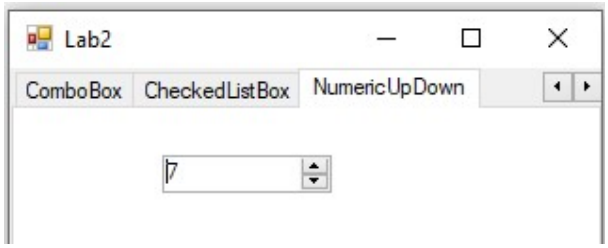


1.15. Елементи NumericUpDown і DomainUpDown

1.15.1. NumericUpDown

Елемент `NumericUpDown` представляє користувачеві вибір числа з певного діапазону. Для визначення діапазону чисел для вибору `NumericUpDown` має дві властивості: `Minimum` (задає мінімальне число) і `Maximum` (задає максимальне число).

Саме значення елемента зберігається у властивості `Value`:



За замовчуванням елемент відображає десяткові числа. Однак якщо встановити його властивість `Hexadecimal` рівною `true`, то елемент буде відображати всі числа в шістнадцятковій системі:

Навіть якщо в коді встановити звичайне десяткове значення:

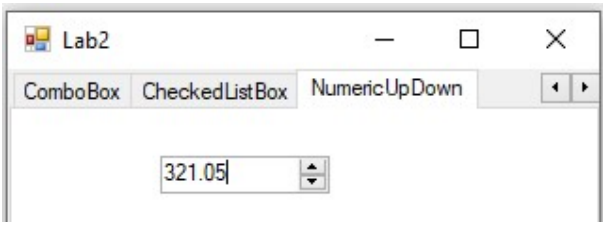
```
numericUpDown1.Value = 66;
```

то елемент всеодно відобразить його в шістнадцятковій системі.

Якщо потрібно відображати в полі дробові числа, то можна використати властивість `DecimalPlaces`, яке вказує, скільки знаків після коми має відображатись. За замовчуванням ця властивість дорівнює нулю.

Також можна задати відображення тисячного роздільника. Для цього для властивості `ThousandsSeparator` треба встановити значення `true`.

Наприклад, `numericUpDown` при `Value=321.05`, `DecimalPlaces=2` і `ThousandsSeparator=true` буде мати такий вигляд:



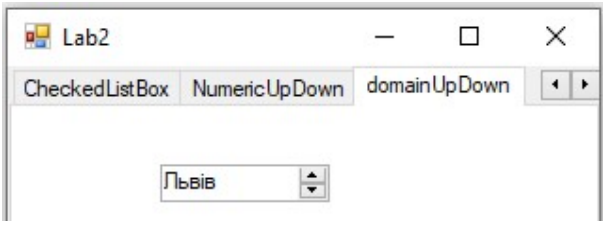
При цьому треба враховувати, що якщо встановити значення для властивості `Value` в вікні властивостей, то там як роздільник цілої і дробової частини використовується кома. Якщо ж встановити цю властивість у коді, тоді як роздільник використовується крапка.

За замовчуванням при натисканні на стрілочки вгору-вниз на елементі значення буде збільшуватися, або зменшуватися на одиницю. Але за допомогою властивості `Increment` можна задати інший крок збільшення, у тому числі і дробовий.

При роботі з `NumericUpDown` слід враховувати, що його властивість `Value` (як і властивості `Minimum` та `Maximum`) зберігає значення `decimal`. Тому в коді також потрібно з ним працювати як з `decimal`, а не як з типом `int` або `double`.

1.15.2. DomainUpDown

Елемент `DomainUpDown` призначений для введення текстової інформації. Він має текстове поле для введення рядка і дві стрілки для переміщення по списку рядків:



Список для DomainUpDown задається за допомогою властивості Items. Список можна відразу впорядкувати за алфавітом. Для цього треба властивості Sorted присвоїти значення true.

Щоб можна було циклічно переміщатися по списку, тобто при досягненні кінця або початку списку його перегляд починався з першого або останнього елемента, для властивості Wrap треба встановити значення true.

У коді вибране значення в DomainUpDown доступне через властивість Text. Наприклад, додамо програмно список рядків в DomainUpDown і опрацюємо зміну вибору в списку:

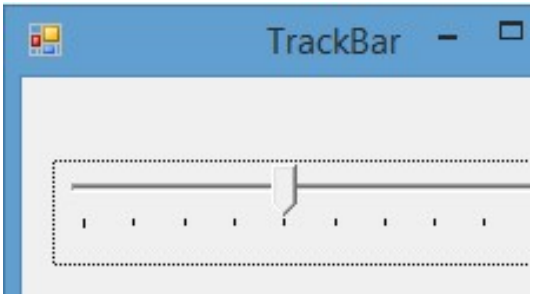
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        List<string> states = new List<string>
        {
            "Київ", "Львів", "Одеса", "Харків", "Дніпро"
        };

        // додаємо список елементів
        domainUpDown1.Items.AddRange(states);
        domainUpDown1.TextChanged += domainUpDown1_TextChanged;
    }
    // опрацювання зміни тексту в елементі
    void domainUpDown1_TextChanged(object sender, EventArgs e)
    {
        MessageBox.Show(domainUpDown1.Text);
    }
}
```

Для опрацювання зміни тексту, як і для елемента TextBox, можна використовувати подію TextChanged, в обробнику якого ввівши обраний текст в повідомлення.

1.16. TrackBar, Timer і ProgressBar

1.16.1. TrackBar

TrackBar є елементом, який за допомогою переміщення повзунка дає змогу вводити числові значення.



Деякі важливі властивості елемента `TrackBar`:

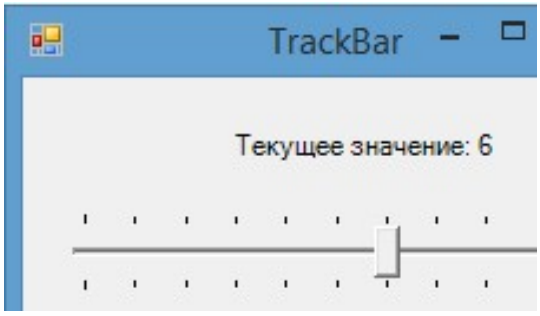
- `Orientation`: задає орієнтацію повзунка - розташування по горизонталі або по вертикалі;
- `TickStyle`: задає розташування поділок на повзунку;
- `TickFrequency`: задає частоту поділів на повзунку;
- `Minimum`: мінімальне значення на повзунку (за замовчуванням 0);
- `Maximum`: максимальне значення на повзунку (за замовчуванням 10);
- `Value`: поточне значення повзунка. Повинно бути між *Minimum* і *Maximum*.

Властивість `TickStyle` може приймати ряд значень:

- `None`: ділення немає;
- `Both`: ділення розташовані по обидва боки повзунка;
- `BottomRight`: у вертикального повзунка ділення знаходяться праворуч, а у горизонтального – знизу;
- `TopLeft`: у вертикального повзунка ділення знаходяться зліва, а у горизонтального - зверху (застосовується за замовчуванням).

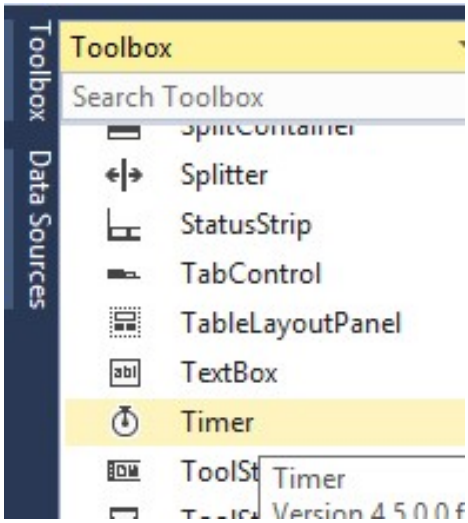
До найбільш важливих подій елемента слід віднести подію `Scroll`, яка дає змогу опрацювати переміщення повзунка від одного ділення до іншого. Що може бути корисно, якщо потрібно, наприклад, встановлювати відповідну гучність звуку в залежності від значення повзунка, або інші налаштування:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        // установка обробника події Scroll
        trackBar1.Scroll+=trackBar1_Scroll;
    }

    private void trackBar1_Scroll(object sender, EventArgs e)
    {
        label1.Text = String.Format("Поточне значення: {0}", trackBar1.Value);
    }
}
```



1.16.2. Timer

`Timer` є компонентом для запуску дій, що повторюються через певний проміжок часу. Хоча він не є візуальним елементом, але його також можна перетягнути з *Панелі Інструментів* на форму:



Найбільш важливі властивості і методи таймера:

- властивість `Enabled`: при значенні `true` вказує, що таймер буде запускатися разом із запуском форми;
- властивість `Interval`: вказує інтервал в мілісекундах, через який буде спрацьовувати обробник події `Tick`;
- метод `Start()`: запускає таймер;
- метод `Stop()`: зупиняє таймер.

Для прикладу додамо на форму кнопку і таймер. У файлі коду форми запишемо наступний код:

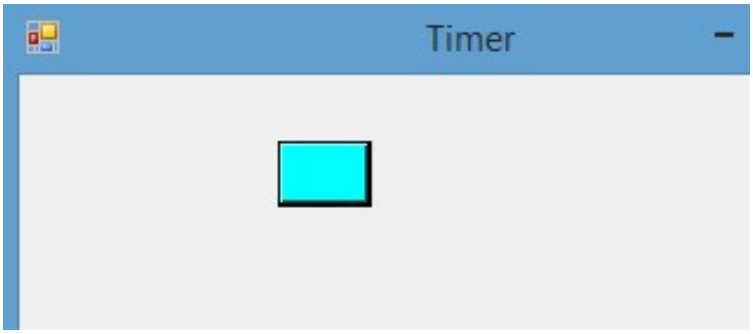
```
public partial class Form1 : Form
{
    int koef = 1;
    public Form1()
    {
        InitializeComponent();

        this.Width = 400;
```

```
        button1.Width = 40;
        button1.Left = 40;
        button1.Text = "";
        button1.BackColor = Color.Aqua;

        timer1.Interval = 500; // 500 мілісекунд
        timer1.Enabled = true;
        button1.Click += button1_Click;
        timer1.Tick += timer1_Tick;
    }
    // обробник події Tick таймера
    void timer1_Tick(object sender, EventArgs e)
    {
        if (button1.Left == (this.Width-button1.Width-10))
        {
            koef=-1;
        }
        else if (button1.Left == 0)
        {
            koef = 1;
        }
        button1.Left += 10 *koef;
    }
    // обробник натискання на кнопку
    void button1_Click(object sender, EventArgs e)
    {
        if(timer1.Enabled==true)
        {
            timer1.Stop();
        }
        else
        {
            timer1.Start();
        }
    }
}
```

Тут в конструкторі форми встановлюються початкові значення для таймера, кнопки і форми.

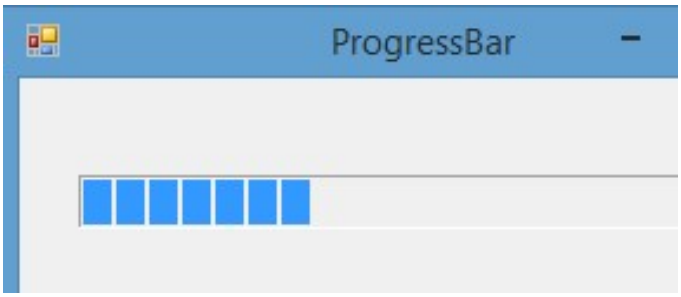
Через кожний інтервал таймера буде спрацьовувати обробник `timer1_Tick`, в якому змінюється положення кнопки по горизонталі за допомогою властивості `button1.Left`. А за допомогою додаткової змінної `koef` можна керувати напрямком руху.

