МІНІСТЕРСТВО ОСВІТІ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

О.С. Зеленський

В.С. Лисенко

РОЗРОБКА WINDOWS-ДОДАТКІВ НА МОВІ C#

Частина 1

Навчальний посібник

Кривий Ріг

2023

В навчальному посібнику (перша частина) розглядаються можливості Windows Forms – бібліотеки класів Windows платформи .NET. Розглядається створення динамічних користувальних інтерфейсів, реалізація графічного виведення на дисплей та принтер, управління клавіатурою, мишкою і таймером. У посібнику докладно розглянуто ієрархію класів .NET Framework. Закладаються основи розробки власних програм із застосуванням Windows Forms. При цьому кожна частина посібника має бути функціонально незалежною, що дозволяє її використовувати для конкретної дисципліни.

В основу матеріалу покладено класичну, на наш погляд, книгу Петцольда Чарльза. Багато прикладів скориговані з урахуванням сучасної .NET FrameWork 6.0. Наведено авторські розробки.

У другій частині детально вивчатимуться елементи управління. Далі будуть розглянуті робота з базою даних, Web-програмування, використання технологій WPF.

Навчальний посібник адресовано студентам, аспірантам, викладачам. Може бути використаний як самовчитель.

Автори: Зеленський О.С., Лисенко В.С. – Кривий Ріг: Державний університет економіки і технологій, 2023.-160 с.

Рецензенти:

А.І. Купін, д.т.н, професор, завідувач кафедри комп'ютерних систем та мереж, Криворізький національний університет.

І.О. Музика, к.т.н, доцент кафедри комп'ютерних систем та мереж, декан факультету інформаційних технологій Криворізький національний університет.

В.Б. Хоцкіна, к.т.н., доцент кафедри інформатики і прикладного програмного забезпечення, Державний університет економіки і технологій.

Рекомендовано Вченою радою Державного університету економіки і технологій

Протокол № 10 від 30.03.2023 р.

Зміст

Вступ

До цього часу ми вивчали можливості мови C# на прикладі консольних додатків [1]. У цьому посібнику наводяться принципи створення Windows-додатків і дається уявлення про основні класи. Операційна система забезпечує поділ ресурсів: кожному додатку виділяється свій адресний простір, розподіляється процесорний час і організовуються черги для доступу до зовнішніх пристроїв. Всередині програми також можна організувати паралельне виконання кількох фрагментів, що називаються потоками. У різних версіях Windows використовуються різні механізми диспетчеризації. Дотримується незалежність програм від апаратури. Для керування апаратними засобами будь-яка програма звертається до операційної системи. Це забезпечує незалежність від конкретних фізичних характеристик пристроїв: при зміні пристрою жодних змін до програми вносити не потрібно. Керування зовнішніми пристроями забезпечується за допомогою драйверів. Основна взаємодія з користувачем здійснюється у графічному режимі. Кожна програма виконує виведення у відведену йому прямокутну область екрану, яка називається вікном. Вікно складається із стандартних елементів. Це спрощує освоєння програм користувачем і полегшує роботу програміста, оскільки в його розпорядження надаються бібліотеки інтерфейсних компонентів. Інтерфейсні компоненти звертаються до апаратури не безпосередньо, а через функції операційної системи, що називаються API (Application Program Interface – програмний інтерфейс програми). Функції API знаходяться в динамічних бібліотеках (Dynamic Link Library, DLL). Ці бібліотеки називаються динамічними тому, що функції, що знаходяться в них, не підключаються до кожного виконуваного файлу до виконання програми, а викликаються в момент звернення до них. Програма може мати кілька вікон, одне з них є основним. Після закриття головного вікна програма завершується. Кожному додатку доступно простір адрес оперативної пам'яті розміром до 4 Гбайт (для 32-бітної Windows). Надається можливість обміну даними між додатками. Програми можуть обмінюватися даними через буфер обміну або використовуючи інші механізми, наприклад OLE (Object Linking and Embedding – зв’язування та впровадження об’єктів). У 32-розрядних версіях Windows можна виконувати 16-розрядні Windows-програми. В основу Windows покладено принцип керування подіями. Це означає, що і сама система, і програми після запуску очікують на дії користувача і реагують на них заздалегідь заданим чином. Будь-яка дія користувача (натискання клавіші на клавіатурі, клацання кнопкою миші, переміщення миші) називається подією. Середовище Visual Studio.NET містить зручні засоби розробки Windows-додатків, що виконують замість програміста рутинну роботу – створення шаблонів програми і форм, заготовок обробників подій, організацію циклу обробки повідомлень тощо.

Форма, яка використовується у якості головного вікна додатку, зазвичай складається із заголовку з ім’ям додатку, меню, розташованого під заголовком, та внутрішньої області (клієнтської області). Як видно з рис.1, клас *Form* спадкується від *ContainerControl*, але насправді в нього довгий родовід, який

починається з класу *Object*, від якого спадкуються усі класи .NET Framework. Об’єкти інтерфейсу користувача (кнопки, смуги прокручування, поля редагування і т.п.) позначаються терміном елемент управління (*Control*). Основна частина підтримки цих об’єктів, зокрема, введення з клавіатури та мишки і візуальне відображення, реалізована у класі *Control*. У класі *ScrollableControl* додається підтримка автоматичного прокручування. Клас *ContainerControl* дозволяє елементам управління, подібно до діалогових вікон, виступати в якості батьків (parent) інших елементів управління.