Крім того, за допомогою обробника натискання кнопки `button1_Click` можна зупиняти таймер (і разом з ним рух кнопки), або знову його запускати.



1.16.3. Індикатор прогресу ProgressBar

Елемент `ProgressBar` використовується для того, щоб відобразити користувачеві інформацію про хід виконання будь-якої задачі.



Найбільш важливі властивості `ProgressBar`:

- `Minimum`: мінімальне значення;
- `Maximum`: максимальне значення;
- `Value`: поточне значення елемента;
- `Step`: крок, на який зміниться значення `Value` при виклику методу `PerformStep`.

Для імітації роботи `ProgressBar` поставимо на форму таймер, а в коді форми напишемо наступний код:

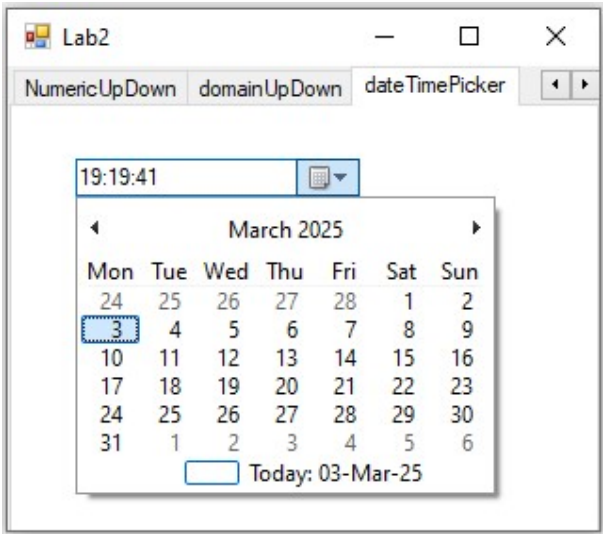
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        timer1.Interval = 500; // 500 мілісекунд
        timer1.Enabled = true;
        timer1.Tick += timer1_Tick;
    }
    // обробник події Tick таймера
    void timer1_Tick(object sender, EventArgs e)
    {
        progressBar1.PerformStep();
    }
}
```

1.17. DateTimePicker і MonthCalendar

Для роботи з датами в *Windows Forms* є елементи `DateTimePicker` і `MonthCalendar`.

1.17.1. DateTimePicker

`DateTimePicker` представляє календар, що розкривається при натисканні, в якому можна вибрати дату.



Найбільш важливі властивості `DateTimePicker`:

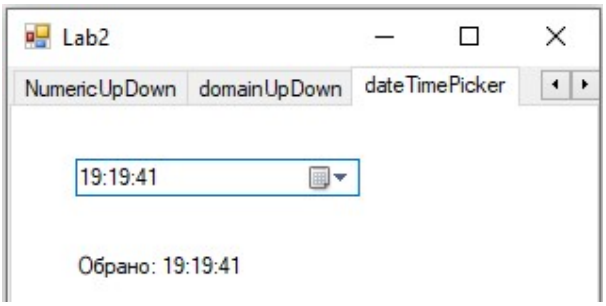
- `Format`: визначає формат відображення дати в елементі управління. Може приймати наступні значення:
  - `Custom`: формат задається розробником;
  - `Long`: повна дата;
  - `Short`: дата в скороченому форматі;
  - `Time`: формат для роботи з часом;
- `CustomFormat`: задає формат відображення дати, якщо для властивості `Format` встановлено значення `Custom;`
- `MinDate`: мінімальна дата, яку можна вибрати;
- `MaxDate`: найбільша дата, яку можна вибрати;
- `Value`: визначає поточне вибране значення в `DateTimePicker;`
- `Text`: представляє текст, що відображається в елементі.

При виборі дати елемент генерує подію `ValueChanged`. Наприклад, опрацюємо цю подію і присвоїмо вибране значення тексту мітки:

```
public partial class Form1 : Form
{
    public Form1()
    {
```

```
        InitializeComponent();
        dateTimePicker1.Format = DateTimePickerFormat.Time;
        dateTimePicker1.ValueChanged+=dateTimePicker1_ValueChanged;
    }

    private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
    {
        label1.Text = String.Format("Ви вибрали: {0}",
dateTimePicker1.Value.ToLongTimeString());
    }
}
```

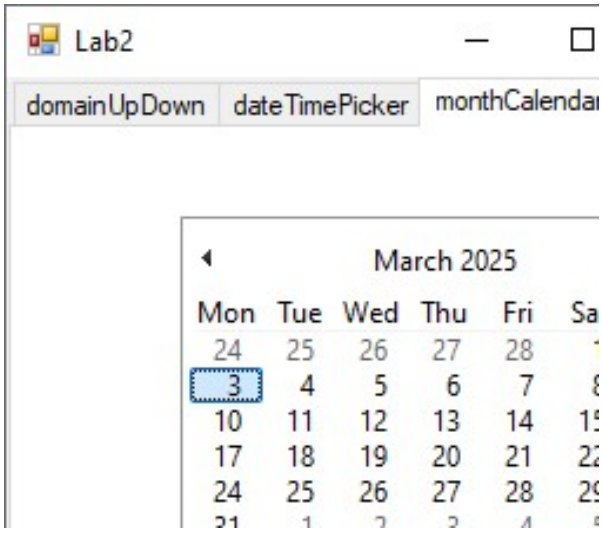
Властивість `Value` зберігає об'єкт `DateTime`, тому з ним можна працювати як і з будь-якою іншою датою. У даному випадку обрана дата перетворюється в рядок часу.



У цьому випадку значення `dateTimePicker1.Value.ToLongTimeString()` аналогічно тому тексту, який відображається в елементі. Тому можна написати так:
`label1.Text = String.Format("Обрано: {0}", dateTimePicker1.Text);`

1.17.2. MonthCalendar

За допомогою `MonthCalendar` також можна вибрати дату, тільки в даному випадку цей елемент представляє сам календар, який не треба розкривати:

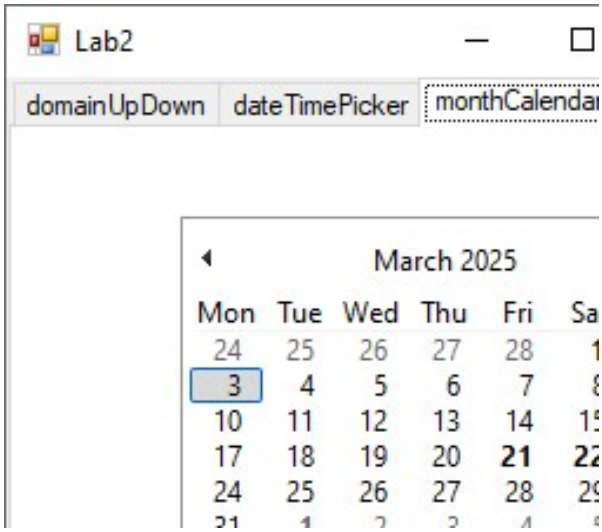


Розглянемо деякі основні властивості елемента.
Властивості виділення дат:

- `AnnuallyBoldedDates`: містить набір дат, які будуть виділені жирним в календарі для кожного року;
- `BoldedDates`: містить набір дат, які будуть виділені жирним (тільки для поточного року);
- `MonthlyBoldedDates`: містить набір дат, які будуть виділені жирним для кожного місяця.

Додавання виділених дат робиться за допомогою певних методів (як і видалення):

```
monthCalendar1.AddBoldedDate(new DateTime(2025, 03, 21));
monthCalendar1.AddBoldedDate(new DateTime(2025, 03, 22));
monthCalendar1.AddAnnuallyBoldedDate(new DateTime(2025, 03, 9));
monthCalendar1.AddMonthlyBoldedDate(new DateTime(2025, 03, 1));
```



Для зняття виділення можна використовувати аналоги цих методів:

```
monthCalendar1.RemoveBoldedDate(new DateTime(2025, 05, 21));
monthCalendar1.RemoveBoldedDate(new DateTime(2025, 05, 22));
monthCalendar1.RemoveAnnuallyBoldedDate(new DateTime(2025, 05, 9));
monthCalendar1.RemoveMonthlyBoldedDate(new DateTime(2025, 05, 1));
```

Властивості для визначення дат в календарі:

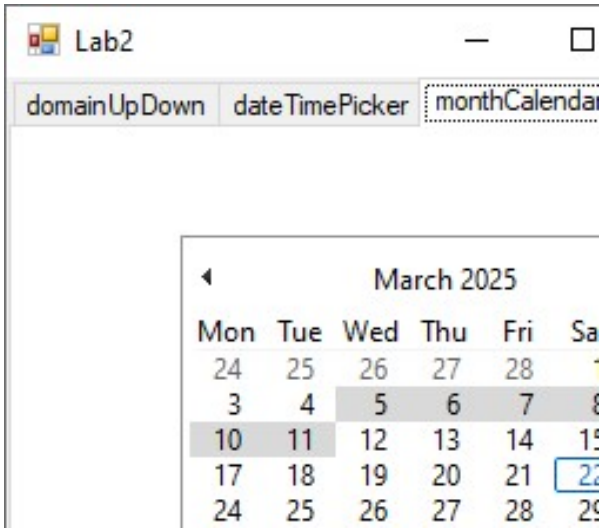
- `MinDate`: визначає мінімальну дату для вибору в календарі;
- `MaxDate`: визначає найбільшу дату для вибору в календарі;
- `FirstDayOfWeek`: визначає день тижня, з якого повинен починатися тиждень в календарі;
- `SelectionRange`: визначає діапазон виділених дат;
- `SelectionEnd`: визначає кінцеву дату виділення;
- `SelectionStart`: визначає початкову дату виділення;
- `ShowToday`: при значенні `true` відображає внизу календаря поточну дату;
- `ShowTodayCircle`: при значенні `true` поточна дата буде обведена кружечком;

- `TodayDate`: визначає поточну дату. За замовчуванням використовується системна дата на комп'ютері, але за допомогою даної властивості можна її змінити.

Наприклад, при встановленні властивостей:

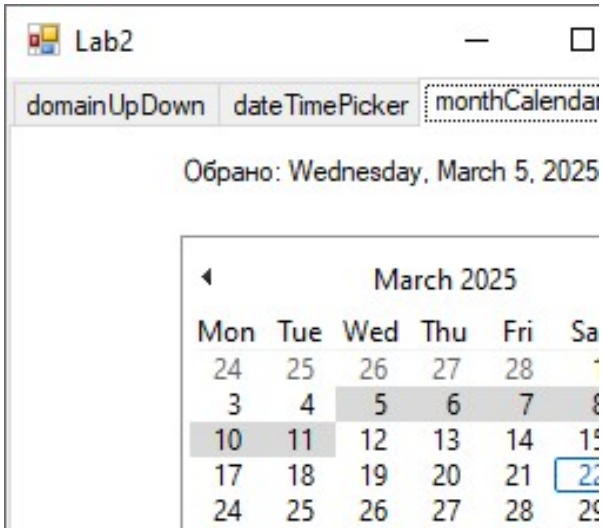
```
monthCalendar1.TodayDate= new DateTime(2025, 03, 22);
monthCalendar1.ShowTodayCircle = true;
monthCalendar1.ShowToday = false;
monthCalendar1.SelectionStart = new DateTime(2025, 03, 1);
monthCalendar1.SelectionEnd = new DateTime(2025, 03, 11);
```

буде наступне відображення календаря:



Найбільш цікавими подіями елемента є події `DateChanged` і `DateSelected`, які виникають при зміні дати, обраної в елементі. Однак треба враховувати, що обрана дата буде представляти першу дату з діапазону виділених дат:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        monthCalendar1.DateChanged += monthCalendar1_DateChanged;
    }

    void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
    {
        label1.Text = String.Format("Обрано: {0}", e.Start.ToLongDateString());
        // або так - аналогічний код
        //label1.Text = String.Format("Обрано: {0}",
monthCalendar1.SelectionStart.ToLongDateString());
    }
}
```

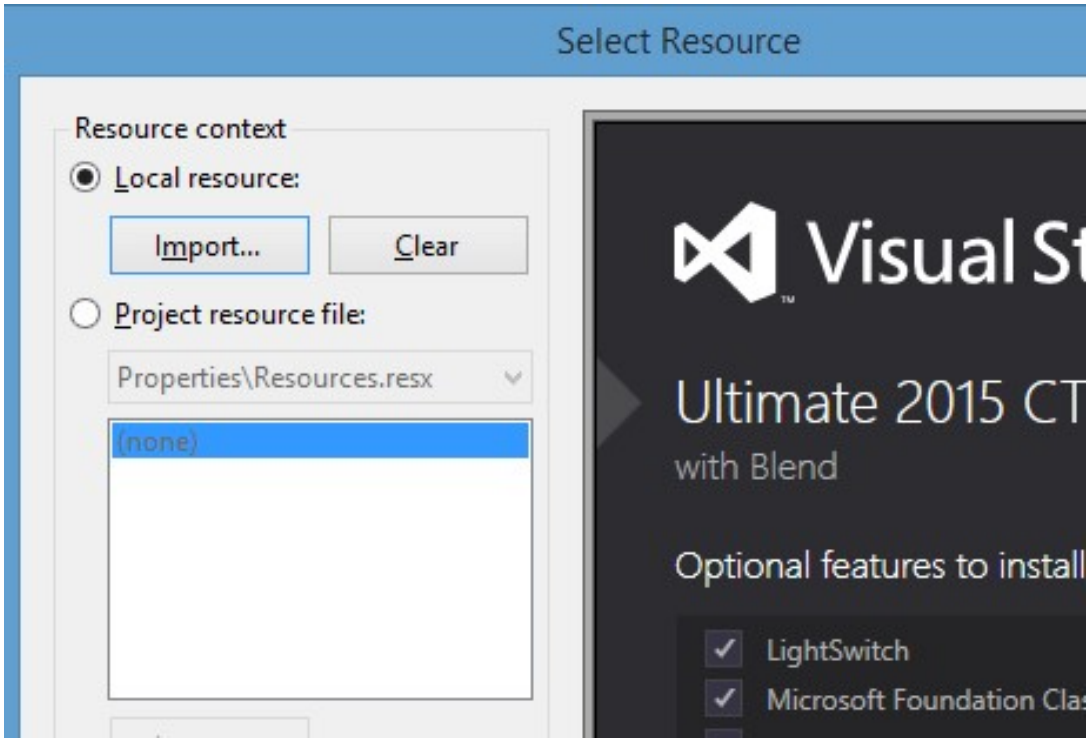


1.18. Елемент PictureBox

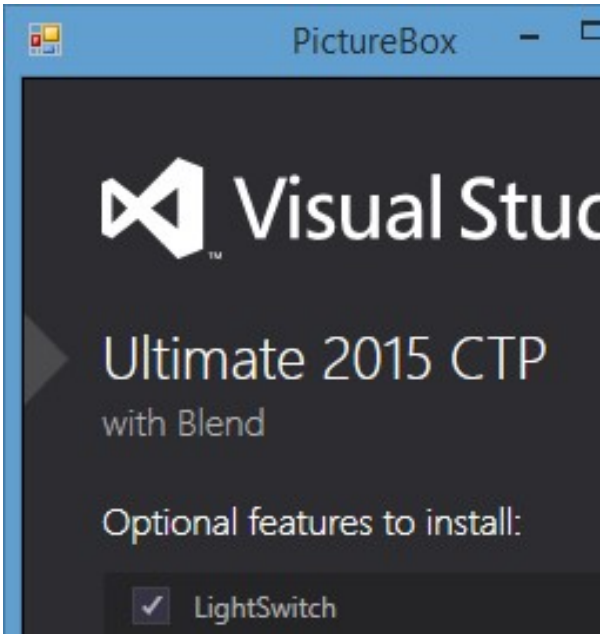
`PictureBox` призначений для відображення зображень. Він дає змогу відобразити файли у форматі *bmp, jpg, gif*, а також метафайли зображень та піктограми. Щоб вибрати зображення `PictureBox` можна використовувати ряд властивостей:

- `Image`: встановлює об’єкт типу `Image`;
- `ImageLocation`: встановлює шлях до зображення на диску або в Інтернеті;
- `InitialImage`: деяке початкове зображення, яке буде відображатися під час завантаження головного зображення, що зберігається у властивості `Image`;
- `ErrorImage`: зображення, яке відображається, якщо основне зображення не вдалося завантажити в `PictureBox`.

Щоб встановити зображення в *Visual Studio*, треба в панелі властивостей `PictureBox` вибрати властивість `Image`. Відкриється вікно імпорту зображення в проект, де власне і можна вибрати потрібне зображення на комп’ютері і встановити його для `PictureBox`:



Після цього це зображення можна побачити в `PictureBox`:



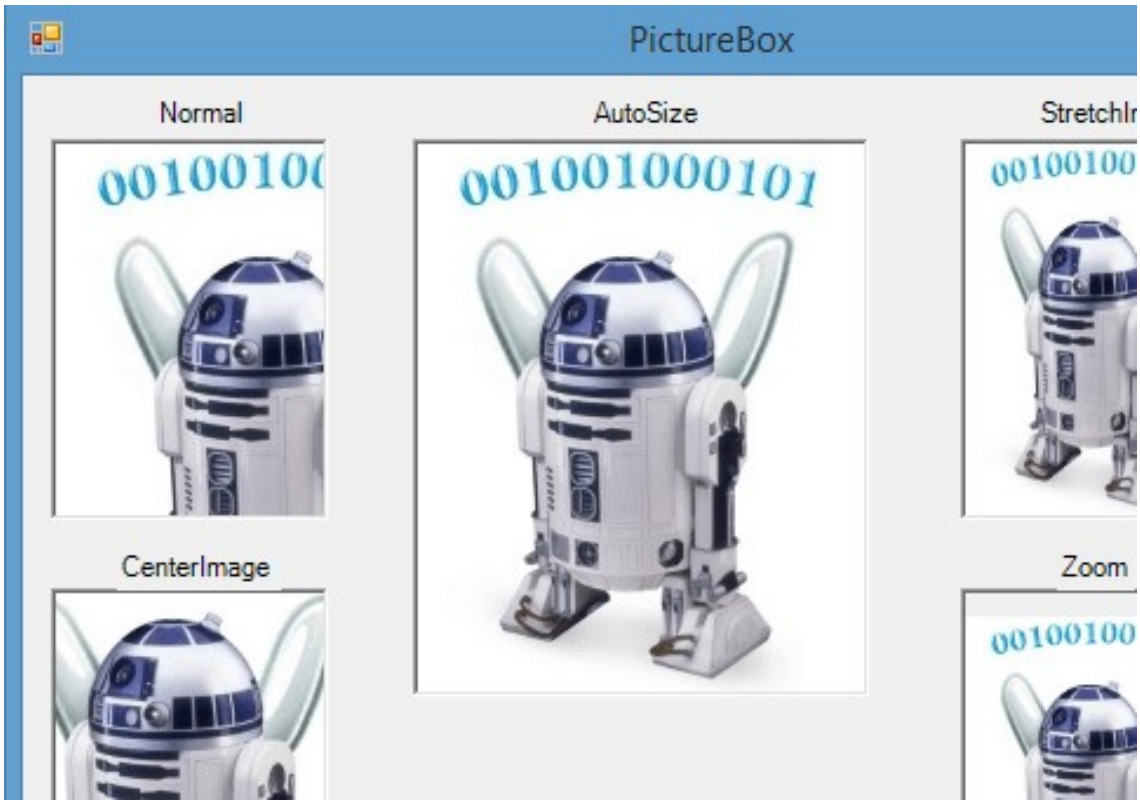
Або можна завантажити зображення в коді:

```
pictureBox1.Image = Image.FromFile("D:\Pictures\12.jpg");
```

1.18.1. Розмір зображення

Щоб вибрати зображення `PictureBox` використовується властивість `SizeMode`, яка має такі значення:

- `Normal`: зображення позиціонується в лівому верхньому кутку `PictureBox`, і розмір зображення не змінюється. Якщо `PictureBox` більше за розміри зображення, то справа і знизу з'являються порожнечі, якщо менше - то зображення обрізається;
- `StretchImage`: зображення розтягується або стискається таким чином, щоб вміститися по всій ширині і висоті елемента `PictureBox;`
- `AutoSize`: елемент `PictureBox` автоматично розтягується, підлаштовуючись під розміри зображення;
- `CenterImage`: якщо `PictureBox` менше зображення, то зображення обрізається по краях і виводиться тільки його центральна частина. Якщо ж `PictureBox` більше зображення, то воно розташовується по центру;
- `Zoom`: зображення масштабується під розміри `PictureBox`, зберігаючи при цьому пропорції.



1.19. Елемент NotifyIcon

Елемент `NotifyIcon` дає змогу задати піктограму, яка буде відображатися під час запуску програми в панелі завдань.

Розглянемо основні його властивості:

- `BallonTipIcon`: піктограма, що буде використовуватись у підказці. Ця властивість може мати наступні значення: `None`, `Info`, `Warning`, `Error`;
- `BalloonTipText`: текст, що відображається у підказці;
- `BalloonTipTitle`: загаловок підказки;

- `ContextMenuStrip`: встановлює контекстне меню для об'єкта `NotifyIcon`;
- `Icon`: задає піктограму, яка буде відображатися в системному треї;
- `Text`: встановлює текст підказки, що з'являється при знаходженні вказівника миші над позначкою;
- `Visible`: встановлює видимість піктограми в системному треї.

Щоб додати на форму елемент `NotifyIcon`, перенесемо даний елемент на форму з панелі інструментів. Після цього компонент `NotifyIcon` відобразиться внизу дизайнера форми.

Потім встановимо для властивості `Icon` елемента `NotifyIcon` певну піктограму в форматі `.ico`. І також встановимо для властивості `Visible` значення `true`. Далі для властивості `Text` задамо певний текст, наприклад, "Показати форму". Цей текст відобразиться при проходженні вказівника миші над позначкою `NotifyIcon` в системному треї.