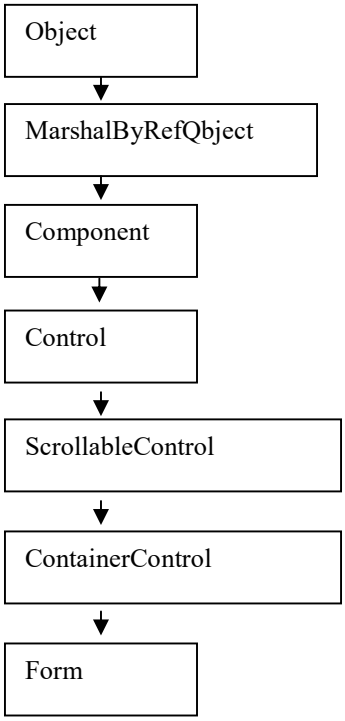


Рис. 1. Родословна класу Form

1. Метод MessageBox.Show у Windows Forms

Часто у *Windows Forms* при выдлагодженны програм або виведенні тексту використовується діалогове вікно, яке створюється методом *MessageBox.Show***.** Метод є статичним, тому необхідно просто ввести "*MessageBox*", натиснути точку та вибрати "*Show*". У лістингі 1 показаний метод *MessageBox.Show*, який використовується в обробнику подій *Form1_DoubleClick.*

Лістинг 1. (examp01)

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

namespace exam1
```

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

        public void mes()
        {
            DialogResult
            dr = MessageBox.Show("Перший Windows-додаток",
               "Заголовок",MessageBoxButtons.YesNoCancel,
               MessageBoxIcon.Question);
            if (dr == DialogResult.Yes)
            MessageBox.Show("Yes");
            else if (dr == DialogResult.No)
            MessageBox.Show("No");
            else
            MessageBox.Show("Cancel");
        }

      private void Form1_DoubleClick(object sender, EventArgs e)
        {
           mes();
        }
    }
}
```

Вище представлені декілька викликів методу *MessageBox.Show* з використанням різних перевантажень.

Один з прикладів перевантаженої функції:

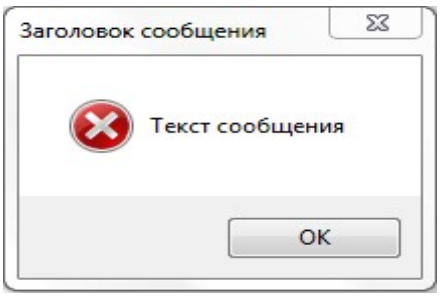
```
System.Windows.Forms.DialogResult MessageBox.Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIcon.icon);
```

Значення `MessageBoxButtons`:

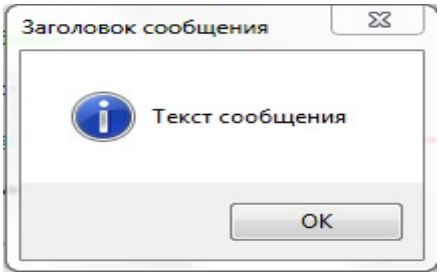
1. *MessageBoxButtons.AbortRetryIgnore* – Перервати | Повтор | Пропустити
2. *MessageBoxButtons.OK* - ОК
3. *MessageBoxButtons.OKCancel* – ОК | Скасування
4. *MessageBoxButtons.RetryCancel* – Повтор | Скасування
5. *MessageBoxButtons.YesNo* – Так | Ні
6. *MessageBoxButtons.YesNoCancel* – Так | Ні | Скасування

Параметр *MessageBoxIcon* встановлює тип повідомлення, та може приймати наступні значення:

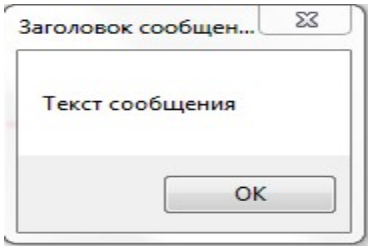
1. *MessageBoxIcon.Error*



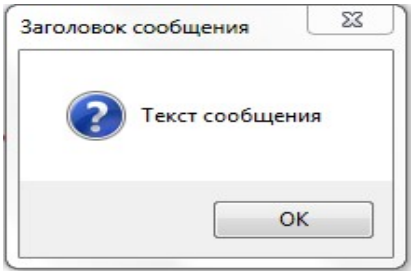
2. *MessageBoxIcon.Information*



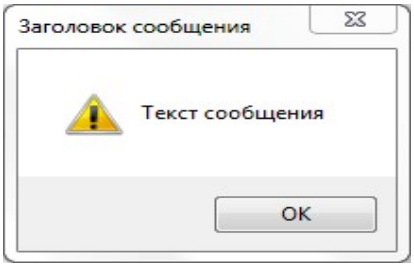
3. *MessageBoxIcon.None*



4. *MessageBoxIcon.Question*



5. *MessageBoxIcon.Warning*



Якщо в повідомленні приймають участь декілька кнопок, то нам треба записати результат натиснутої кнопки до змінної, робиться це наступним чином:

```
DialogResult result = MessageBox.Show("Будете продовжувати вво-
дити данні?”, "Увага!", MessageBoxButtons.YesNo,
MessageBoxIcon.Question);
```

DialogResult не є дійсним класом, але є іменованою константою з перелічень. Це означає, що неможна створити новий *DialogResult* з новим оператором. Щоб перевірити результат *DialogResult*, спочатку призначте змінній результат *MessageBox.Show*. Потім введіть "=", і Visual Studio запропонує варіанти з перелічення *DialogResult*. Можна порівнювати *DialogResult* так же, як можна порівнювати цілочисельні типи, наприклад *int*.

```
if (result == DialogResult.Yes) //Натиснута Так
```

`DialogResult` може приймати наступні значення:

1. *DialogResult.Abort* – Перервати
2. *DialogResult.Cancel* – Скасування
3. *DialogResult.Ignore* – Пропустити
4. *DialogResult.No* – Ні
5. *DialogResult.Yes* – Так
6. *DialogResult.OK* – ОК
7. *DialogResult.Retry* – Повтор

2. Виведення тексту

Клас *Graphics* містить багато методів для малювання графічних фігур, наприклад, ліній, кривих, прямокутників, еліпсів та растрових зображень. Крім того, цей клас містить достатньо універсальний метод для виведення тексту *DrawString*. Цей метод має декілька перевантажених версій, але перші три параметри у них завжди однакові. Самі прості версії *DrawString* мають вигляд:

void DrawString (string str, Font font, Brush brush, float x, float y)
void DrawString (string str, Font font, Brush brush, PointF point)

Перший параметр – рядок, який виводиться. Другий та третій – відповідно шрифт та колір тексту. Останні параметри – координати виведення рядка, які визначають верхній лівий кут тексту, що створюється. Достатньо сказати, що для програм *Windows Forms* доступна велика кількість шрифтів, розміри яких можна масштабувати. Можна використовувати шрифт за умовчанням. Дуже зручно, що кожний клас, успадкований від *Control*, спадкує властивість *Font*, в якій зберігається шрифт елементу управління за умовчанням.