Для відкриття форму при натисканні на піктограмі в треї, потрібно опрацювати подію `Click`. В коді форми визначимо обробник для цієї події:

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

        this.ShowInTaskbar = false;
        notifyIcon1.Click += notifyIcon1_Click;
    }

    void notifyIcon1_Click(object sender, EventArgs e)
    {
        this.WindowState = FormWindowState.Normal;
    }
}
```

В обробнику просто переводимо форму з мінімізованого стану в звичайний. Крім того, щоб форма не відображалася на панелі завдань, задаємо її властивість `ShowInTaskbar = false`.

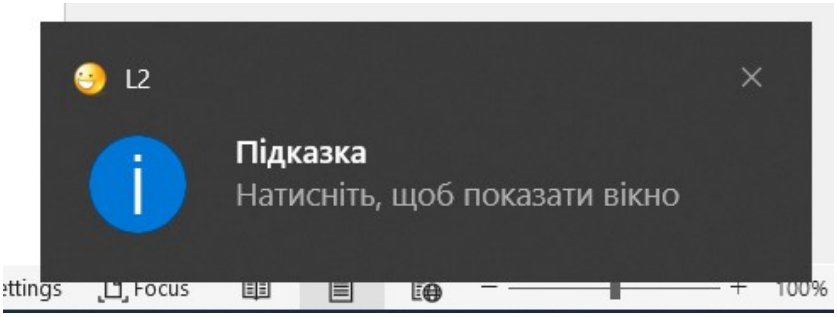
Після запуску програми в треї буде відображатися піктограма `NotifyIcon`, натиснувши на яку при згорнутій формі, можна заново її відкрити.

Тепер використовуємо підказку. Для цього змінимо конструктор форми:

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

        this.ShowInTaskbar = false;
        notifyIcon1.Click += notifyIcon1_Click;
```

```
        // задаємо піктограму підказки
        notifyIcon1.BalloonTipIcon = ToolTipIcon.Info;
        // задаємо текст підказки
        notifyIcon1.BalloonTipText = "Натисніть, щоб показати вікно";
        // встановлюємо заголовок
        notifyIcon1.BalloonTipTitle = "Підказка";
        // відображаємо підказку 12 секунд
        notifyIcon1.ShowBalloonTip(12);
    }

    void notifyIcon1_Click(object sender, EventArgs e)
    {
        this.WindowState = FormWindowState.Normal;
    }
}
```

При запуску відобразиться підказка:



1.20. OpenFileDialog і SaveFileDialog

Вікна відкриття і збереження файлу представлені класами `OpenFileDialog` і `SaveFileDialog`. Вони мають багато в чому схожу функціональність, тому розглянемо їх разом.

Компоненти `OpenFileDialog` і `SaveFileDialog` мають ряд загальних властивостей, серед яких можна виділити наступні:

- `DefaultExt`: встановлює розширення файлу, яке додається за замовчуванням, якщо користувач ввів ім'я файлу без розширення;
- `AddExtension`: при значенні `true` додає до імені файлу розширення при його відсутності. Розширення береться з властивості `DefaultExt` або `Filter`;
- `CheckFileExists`: якщо має значення `true`, то перевіряє наявність файлу з вказаним ім'ям;
- `CheckPathExists`: якщо має значення `true`, то перевіряє існування шляху до файлу з вказаним ім'ям;
- `FileName`: повертає повне ім'я файлу, обраного в діалоговому вікні;
- `Filter`: задає фільтр файлів, завдяки чому в діалоговому вікні можна відфільтрувати файли по розширенню. Фільтр задається в наступному форматі `Назва_файлів | *.розширення`. Наприклад, *Текстові файли (*.Txt)* | *.txt. Можна задати відразу кілька фільтрів, для цього вони відокремлюються

вертикальною лінією |. Наприклад, *Bitmap files (*.bmp)* | *.bmp | *Image files (*.jpg)* | *.jpg;

- `InitialDirectory`: встановлює каталог, який відображається під час першого виклику вікна;
- `Title`: заголовок діалогового вікна.

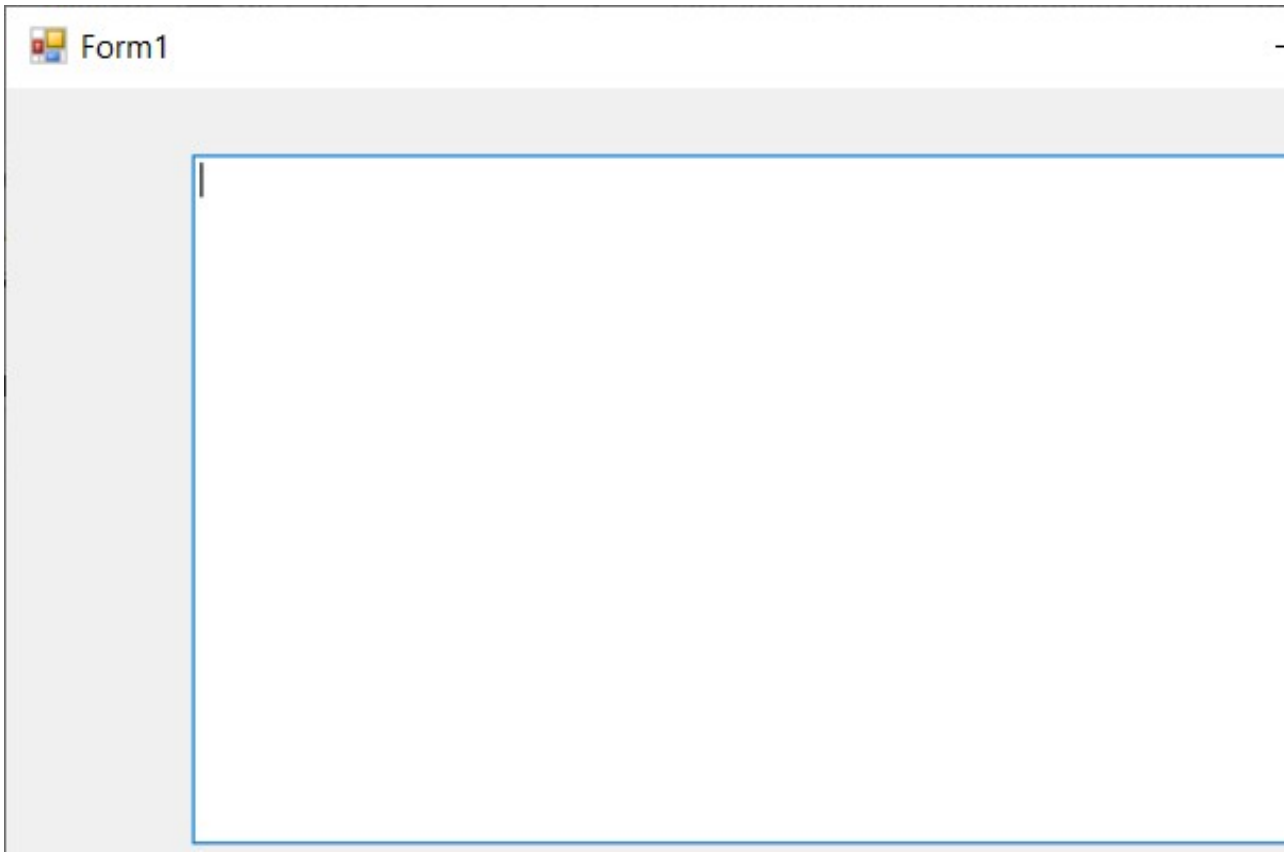
Для `SaveFileDialog` можна виділити ще декілька властивостей:

- `CreatePrompt`: при значенні `true` у випадку, якщо вказаний неіснуючий файл, буде відображатися повідомлення про його створення;
- `OverwritePrompt`: при значенні `true` у випадку, якщо вказаний існуючий файл, буде відображатися повідомлення про те, що файл буде перезаписано.

Щоб відобразити діалогове вікно, треба викликати метод `ShowDialog()`.

Розглянемо обидва діалогових вікна на прикладі.

Додамо на форму текстове поле `textBox1` і дві кнопки `button1` і `button2`, які назвемо “Відкрити” та “Зберегти” відповідно. Також перетягнемо з панелі інструментів компоненти `OpenFileDialog` і `SaveFileDialog`. Після додавання вони з’являться внизу дизайнера форми. У результаті форма буде виглядати приблизно так:



Тепер змінимо код форми:

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

```
{
    public Form1()
    {
        InitializeComponent();

        button1.Click += button1_Click;
        button2.Click += button2_Click;
        openFileDialog1.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
        saveFileDialog1.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
    }
    // збереження файлу
    void button2_Click(object sender, EventArgs e)
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // отримуємо обраний файл
        string filename = saveFileDialog1.FileName;
        // зберігаємо текст в файл
        System.IO.File.WriteAllText(filename, textBox1.Text);
        MessageBox.Show("Файл збережений ");
    }
    // відкриття файлу
    void button1_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // отримуємо обраний файл
        string filename = openFileDialog1.FileName;
        // читаємо файл в рядок
        string fileText = System.IO.File.ReadAllText(filename);
        textBox1.Text = fileText;
        MessageBox.Show("Файл відкритий");
    }
}
```

При натисканні на першу кнопку буде відкриватися вікно відкриття файлу. Після вибору файлу він буде зчитуватися, а його текст буде відображатися в текстовому полі. Натискання на другу кнопку відобразить вікно для збереження файлу, в якому треба вказати його назву. Після цього відбудеться збереження тексту з текстового поля в файл.

1.21. FontDialog і ColorDialog

1.21.1. FontDialog

Для вибору шифту і його параметрів використовується компонент `FontDialog`.

`FontDialog` має ряд властивостей, серед яких варто відзначити наступні:

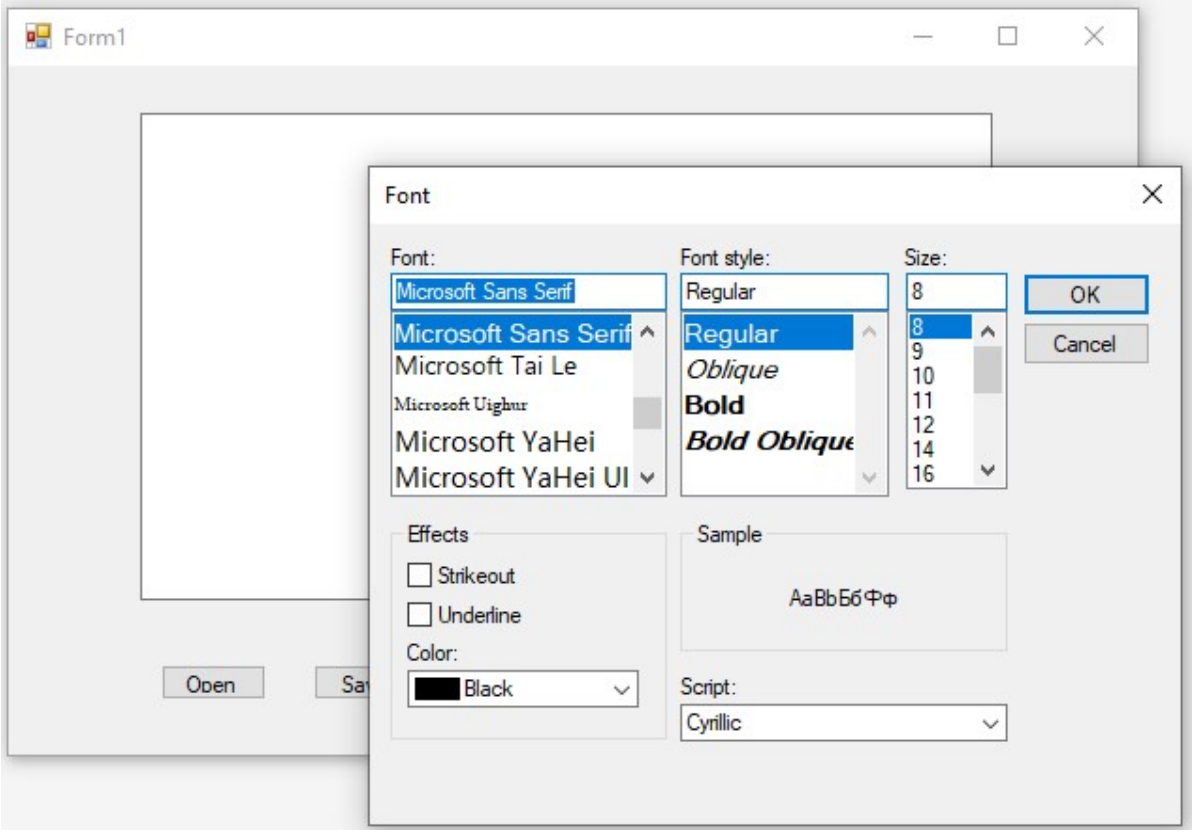
- `ShowColor`: при значенні `true` дає змогу вибирати колір шрифту;

- `Font`: вибраний в діалоговому вікні шрифт;
- `Color`: вибраний в діалоговому вікні колір шрифту;
- Для відображення діалогового вікна використовується метод `ShowDialog()`.

Для використання цього компонент перенесемо його з панелі інструментів на форму. Крім того, нехай на формі є кнопка `button1`. Тоді в коді форми запишемо наступне:

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

        button1.Click += button1_Click;
        // додаємо можливість вибору кольору шрифту
        fontDialog1.ShowColor = true;
    }

    void button1_Click(object sender, EventArgs e)
    {
        if (fontDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // установка шрифту
        button1.Font = fontDialog1.Font;
        // установка кольору шрифту
        button1.ForeColor = fontDialog1.Color;
    }
}
```

Якщо запустити програму і натиснути на кнопку, то відобразиться діалогове вікно, де можна вибрати всі параметри шрифту. Після підтвердження встановлені налаштування будуть застосовані до шрифту кнопки:



1.21.2. ColorDialog

Компонент `ColorDialog` дає змогу вибрати налаштування кольору.

Серед властивостей `ColorDialog` слід виділити наступні:

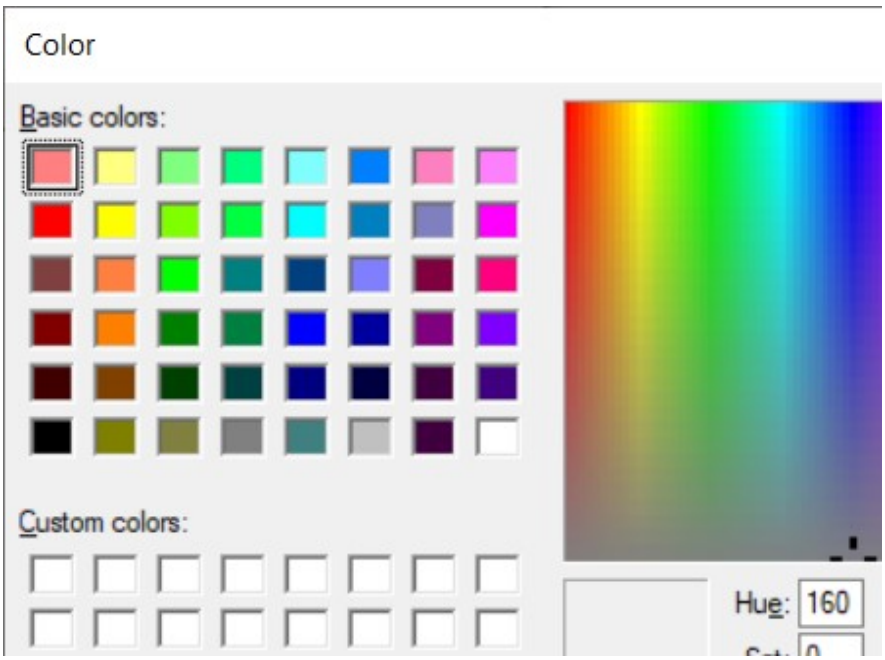
- `FullOpen`: при значенні `true` відображається діалогове вікно з розширеними налаштуваннями для вибору кольору;
- `SolidColorOnly`: при значенні `true` дає змогу вибирати тільки між однотонними відтінками кольорів;
- `Color`: вибраний в діалоговому вікні колір.

Перенесемо його з Панелі інструментів на форму. І змінимо код форми:

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

        button1.Click += button1_Click;
        // розширене вікно для вибору кольору
        colorDialog1.FullOpen = true;
        // установка початкового кольору для colorDialog
        colorDialog1.Color = this.BackColor;
    }
```

```
    void button1_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.Cancel)
            return;
        // встановлення кольору форми
        this.BackColor = colorDialog1.Color;
    }
}
```

При натисканні кнопки на відобразиться діалогове вікно, в якому можна встановити колір форми:



1.22. Меню MenuStrip

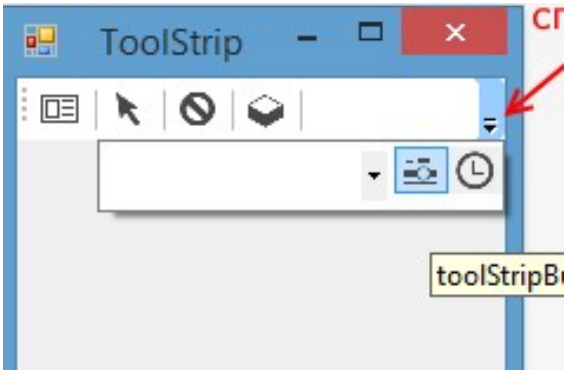
Для створення меню в *Windows Forms* застосовується елемент `MenuStrip`. Даний клас успадкований від `ToolStrip` і тому успадковує його функціональність.

Найбільш важливі властивості компонента `MenuStrip`:

- `Dock`: прикріплює меню до однієї зі сторін форми;
- `LayoutStyle`: задає орієнтацію панелі меню на формі. Ця властивість може набувати наступних значень:
  - `HorizontalStackWithOverflow`: розташування 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 розташовуються автоматично з переповненням;
  - `VerticalStackWithOverflow`: елементи розташовуються вертикально з переповненням;

  - `Flow`: елементи розміщуються автоматично, але без переповнення - якщо довжина панелі меню менше довжини контейнера, то елементи, що виходять за межі, переносяться;
  - `Table`: елементи позиціонуються у вигляді таблиці;
- `ShowItemToolTips`: вказує, чи будуть відображатися спливаючі підказки для окремих елементів меню;
- `Stretch`: дає змогу розтягнути панель по всій довжині контейнера;
- `TextDirection`: задає напрямок тексту в пунктах меню.

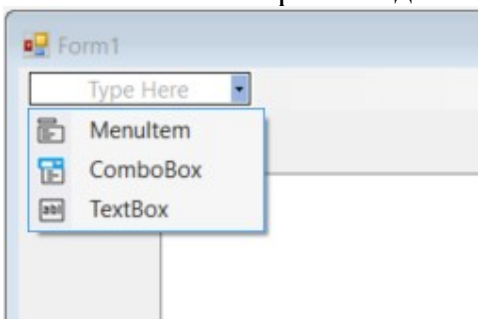
Якщо `LayoutStyle` має значення `HorizontalStackWithOverflow` / `VerticalStackWithOverflow`, то за допомогою властивості `CanOverflow` можна задати поведінку при переповненні. Так, якщо це властивість дорівнює `true` (значення за замовчуванням), то для елементів, які не потрапляють в межі `ToolStrip`, створюється список, що випадає:



При значенні `false` подібний випадаючий список не створюється.

`MenuStrip` є певним контейнером для окремих пунктів меню, що представлені об’єктом `ToolStripMenuItem`.

Додати нові елементи в меню можна в режимі дизайнера:



Для додавання є три види елементів: `MenuItem` (об’єкт `ToolStripMenuItem`), `ComboBox` і `TextBox`. Таким чином, в меню можна використовувати списки, що випадають і текстові поля, однак, переважно, ці елементи застосовуються в основному на панелі інструментів. Меню ж, переважно, містить набір об’єктів `ToolStripMenuItem`.

Також можна додати пункти меню в коді:

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

```
    public Form1()
    {
        InitializeComponent();

        ToolStripMenuItem fileItem = 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");
    }
}
```

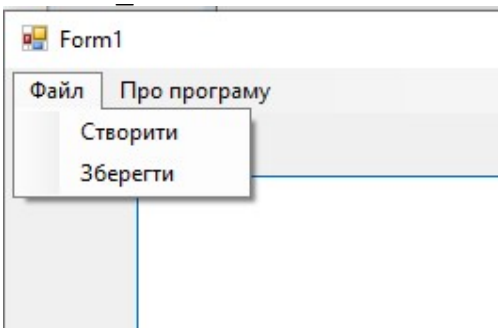
`ToolStripMenuItem` в конструкторі приймає текстову мітку, яка буде використовуватися як текст меню. Кожний подібний об'єкт має колекцію `DropDownItems`, яка зберігає дочірні об'єкти `ToolStripMenuItem`. Тобто один елемент `ToolStripMenuItem` може містити набір інших об'єктів `ToolStripMenuItem`. Таким чином, утворюється ієрархічне меню або структура у вигляді дерева.

Якщо передати при додаванні рядок тексту, то для нього неявним чином буде створено об'єкт:

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

Призначивши обробники для події `Click`, можна опрацювати натискання на пункти меню:

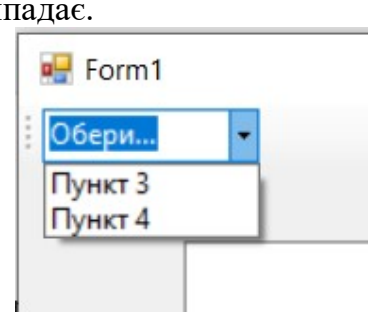
```
aboutItem.Click += aboutItem_Click
```



Панель `ToolStrip` може містити об'єкти наступних класів:

- `ToolStripLabel`: текстова мітка на панелі інструментів, представляє функціональність елементів `Label` і `LinkLabel`;

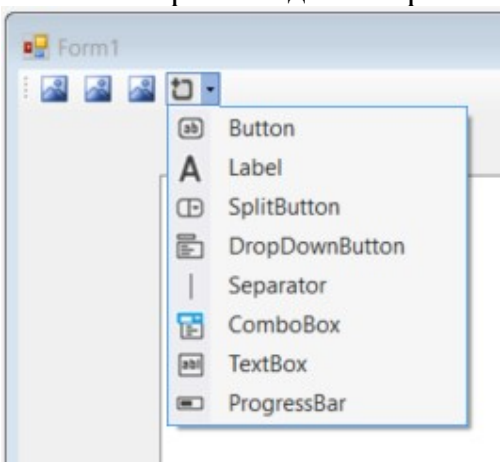
- `ToolStripButton`: аналогічний елементу `Button`. Так само має подію `Click`, за допомогою якого можна обробити натискання користувача на кнопку;
- `ToolStripSeparator`: візуальний роздільник між іншими елементами на панелі інструментів;
- `ToolStripToolStripComboBox`: подібний до стандартного елементу `ComboBox`;
- `ToolStripTextBox`: аналогічний текстовому полю `TextBox`;
- `ToolStripProgressBar`: індикатор прогресу, як і елемент `ProgressBar`;
- `ToolStripDropDownButton`: представляє кнопку, після натискання на яку відкривається меню, що випадає.