2.1. Об'єкт Font

Окремо об'єкт *Font* можна отримати різноманітними конструкторами класу *Font*. Нижче вони будуть розглянуті більш детально, а поки розглянемо найбільш часто використовуваних два з них:

```
Font(string familyName, float fSizelnPoints)
Font(string familyName, float fSizelnPoints, FontStyle fs)
```

Назва шрифту FontFamily можна отримати з наявного шрифту. Наприклад, якщо треба отримати шрифт на основі існуючого, але з іншими розмірами, можна послатися на властивість Name наявного шрифту

```
Font font = new Font(font.Name, 18, font.Style);
```

У конструкторі можна явно вказати назву шрифту: "Times New Roman", "Arial", "Courier New" і т.д. Ось як можна створити 24-пунктний шрифт Times New Roman:

```
Font font = new Font("Times New Roman", 24);
```

Можна перший параметр отримати із статичних властивостей методу FontFamily:

```
Font font = new Font(FontFamily.GenericSerif, 24)
```

Таблиця 1.

Статичні властивості FontFamily (вибірково)

Властивість	Опис
GenericSerif	Наприклад, Times New Roman
GenericSansSerif	Наприклад, Arial
GenericMohospace	Наприклад, Courier New

FontStyle у конструкторі *Font* – це перелічення, що складається із серії однобітних прапорів:

Таблиця 2.

Перелічення FontStyle

Член	Значення
Regular	0
Bold	1
Italic	2
Underline	4
Strikeout	8

2.2. Об'єкти Brush і Color

Найпростіше домогтися різноманіття кольорів – використати клас *Brushes* (*using System.Drawing*). Клас *Brushes* містить більше ніж 100 статичних властивостей з різними назвами кольорів. Приклад його використання:

```
DrawString("Привет", Font, Brushes.Blue, 100f, 100f)
```

Властивості класу *Brushes* – це об'єкти типу *Brush*. В основному використовується *Brushes.Black*, тобто чорний колір.

Крім виведення тексту об'єкт *Brush* використовується для заливання замкнутих областей. Brush є абстрактним (*abstract*) класом, тобто створити його екземпляр неможна. Клас *Brush* – батько п'яти інших класів: *SolidBrush*, *HatchBrush*, *TextureBrush*, *LinearGradientBrush* та *PatbGradientBrush*. Далі будуть розглянуті різні пензлі. У нашому випадку в основному використовується суцільний колір та об'єкт *Brush* можна створити так:

```
Brush brush = new SolidBrush(Color),
```

де `Color` – об'єкт;

Так як для визначення кольору треба дуже мало даних, колір претендує на то, щоб бути структурою, а не класом, і справді *Color* –ще одна важлива структура простору імен *SystemDrawing*. Кольори у *Windows Forms* основані на моделі ARGB (alpha-red-green-blue). Кольори визначаються однобайтовими значеннями червоного, зеленого та синього. Альфа-канал визначає прозорість кольору. Структура *Color* має один конструктор за умовчанням, який можна використовувати так:

```
Color color = new Color();
```

Екземпляр класу створювати недоцільно. Замість цього об'єкти типу *Color* створюються статичними методами або властивостями. Статичні властивості в *Color* забезпечують повертання 141 назви для різних кольорів.

Приклад: *Color.Red* – буде повертати об'єкт *Color*, який характеризує червоний колір.

У ряді випадків для отримання об'єктів *Color* доцільно використання сттичного методу

```
Color.FromArgb(int r, int g, int b),
```

де *r*, *g*, *b* – відповідно червоний, зелений та синій кольори, значення яких знаходяться у діапазоні 0 – 255.

Color.FromArgb() – одне з чотирьох перевантажень статичної функції.

Властивості класу *Brushes* – це об'єкти типу *Brush*. В основному використовується *Brushes.Black*, тобто чорний колір.

Структури *Color* можна отримати властивостями класу *SystemColors*, які наведені у таблиці 3.

Приклад: *SystemColors.Menu* – буде повертати структуру *Color*, яка є кольором фону меню.

Таблиця 3.