До кожного елементу меню, що випадає додатково можна прикріпити обробник натискання і обробити клік по цим пунктам меню:

- `ToolStripSplitButton`: об'єднує функціональність `ToolStripDropDownButton` і `ToolStripButton`

Додати нові елементи можна в режимі дизайнера:



Також можна додавати нові елементи програмно в коді. Їх розташування на панелі інструментів буде відповідати порядку додавання. Всі елементи зберігаються в `ToolStrip` у властивості `Items`. Можна додати будь-який об’єкт класу `ToolStripItem` (тобто будь-який з вище перерахованих класів, оскільки вони успадковуються від `ToolStripItem`):

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

        ToolStripButton clearBtn = new ToolStripButton();
```

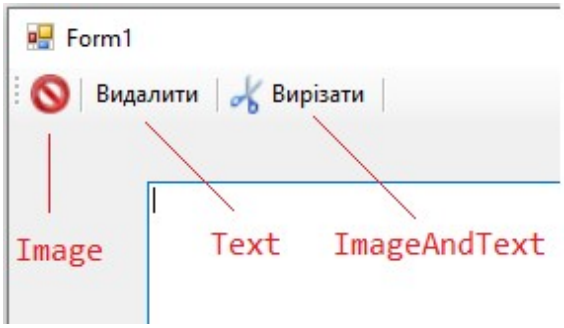
```
        clearBtn.Text = "Очистити";
        // встановлюємо обробник натискання
        clearBtn.Click += btn_Click;
        toolStrip1.Items.Add(clearBtn);
    }

    void btn_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Виконується видалення");
    }
}
```

Крім того, тут задається обробник, що дозволяє обробляти натискання на кнопки на панелі інструментів.

Елементи `ToolStripButton`, `ToolStripDropDownButton` і `ToolStripSplitButton` можуть відображати як текст, так і зображення, або відразу і те, і інше. Для керування розміщенням зображень в цих елементах є такі властивості:

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

Щоб розмістити зображення на кнопці, у властивості `DisplayStyle` треба встановити значення `Image`. Якщо потрібно, щоб кнопка відображала тільки текст, то треба вказати значення `Text`, або можна комбінувати два значення за допомогою іншого значення `ImageAndText`:



Всі ці значення зберігаються в переліку `ToolStripItemDisplayStyle`. Також можна встановити властивості в коді:

```
ToolStripButton clearBtn = new ToolStripButton();
clearBtn.Text = "Пошук";
clearBtn.DisplayStyle = ToolStripItemDisplayStyle.ImageAndText;
clearBtn.Image = Image.FromFile(@"D:\Icons\0023\search32.png");
// додаємо на панель інструментів
toolStrip1.Items.Add(clearBtn);
```

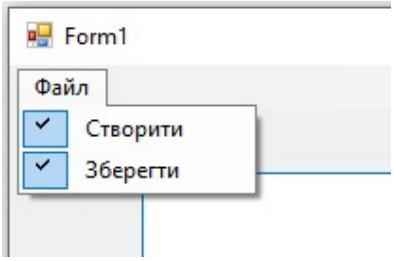
1.22.1. Відмітки пунктів меню

Властивість `CheckOnClick` при значенні `true` дає змогу на кліку відмітити пункт меню. А за допомогою властивості `Checked` можна встановити, чи буде пункт меню відмічений при запуску програми.

Ще одна властивість `CheckState` повертає стан пункту меню - відмічений він чи ні. Воно може приймати три значення: `Checked` (відмічений), `Unchecked` (не відмічений) і `Indeterminate` (в невизначеному стані)

Наприклад, створимо ряд зазначених пунктів меню і опрацюємо подію встановлення/зняття позначки:

```
public partial 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);
    }

    void menuItem_CheckedChanged(object sender, EventArgs e)
    {
        ToolStripMenuItem menuItem = sender as ToolStripMenuItem;
        if (menuItem.CheckState == CheckState.Checked)
            MessageBox.Show("Відмічений");
        else if (menuItem.CheckState == CheckState.Unchecked)
            MessageBox.Show("Не відмічений");
    }
}
```



1.22.2. Клавіші швидкого доступу

Якщо потрібно швидко звернутися до певного пункту меню, то можна використовувати клавіші швидкого доступу. Для встановлення клавіш швидкого доступу використовується властивість `ShortcutKeys`:

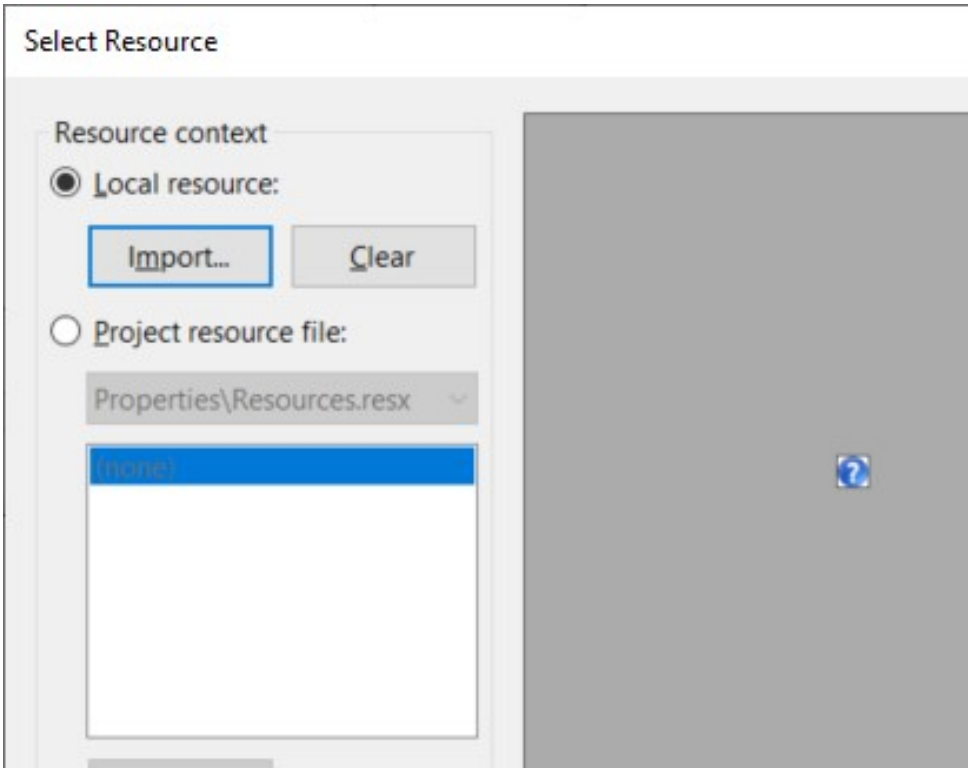
```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        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 переліку `Keys`. У даному випадку після натискання на комбінацію клавіш `Ctrl + P`, буде спрацьовувати натискання на пункт меню `"Зберегти"`.

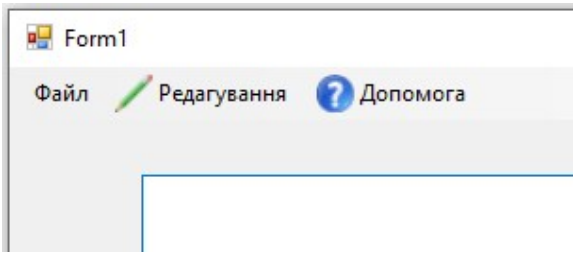
За допомогою зображень можна урізноманітнити зовнішній вигляд пунктів меню. Для цього можна використовувати такі властивості:

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

Якщо зображення для пункту меню встановлюється в режимі дизайнера, то потрібно вибрати у вікні властивість пункт `Image`, після чого відкриється вікно для імпорту ресурсу зображення в проект:



Щоб задати розміщення зображення, у властивості `DisplayStyle` треба встановити значення `Image`. Якщо потрібно, щоб кнопка відображала тільки текст, треба вказати значення `Text`, або можна комбінувати два значення за допомогою іншого значення `ImageAndText`. За замовчуванням зображення розміщується зліва від тексту:



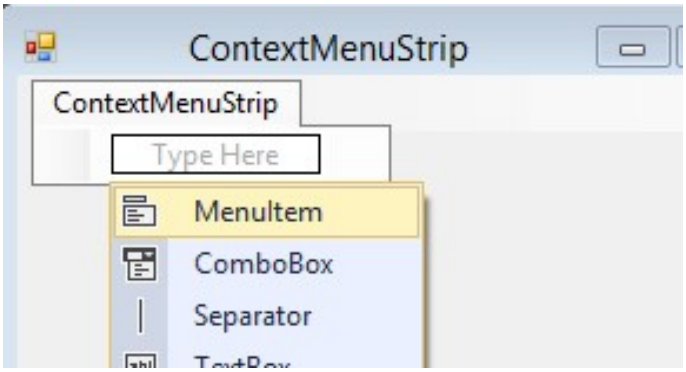
Також можна встановити зображення дінамічно в коді:

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

1.23. Контекстне меню ContextMenuStrip

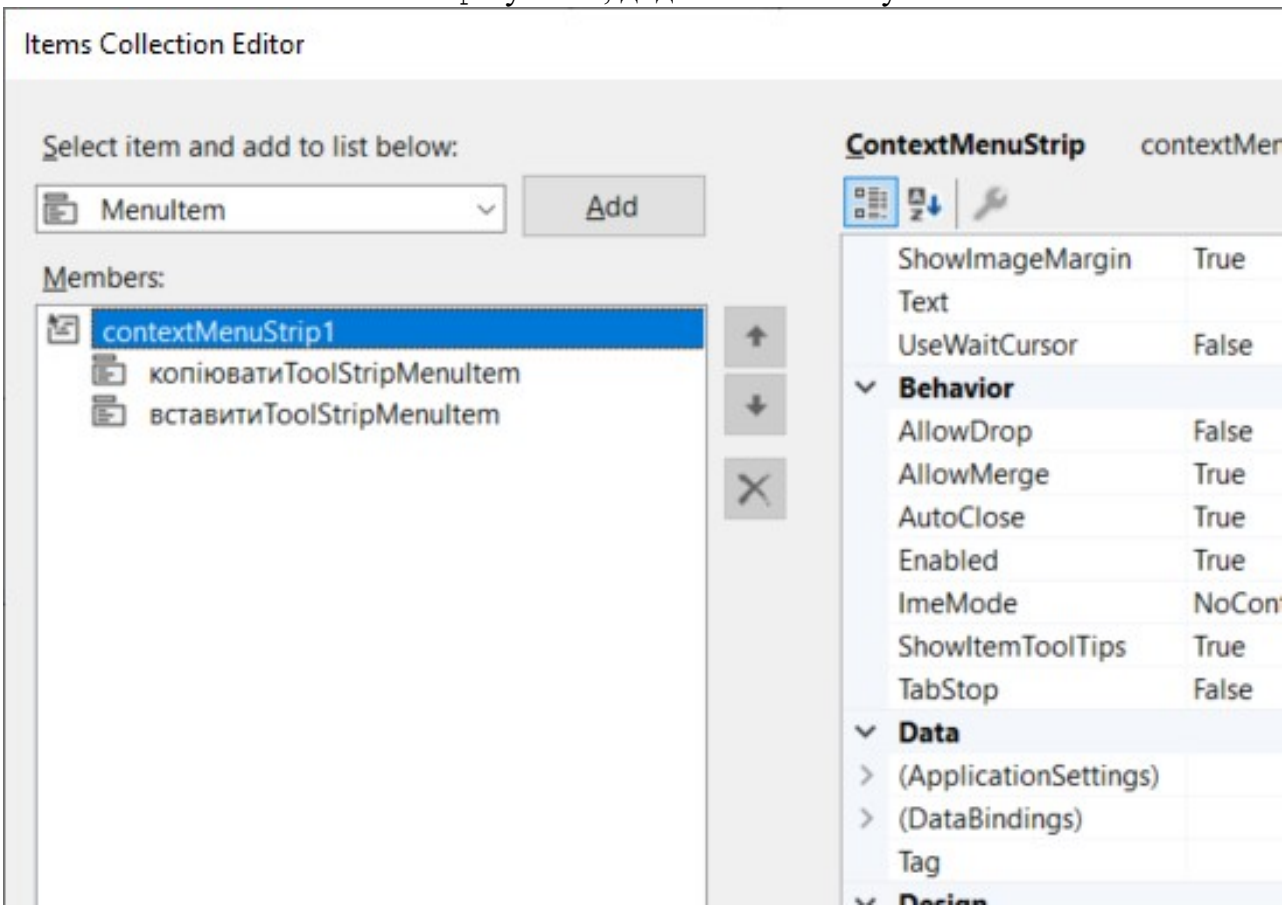
Компонент `ContextMenuStrip` представляє контекстне меню, він дуже схожий до елемента `MenuStrip` за винятком того, що контекстне меню не може використовуватись саме по собі, воно обов'язково застосовується до певного іншого елемента, наприклад, текстового поля.