Властивості класу SystemColors

Назва	Опис
ActiveBorder	Отримує структуру *Color*, яка є кольором границі активного вікна.
ActiveCaption	Повертає структуру *Color*, яка є кольором фону рядка заголовка активного вікна.
ActiveCaptionText	Отримує структуру *Color*, яка є кольором тексту заголовка активного вікна.
AppWorkspace	Отримує структуру *Color*, яка є кольором робочої області додатку.
ButtonFace	Повертає структуру *Color*, яка є кольором лицьового боку тривимірного елементу.
ButtonHighlight	Повертає структуру *Color*, яка є кольором виділенні тривимірного елементу.
ButtonShadow	Повертає структуру *Color*, яка є кольором тіні тривимірного елементу.
Control	Повертає структуру *Color*, яка є кольором лицьового боку тривимірного елементу.
ControlDark	Повертає структуру *Color*, яка є кольором тіні тривимірного елементу.
ControlDarkDark	Повертає структуру *Color*, Яка є темним кольором тіні тривимірного елементу.
ControlLight	Повертає структуру *Color*, яка є світлим кольором тривимірного елементу.
ControlLightLight	Повертає структуру *Color*, яка є кольором виділення тривимірного елементу.
ControlText	Повертає структуру *Color*, яка є кольором тексту у тривимірному елементі.
Desktop	Отримує структуру *Color*, яка є кольором робочого столу.
GradientActiveCaption	Повертає структуру *Color*, яка є самим світлим кольором градієнту кольору у рядку заголовку активного вікна.
GradientInactiveCaption	Повертає структуру *Color*, яка є самим світлим кольором градієнту кольору у рядку заголовку неактивного вікна.
GrayText	Повертає структуру *Color*, яка є кольором сірого тексту.
Highlight	Повертає структуру *Color*, яка є кольором фону вибраних елементів.
HighlightText	Отримує структуру *Color*, яка є кольором тексту вибраних елементів.
HotTrack	Повертає структуру *Color*, яка є кольором, що використовується для позначення відстеженого елементу.
InactiveBorder	Отримує структуру *Color*, яка є кольором границі неактивного вікна.
InactiveCaption	Повертає структуру *Color*, Яка є кольором фону рядка заголовку неактивного вікна.
InactiveCaptionText	Повертає структуру *Color*, яка є кольором тексту у рядку заголовку неактивного вікна.
Info	Повертає структуру *Color*, яка є кольором фону спливаючої підказки.

Продовження таблиці 3

Назва	Опис
InfoText	Повертає структуру *Color*, яка є кольором тексту спливаючої підказки.
Menu	Отримує структуру *Color*, яка є кольором фону меню.
MenuBar	Повертає структуру *Color*, яка є кольором фону рядка меню.
MenuHighlight	Повертає структуру *Color*, яка є кольором, що використовується для виділення пунктів меню, коли меню відображається як плоске меню.
MenuText	Отримує структуру *Color*, яка є кольором тексту меню.
ScrollBar	Повертає структуру *Color*, яка є кольором фону смуги прокручування.
Window	Повертає структуру *Color*, яка є кольором фону клієнтської області вікна.
WindowFrame	Отримує структуру *Color*, яка є кольором рамки вікна.
WindowText	Отримує структуру *Color*, яка є кольором тексту у клієнтській області вікна.

Підвищення продуктивності досягається за допомогою властивостей класів *SystemPens* або *SystemBrushes*, а не шляхом створення нового пера або пензля на основі значення із об'єкту *SystemColors*. Наприклад, якщо треба отримати пензель для кольору лицьового боку тривимірного елементу, використовуйте властивість *SystemBrushes.Control*, так як вона повертає вже існуючий пензель, у той час як виклик конструктору *SolidBrush* із значенням параметру *SystemColors.Control* створить новий пензель.

2.3. Приклади використання методу DrawString класса Graphics (grfx):

```
1. grfx.DrawString("Hello", Font, new SolidBrush(ForeColor), ClientSize.Width /2, 0);

2. grfx.DrawString("Hello", Font, new SolidBrush(Color.FromArgb(255, 0, 0)) , 50.0F, 50.0F);

3. grfx.DrawString("Hello", Font, Brushes.Blue , 10.0F,  10.0F);

4. grfx.DrawString("Hello", Font, new SolidBrush(Color.Blue) , 0,0);
grfx.DrawString("Hello", new Font(Font.Name, 18, Bold), Brushes.Red , 0, 0);

5. grfx.DrawString("Hello", new Font("Times New Roman", 14), new SolidBrush(Color.Blue) ,
0, 0);
6. grfx.DrawString("Hello", new Font(FontFamily.GenericSerif, 24) Brushes.Red, 0, 0);

6. String drawString = "Sample Text";
   Font drawFont = new Font("Arial", 16);
   SolidBrush drawBrush = new SolidBrush(Color.Black);
   PointF drawPoint = new PointF(150.0F, 150.0F);
   grfx.DrawString(drawString, drawFont, drawBrush, drawPoint);
```

Як зазначалося, координати точки, переданої *DrawString*, вказують позицію верхнього лівого кута. Це не завжди зручно. Скажімо, необхідно вивести текст строго посередині висоти клієнтського вікна. Якщо вказати координату *Y*, що дорівнює середині висоти вікна, то текст все одно буде нижче середини вікна приблизно на половину висоти шрифту. Виникає необхідність керувати розташуванням тексту щодо заданих координат. Саме це розгляда-

ється в перевантаженій версії *DrawString*, що містить ще один параметр поряд з текстовим рядком, шрифтом, пензлем і початковою позицією. Додатковий параметр – об'єкт *StringFormat*, призначений для точного вказівки того, як відображати текст.

void DrawString (string str, Font font, Brush brush, float x, float y, StringFormat strfm)
або
void DrawString (string str, Font font, Brush brush, PointF point, StringFormat strfm)

Рассмотрим только наиболее часто используемую его возможность выравнивания текста. Для этого создается объект *StringFormat*:

```
StringFormat strfm = new StringFormat();
```

Потім, щоб вказати розташування тексту, надаються значення двом властивостям цього об’єкту *Alignment* (вирівнювання по горизонталі) і *LineAlignment* (вирівнювання по вертикалі).

Обидві ці властивості мають тип перелічення StringAlignment (табл. 4)

Таблиця 4

Перелічення *StringAlignment*

Член	Значення	Опис
Near	0	лівий або верхній
Center	1	центр
Far	2	правий або нижній

Приклад виведення тексту з використання методу *DrawString* та об’єкту *StringFormat* показано на лістингу 2 та рис. 2.

Лістинг 2. (exam02)

```
using System;
using System

namespace exam2
{
    public partial class Form1 : Form
    {
        Brush br;
        Pen pen;
        Font  font;
        StringFormat strfm = new StringFormat();
        Point sr;



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

```
            br = new SolidBrush(Color.Blue);
            pen = new Pen(Color.Red);
            font = new Font("Times New Roman",
                            20.25f,FontStyle.Bold);
            sr = new Point(ClientSize.Width / 2,
            ClientSize.Height / 2);
        }

 private void Form1_Paint(object sender, PaintEventArgs e)
{
   Graphics gr = e.Graphics;
   strfm.Alignment = StringAlignment.Center;
   strfm.LineAlignment = StringAlignment.Near;
   gr.DrawString("Привет1 программистам", font, br,sr, strfm);

   strfm.LineAlignment = StringAlignment.Center;
   gr.DrawString("Привет2 программистам", font, br, sr, strfm);

   strfm.LineAlignment = StringAlignment.Far;
   gr.DrawString("Привет3 программистам", font, br, sr, strfm);

   gr.DrawLine(pen, 0, ClientSize.Height / 2, ClientSize.Width,
               ClientSize.Height / 2);
   gr.DrawLine(pen, ClientSize.Width / 2, 0,
                ClientSize.Width / 2, ClientSize.Height);
}
```

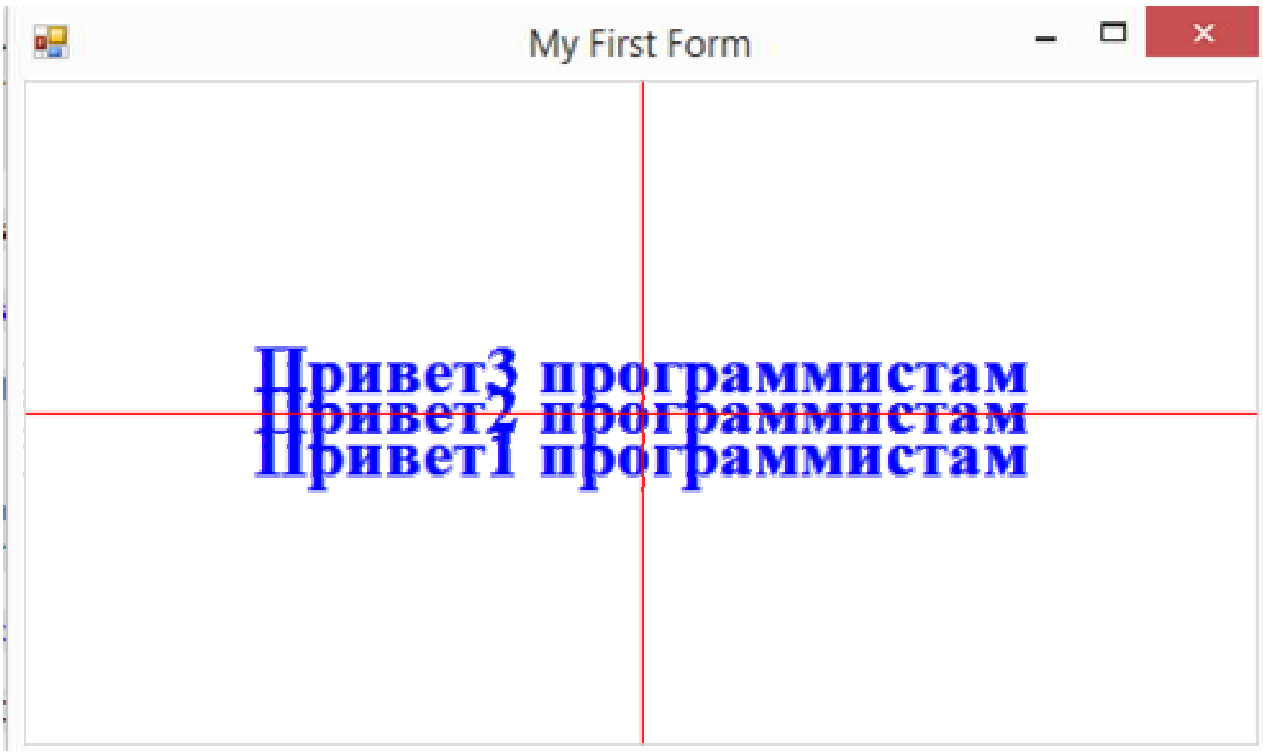


Рис. 2. До виведення тексту в *Windows Form*

Ми розглянули чотири версії методу *Drawstring*:

void DrawString (string str, Font font, Brush brush, float x, float y)
void DrawString (string str, Font font, Brush brush, PointF point)
void DrawString (string str, Font font, Brush brush, float x, float y, StringFormat strfm)
void DrawString (string str, Font font, Brush brush, PointF point, StringFormat strfm)

Із шести основних версій методу *Drawstring* розглянемо решту дві.

void DrawString (string str, Font font, Brush brush, RectangleF rectf, StringFormat strfm)
void DrawString (string str, Font font, Brush brush, RectangleF rectf, StringFormat strfm)

Дві нові версії з *RectangleF* дещо відрізняються від раніше розглянутих. Метод *DrawString* обмежує текст прямокутником, і додатковий параметр *StringFormat* визначає, як текст розташовується всередині прямокутника. Наприклад, якщо *ClientRectangle* передається функції *DrawString* та властивості *Alignment* і *LineAlignment StringFormat* мають значення *StringAlignment.Center*.

Приклад виведення тексту з використанням методу *DrawString* та об'єкту *RectangleF* показані на лістингі 3 та рис. 3.

Лістинг 3. (exam03)

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

namespace exam3
{
    public partial class Form1 : Form
    {
        Brush br;
        Pen   pen;
        Font  font;
        StringFormat strfm = new StringFormat();
        Point sr;
        public Form1()
        {
            InitializeComponent();
            br = new SolidBrush(Color.Blue);
            pen = new Pen(Color.Red);
            font = new Font("Times New Roman",
                                        20.25f,FontStyle.Bold);
        }
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
    Graphics gr = e.Graphics;
    strfm.Alignment = StringAlignment.Center;
    strfm.LineAlignment = StringAlignment.Near;
    gr.DrawString("Привет1 программистам", font, br,
                  ClientRectangle, strfm);
    strfm.LineAlignment = StringAlignment.Center;
    gr.DrawString("Привет2 программистам", font, br,
                  ClientRectangle, strfm);
     strfm.LineAlignment = StringAlignment.Far;
     gr.DrawString("Привет3 программистам", font, br,
                  ClientRectangle, strfm);

    gr.DrawLine(pen, 0, ClientSize.Height / 2, ClientSize.Width,
                ClientSize.Height / 2);
    gr.DrawLine(pen, ClientSize.Width / 2, 0,
                ClientSize.Width / 2, ClientSize.Height);
}
    private void Form1_Resize(object sender, EventArgs e)
     {
       Invalidate();
     }
    }
}
```