Нові елементи в контекстне меню можна додати в режимі дизайнера:



В компонент `ContextMenuStrip` можна додати всі ті ж елементи, що і в `MenuStrip`. Але, як правило, використовується елемент `ToolStripMenuItem`, або елемент `ToolStripSeparator`, що представляє горизонтальну смужку роздільник між іншими пунктами меню.

Крім того, на панелі властивостей можна звернутися до властивості `Items` компонента `ContextMenuStrip` і у вікні, додати та налаштувати всі елементи меню:



Тепер створимо невелику програму. Додамо на форму елементи `ContextMenuStrip` і `TextBox`, які матимуть назви `contextMenuStrip1` і `textBox1` відповідно. Потім змінимо код форми наступним чином:

```
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 з 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(object sender, EventArgs e)
    {
        // якщо виділений текст в текстовому полі, то копіюємо його в буфер
        buffer = textBox1.SelectedText;
    }
}
```

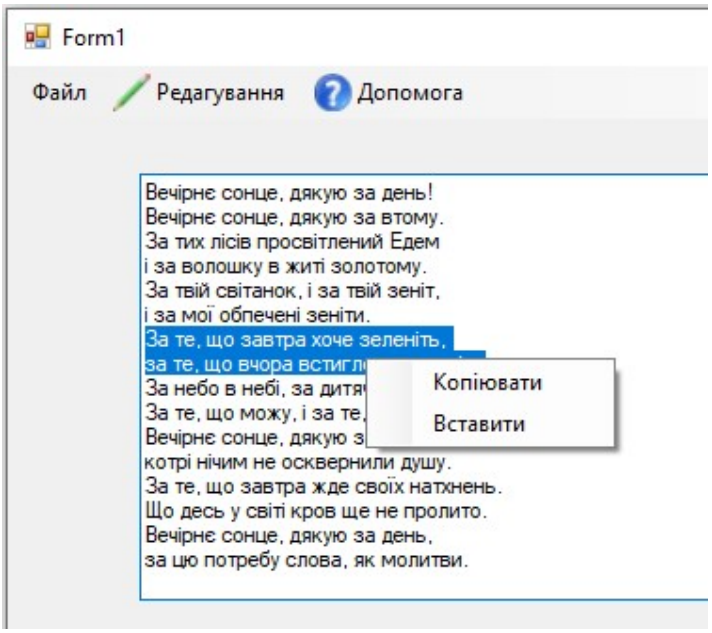
У цьому випадку виконана найпростіша реалізація функціональності `copy-paste`. У меню додається два елементи. А у текстового поля встановлюється багаторядковість, і воно розтягується по ширині контейнера.

У багатьох компонентів є властивість `ContextMenuStrip`, яка дає змогу асоціювати контекстне меню з даним елементом. У випадку з `TextBox` асоціація відбувається наступним чином:

```
textBox1.ContextMenuStrip = contextMenuStrip1
```

І після натискання на текстове поле правою кнопкою миші можна викликати асоційоване контекстне меню.

За допомогою обробників натискання пунктів меню встановлюються копіювання та вставлення рядків.



1.24. Стрічка стану StatusStrip

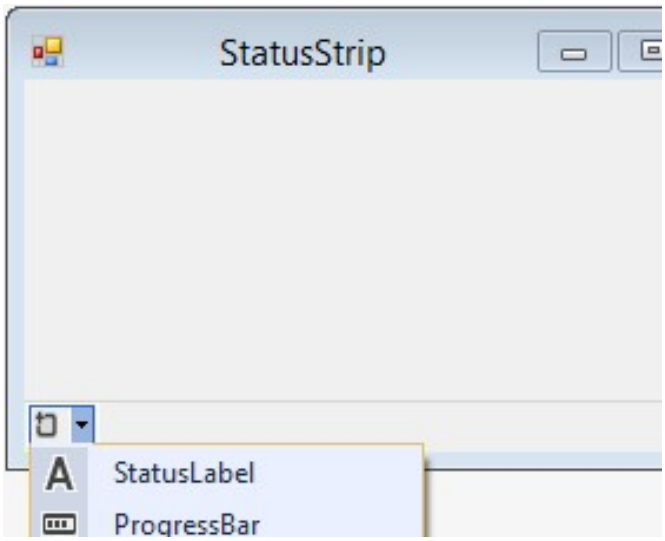
`StatusStrip` є рядком стану, схожим до панелі інструментів `ToolStrip`. Рядок стану призначений для відображення поточної інформації про стан роботи програми.

При додаванні на форму `StatusStrip` автоматично розташовується в нижній частині вікна програми (як і в більшості додатків). Однак при необхідності його можна позиціонувати інакше, керуючи властивістю `Dock`, яке може набувати таких значень:

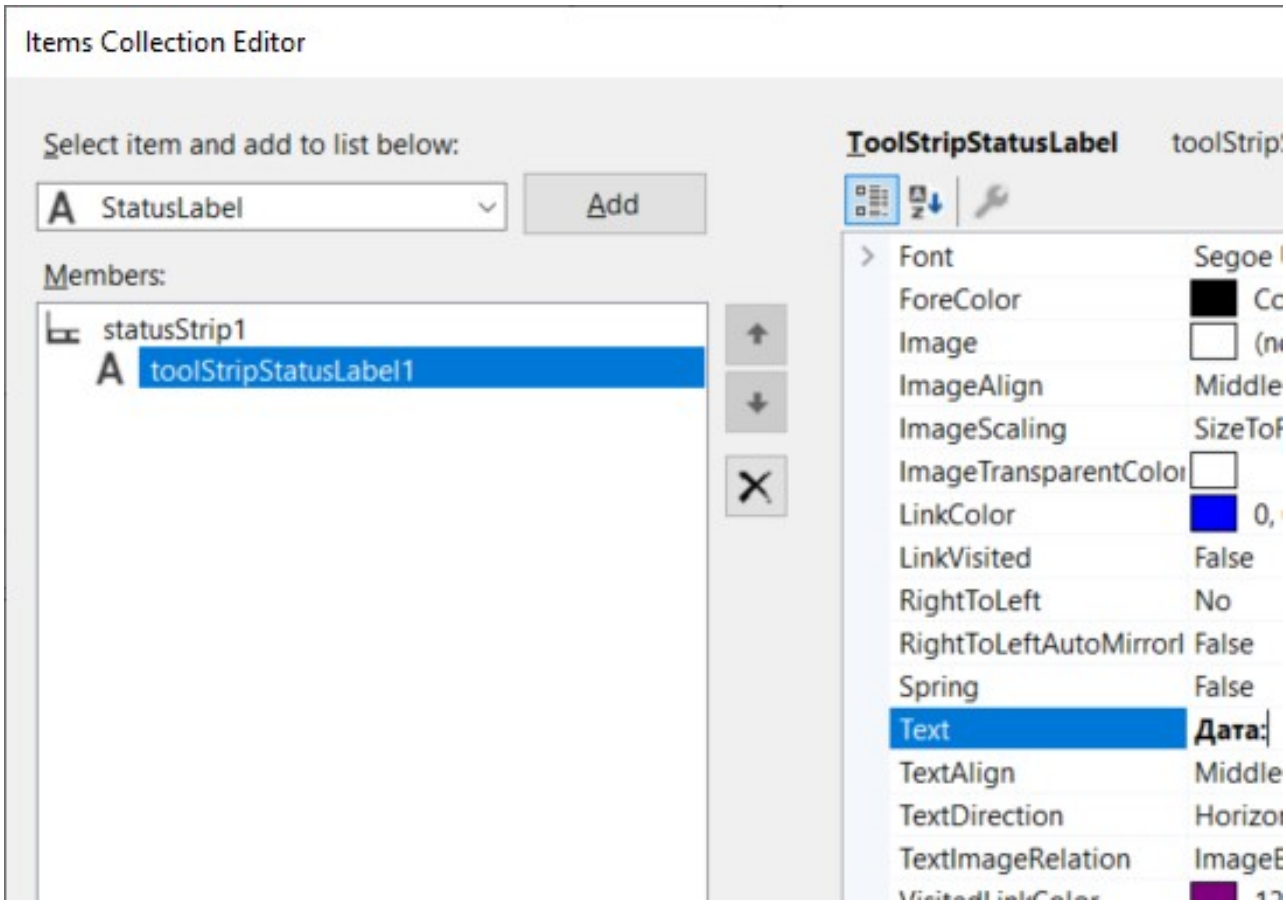
- `Bottom`: розміщення внизу (значення за замовчуванням);
- `Top`: прикріплює статусний рядок до верхньої частини форми;
- `Fill`: розтягує на всю форму;
- `Left`: розміщення в лівій частині форми;
- `Right`: розміщення в правій частині форми;
- `None`: довільне положення.

`StatusStrip` може містити різні елементи. У режимі дизайнера можна додати такі типи елементів:

- `StatusLabel`: мітка для виведення текстової інформації. Представляє об'єкт `ToolStripLabel`;
- `ProgressBar`: індикатор прогресу. Представляє об'єкт `ToolStripProgressBar`;
- `DropDownButton`: кнопка зі списком, що випадає при натисканні. Представляє об'єкт `ToolStripDropDownButton`;
- `SplitButton`: ще одна кнопка, багато в чому аналогічна `DropDownButton`. Представляє об'єкт `ToolStripSplitButton`.