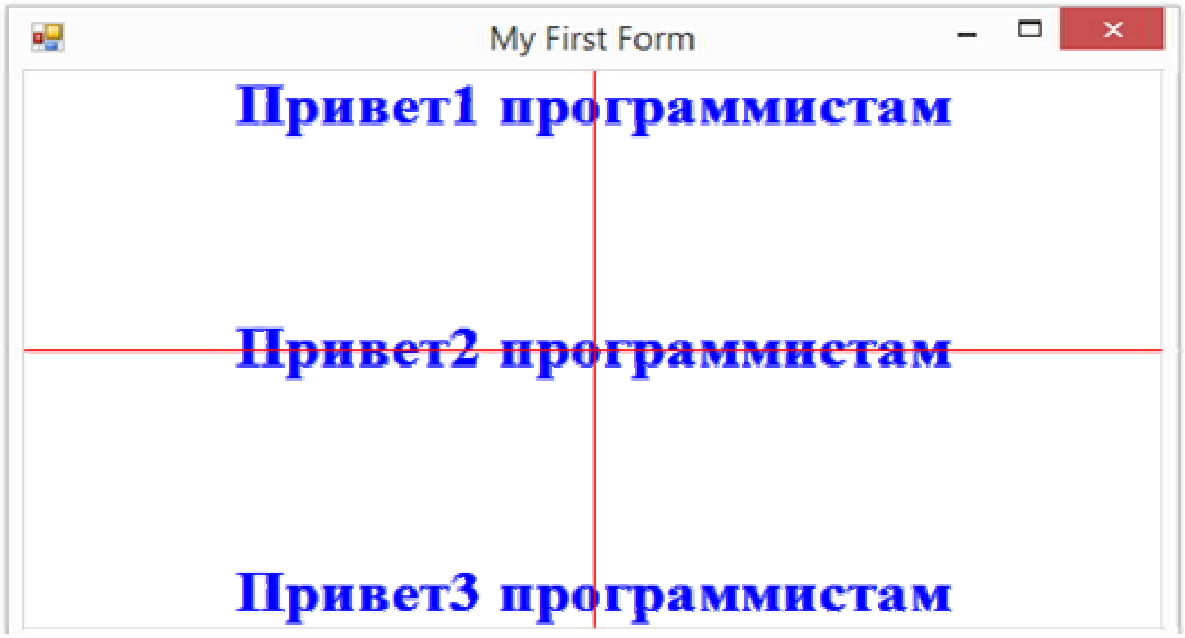


Рис. 3. До виведення тексту в *Windows Form*

2.4. Форма та її клієнтська область

Форма має різні властивості (все вони успадковані класом *Form* від *Control*), які показують розмір форми та її розташування на екрані. Вони наведені в табл. 5.

Таблиця 5.

Властивості Control

Тип	Властивість	Доступ	Коментарі
Point	*Location*	Читання/Запис	Відносно екрану
Size	*Size*	Читання/Запис	Розмір форми
Rectangle	*Bounds*	Читання/Запис	Дорівнює *Rectangle(Location, Size)*
int	*Width*	Читання/Запис	Дорівнює *Size.Width*
int	*Height*	Читання/Запис	Дорівнює *Size.Height*
int	*Left*	Читання/Запис	*Location.X*
int	*Top*	Читання/Запис	*Location.Y*
int	*Right*	Читання	*Location.X + Size.Width*
int	*Bottom*	Читання	*Location.Y + Size.Height*
Point	*DesktopLocation*	Читання/Запис	
Rectangle	*DesktopBounds*	Читання/Запис	

Список властивостей, з якими зазвичай доводиться мати справу, можна скоротити до чотирьох чисел: координати *x* та *y* лівого верхнього кута форми відносно лівого верхнього кута монітору, висота та ширина форми. Останні 2 властивості (табл. 5) схожі на *Location* та *Bounds*, але враховують панель задач.

При програмуванні дуже важливим є розташування та розміри клієнтського вікна. Клієнтська область – внутрішня область форми, де відбувається малювання, виведення тексту, розташування елементів управління. У клієнтську область не входять заголовок, рамка форми, меню, смуги прокрутки.

У класі *Form* тільки дві властивості (спочатку реалізовані в *Control*), які відносяться до розміру клієнтської області: *ClientSize* та *ClientRectangle*.

Властивість *ClientSize* показує висоту та ширину клієнтської області у пікселях. Властивість *ClientRectangle* не дає додаткової інформації в порівнянні з *ClientSize*, тому що властивості *X* і *Y ClientRectangle* завжди дорівнюють 0.

Приклад використання, відповідальний за розмір та місцезнаходження клієнтської області, показано на лістингу 4 та рис. 4.