На панелі властивостей також можна звернутись до властивості `Items` компонента `StatusStrip` і у вікні цієї властивості додати і налаштувати всі елементи:

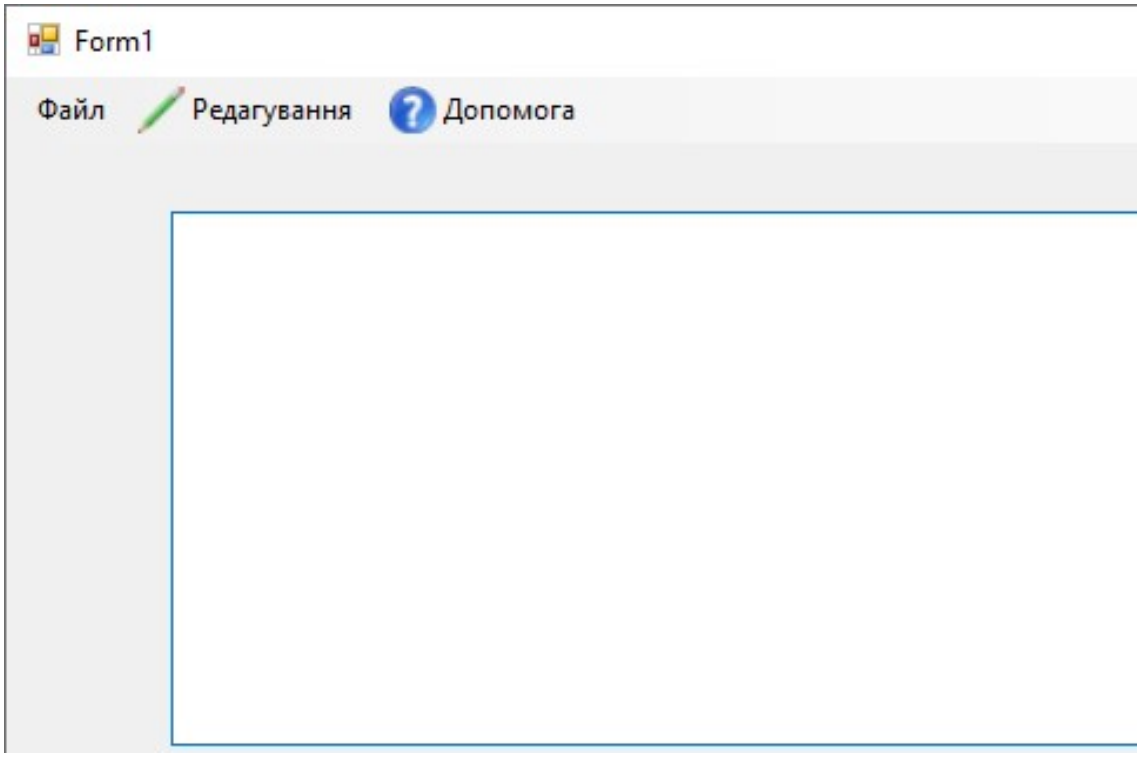


Також можна додати елементи програмно. Створимо невелику програму. Запишемо наступний код форми:

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

```
    ToolStripLabel dateLabel;
    ToolStripLabel timeLabel;
    ToolStripLabel infoLabel;
    Timer timer;
    public Form1()
    {
        InitializeComponent();

        infoLabel = new ToolStripLabel();
        infoLabel.Text = "Поточна дата і година:";
        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();
    }
}
```

Тут створюються три мітки на рядку стану і таймер. Після створення форми таймер запускається, і спрацьовує його подія Tick, в обробнику якого встановлюємо текст міток.



1.25. Опрацювання натиснення клавіш

Для того, щоб компонент опрацьовував натиснення певних клавіш клавіатури на події `PreviewKeyDown` певного компонента потрібно встановити властивість `IsInputKey` у значення `true`, при цьому доступ до клавіш відбувається через об'єкт `Keys`:

```
switch (e.KeyCode) {
                case Keys.Right:
                case Keys.Left:
                case Keys.Down:
                case Keys.Up:
                    e.IsInputKey = true;
                    break;
}
```

У даному прикладі дозволяється опрацьовувати лише клавіші управління курсором вправо, вліво, вниз та вверх.

Доступ до кнопок миші на події `MouseDown` певного компонента відбувається через об'єкт `MouseButtons`:

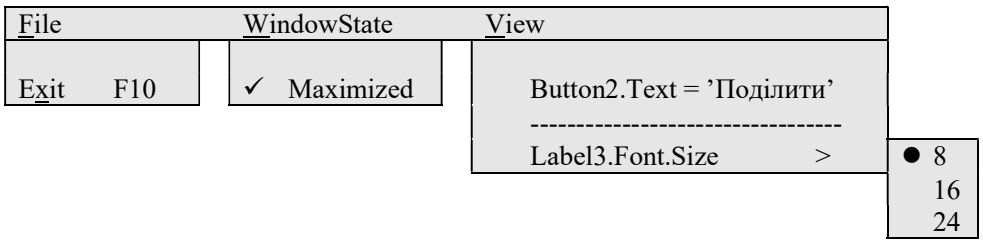
```
switch (e.Button) {
        case MouseButtons.Left:
                опрацювання натиску лівої кнопки миші;
                break;
        case MouseButtons.Right:
                опрацювання натиску правої кнопки миші;
                break;
}
```

У даному прикладі є можливість опрацьовувати лише кнопки миші вліво та вправо.

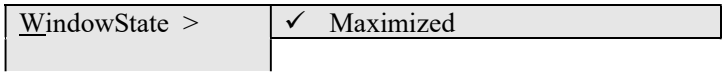
Хід роботи

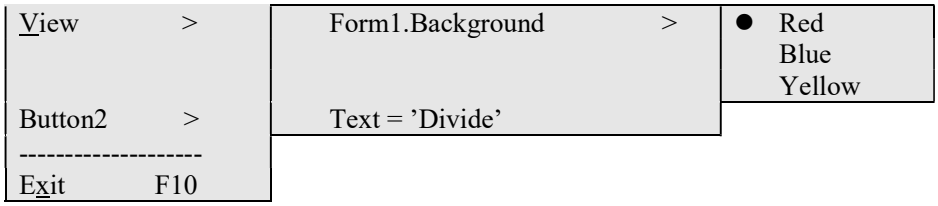
1. Як реакцію на подію *FormClosing* форми, вивести за допомогою функції *MessageBox* повідомлення типу “вивід інформації” з текстом “Зараз вікно буде закрите; чи дійсно потрібно це робити?”. Вказане вікно повинно містити 2 кнопки: натискання кнопки “*Yes*” призводить до закривання вікна, “*No*” – вікно не закривається. *Підказка*: для закривання/не закривання вікна використати один з параметрів події *FormClosing*.
2. Розмістити на формі компонент *TextBox1*, *TextBox2*, *TextBox3* та *Button1*. При натисканні на *Button1* виконати **множення двох дійсних чисел**, введених у компоненти *TextBox1* та *TextBox2*. Забезпечити можливість введення лише цифр. Результат множення виводити одночасно у компонент *Label1* та в компонент *TextBox3*.
3. Розмістити на формі компоненти *ListBox*, *ComboBox* та *Label2*. Компоненти *ListBox* та *ComboBox* повинні мати однакові значення. При по виборі літерного рядка в компоненті *ListBox* виводити номер вибраного елемента списку в компоненті *Label2*, *TextBox* та в заголовку форми. При цьому компонент ComboBox повинен відображати літерний рядок, виділений в компоненті *ListBox*. Аналогічно при виборі певного значення в *ComboBox* компонент *ListBox* також повинен його виділити. Тобто реакція на зміну значення в *ListBox* та *ComboBox* має бути синхронізована.
4. Розмістити на формі компоненти *MenuStrip*, *ContextMenuStrip*, *Button2* та *Label3*. Організувати меню вказаного виду та конфігурації (всі пункти повинні виконувати відповідні дії). Однакові дії мають бути синхронізовані і в *MenuStrip* і в *ContextMenuStrip*. *Примітка*: зверніть увагу на встановлені за замовчуванням значення, а також на підкреслені літери у пунктах меню. Підкреслення літер має з'являтись при натисканні клавіші **Alt**, а при натисканні комбінації клавіш **Alt+*літера*** має виконуватись відповідна дія.

MenuStrip для форми:

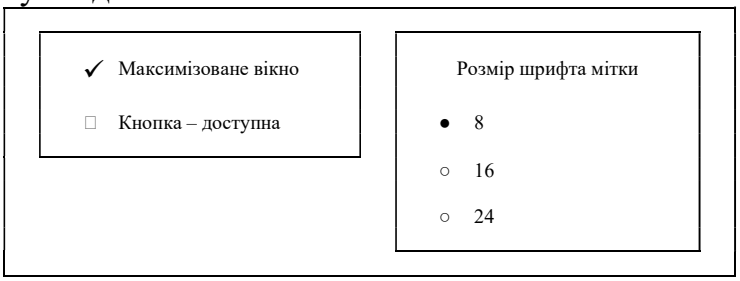


ContextMenuStrip для форми:





5. Розмістити на формі компоненти *OpenFileDialog* та *PictureBox*. Використовуючи компонент *OpenFileDialog*, завантажити графічне зображення і вивести його за допомогою компонента *PictureBox* під час виконання програми. При цьому компонент *OpenFileDialog* повинен "пам'ятати" робочий каталог.
6. Розмістити на формі компоненти *SaveFileDialog, richTextBox, ColorDialog* та *FontDialog*. За допомогою компоненти *OpenFileDialog* реалізувати відкриття файлу і завантаження його вмісту в компонент *richTextBox*, а за допомогою компоненти *SaveFileDialog* – збереження вмісту компоненти *richTextBox* у файл. У вікні *SaveFileDialog* реалізувати можливість вибору типів файлів, за замовчуванням використовувати тип *.txt*. Використовуючи компоненти *ColorDialog* та *FontDialog*, забезпечити можливість встановлювати колір та властивості шрифту для компонента *richTextBox*.
7. Розмістити на формі компоненти *CheckBox1*, *CheckBox2*, *RadioButton1*, *RadioButton2* і *RadioButton3*. Згрупувати ці компоненти згідно рисунку. Залежно від стану перемикачів виконувати відповідні дії над компонентами *Button2* та *Label3.* Також потрібно синхронізувати дії з відповідними пунктами меню, реалізованими у завданні 4.



8. Створити рухомий об'єкт, який переміщується лише при надходженні сигналів від клавіатури (натискання клавіш управління курсором). Реалізувати відбивання об'єкта від границь форми.
9. Створити рухомий об'єкт, який автоматично переміщується при надходженні сигналів від таймера. При надходженні сигналів від клавіатури (натискання клавіш управління курсором) змінює напрям руху чи зупиняється (натискання клавіші "пробіл"). Рух та зупинку об'єкта реалізувати за допомогою компонента *CheckBox3* (старт/стоп), а напрям руху вказувати за допомогою 4-х компонетів *RadioButton* (рух вверх, вниз, вліво, вправо). Натискання клавіш управління курсором та компоненти *RadioButton* повинні бути синхронізовані, тобто при натисненні клавіші "вліво" стає активним відповідний *RadioButton* (рух вліво), при цьому об'єкт повинен одразу реагувати на зміну напряму руху за допомогою компонентів *RadioButton*. Для інших клавіш управління курсором аналогічно налаштувати відповідні дії.

10. Створити рухомий об’єкт, який переміщується в результаті надходження сигналів від мишки (при перетягуванні натиснутою лівою кнопкою мишки). Користувач натискає лівою кнопкою миші на об’єкті, і не відпускаючи її, перетягує його у інше місце на формі і відпускає ліву кнопку миші. Після цього об’єкт повинен зберегти своє нове місце розташування на формі.