Лістинг 4. (exam04)

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

namespace exam4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
        }

           protected override void OnPaint(PaintEventArgs pea)
           {
                  string str = "location: " + Location + "\n" +
                  "Size: " + Size + "\n" +
                  "Bounds: " + Bounds + "\n" +
                  "Width: " + Width + "\n" +
                  "Height: " + Height + "\n" +
                  "Left: " + Left + "\n" +
                  "Top: " + Top + "\n" +
                  "Right: " + Right + "\n" +
                  "Bottom: " + Bottom + "\n" +
                  "DesktopLocation: " + DesktopLocation + "\n" +
                  "DesktopBounds: " + DesktopBounds + "\n" +
                  "ClientSize: " + ClientSize + "\n" +
                  "ClientRectangle: " + ClientRectangle + "\n";
             pea.Graphics.DrawString(str, Font, Brushes.Red, 0, 0);
              }
        private void Form1_MouseDoubleClick(object sender,
                                            MouseEventArgs e)
                {
                       Invalidate();
                }

        private void Form1_LocationChanged(object sender,
                                                  EventArgs e)
        {
            Invalidate();
        }
     }
    }
```

У програмі використовуються відгуки від подвійного клацання миші та переміщення форми.

Рис. 4. Властивості клієнтського вікна

3. Клас System.Object та його методи

На початку всього ланцюга спадкування класів знаходиться клас *System.Object*. Всі решта класів, навіть ті, які ми додаємо у свій проект, а також базові типи, такі як *System.Int32*, є неявно похідними від класу *Object*. Навіть якщо ми не вказуємо клас *Object* у якості базового, за умовчанням неявно клас *Object* все рівно знаходиться на вершині ієрархії спадкування. Тому всі типи і класи можуть реалізувати ті методи, які визначені у класі *System.Object*. Розглянемо ці методи.

Метод **ToString()** виводить повну назву класу з вказівкою простору імен, в якому визначений цей клас. І ми можемо перевизначити даний метод. Розглянемо на прикладі:

```
class A
{
   public string Name { get; set; }
   public override string ToString()
   {
      return Name;
   }
}
```

Для класу *A* метод *ToString()* перевизначений за допомогою ключового слова *override* та виводить значення властивості *Name*. У даному випадку ми могли задіяти обидві реалізації. Наприклад, що буде, якщо не буде встановлено ім'я? Тоді буде виведено порожній рядок. Однак ми можемо змінити реалізацію методу *ToString*, щоб у подібному випадку вона виводила ім'я класу:

```
class A
{
   public string Name { get; set; }
   public override string ToString()
   {
      if (Name == "" || Name == null)
```

```
        {
            return base.ToString();
        }
        else
        {
            return Name;
        }
    }
}
```

Метод **GetHashCode()** дозволяє задати деяке числове значення, яке буде відповідати даному об'єкту або його хеш-код. По даному числу, наприклад, можна порівнювати об'єкти.

Метод **GetType()** дозволяє отримати тип даного об'єкту. Наприклад, у нас є ієрархія класів, де класи *Employee* та *Client* спадкуються від загального класу *Person*. В нас є декілька об'єктів класу *Person*, і ми хочемо звернутися тільки до всіх клієнтів. В цьому випадку ми можемо перевірити їх тип:

```
Person[] persons = new Person[]{new Client("Tom", "Johnes", "SberBank", 200, 20),
new Employee("Bill", "Gates", "Microsoft")};

foreach (Person p in persons)
{
    if (p.GetType() == typeof(Client))
    {
        Console.WriteLine("Это объект класса Client");
    }
}
```

За допомогою ключового слова *typeof* ми отримуємо тип класу та порівнюємо його з типом об'єкту. І якщо цей об'єкт представляє тип *Client*, то виконуємо певні дії.

Метод Equals дозволяє порівнювати два об'єкти на рівність:

```
class Human
{
    public string Name { get; set; }

    public override bool Equals(object obj)
    {
        if (obj.GetType() != this.GetType()) return false;

        Human human2 = (Human)obj;
        return (this.Name == human2.Name);
    }
}
```

Метод *Equals()* приймає у якості параметру об'єкт довільного типу, який ми потім приводимо до поточного, якщо вони є об'єктами одного класу. Потім порівнюємо по назвам. Якщо назви співпадають, повертаємо *true*.

Приклад використання методів класу *object* показано на лістингі 5 та рис. 5. Тут замість відгуку на зміну розміру вікна та виконання *Invalidate()*

для перемальовування вікна у функції *Paint()* можна у конструкторі привласнити *ResizeRedraw = true*.

Лістинг 5. (exam05)

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

namespace exam5
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        public override int GetHashCode()
        {
           return 255;
        }

    private void Form1_Paint(object sender, PaintEventArgs e)
     {
            Text = " П Р И В Е Т   2";
string st = "Объект - " + sender.GetType() + " HashCode - " +
          sender.GetHashCode() + "\n";
          Point pt = new Point(10, 20);
          st = st + "\nHashCode-" + pt.GetHashCode();
          Text = " " + (10^20).ToString();
          Random ra = new Random();
          Graphics gr = e.Graphics;
          gr.Clear(Color.FromArgb(ra.Next(128), ra.Next(128),
          ra.Next(128)));
          gr.DrawString(" П Р И В Е Т   " + st, Font,
          new SolidBrush(Color.FromArgb(ra.Next(128,255),
          ra.Next(128,255),ra.Next(128,255))), 0, 0);
       }
        private void Form1_Resize(object sender, EventArgs e)
        {
            Invalidate();
        }
    }
 }
```

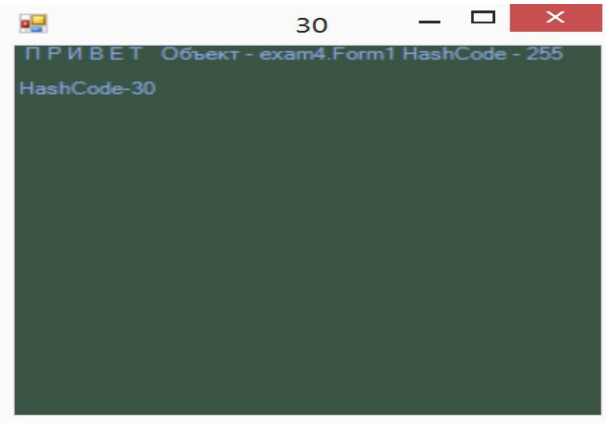


Рис. 5. До використання методів класу *object*

4. Вимірювання рядків, клас SystemInformation

Для центрування тексту по встановленій його довжині ефективно використання методу класу *Graphics MeasureString.* У нього декілька версій, виклик самий простий з них виглядає так:

SizeF sizefText = gr. MeasureString(str, font);

MeasureString повертає структуру *SizeF,* яка показує ширину та висоту рядка у пікселях. Властивість *Height* структури *SizeF* завжди одна й та ж, тобто не залежить від рядка символів. Властивість *Width* структури *SizeF* залежить від символів, що містяться в рядку. Для всіх шрифтів, крім моноширного, ширина символу "*i*" менше, ніж ширина "*W*".

Приклад використання вимірювання рядків показано на лістингу 6 та рис. 6.

Лістинг 6. (exam06)

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

namespace exam6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
        }

     private void Form1_Paint(object sender, PaintEventArgs e)
     {
         Graphics gr = e.Graphics;
         string str = "Hello, world";
         SizeF sizefText = gr.MeasureString(str, Font);
    gr.DrawString(str , Font, new SolidBrush(Color.FromArgb(255,0,0)),
         (ClientSize.Width - sizefText.Width)/2,
```

```
        (ClientSize.Height - sizefText. Height)/2);
        gr.DrawLine(new Pen(Color.FromArgb(0, 0, 0), 3),
        (ClientSize.Width - sizefText.Width) / 2, (ClientSize.Height -
        sizefText.Height) / 2,
        (ClientSize.Width - sizefText.Width) / 2 + (int)sizefText.Width,
        (ClientSize.Height - sizefText.Height) / 2);
        Rectangle rect = new Rectangle(0,0,ClientRectangle.Width,
        ClientRectangle.Height/3);
        rect.Inflate((int)(-ClientRectangle.Width*0.1), -(int)20);
        gr.DrawRectangle(new Pen(Color.FromArgb(255, 0, 0),10), rect);
        StringFormat strf = new StringFormat();
        strf.Alignment = StringAlignment.Center;
        strf.LineAlignment = StringAlignment.Center;
        gr.DrawString(str, Font, Brushes.Blue, rect, strf);
      }
    }
}
```

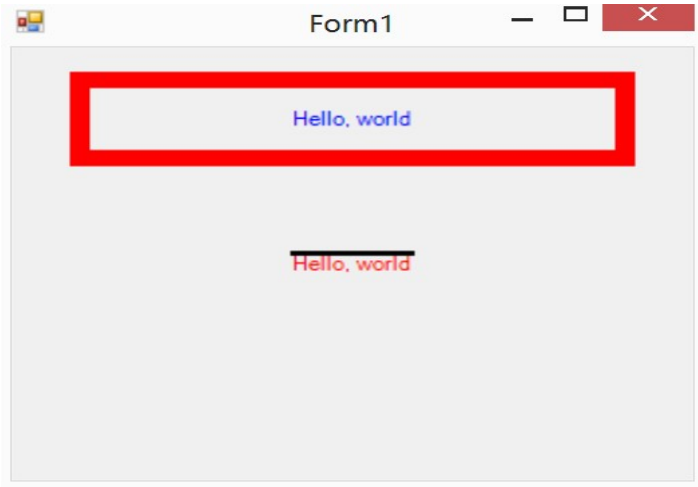


Рис. 6. До центрування тексту

Клас *SystemInformation* у просторі імен *System.WindowsForms* на даний момент вміщує 60 статичних незмінних властивостей, що надають відомості про комп'ютер, на якому виконується програма, і деяких показниках, використовуваних ОС для відображення елементів робочого столу та вашої програми. *SystemInformation* містить дані про кількість кнопок мишки, розмірі значків робочого столу і висоті рядка заголовку форми. У ньому також є відомості про підключення до мережі та імені домену, в якому знаходиться даний комп'ютер. Вся ця інформація представляється даними різного типу – *int*, *bool*, *string*, *Size*, *Rectangle* – і декількома переліченнями. Нижче приведена частина статичних незмінних властивостей класу *SystemInformation*:

ArrangeDirection – повертає значення, що показує напрям, в якому операційна система впорядковує згорнуті вікна.

ArrangeStartingPosition – повертає значення позиції розгорнутого вікна.

Border3DSize – отримує товщину (в точках) тривимірної границі вікна.

BootMode – значення, яке вказує режим завантаження системи.

BorderMultiplierFactor – отримує множник границі, який використовується при визначенні товщини границі для вимірювання розміру вікна.

BorderSize – отримує товщину (в точках) границі вікна.

CaptionButtonSize – повертає звичайний розмір (в точках) кнопки заголовку вікна.

CaptionHeight – отримує висоту (в точках) стандартної області заголовку вікна.

CaretBlinkTime – отримує час мерехтіння курсору.

CaretWidth – отримує ширину (в точках) курсору в полях введення елементів управління.

ComputerName – отримує ім’я комп’ютера.

CursorSize – повертає максимальний розмір, який курсор може займати.

DbcsEnabled – повертає значення, яке визначає можливість операційної системи обробляти символи у двобайтовому кодуванні.

DoubleClickSize – отримує розміри (в точках) області, в якій операційна система обробляє подвійне клацання.

DoubleClickTime – отримує максимальне число мілісекунд, яке може пройти між першим та другим клацанням, щоб операційна система розглядала ці дії мишки як подвійне клацання.

FixedFrameBorderSize – отримує товщину границі фрейму вікна із заголовком.

FontSmoothingContrast – отримує значення контрастності згладжування шрифту, використовуваного у згладжуванні *ClearType*.

FontSmoothingType – повертає поточний тип згладжування шрифтів.

IconHorizontalSpacing – отримує ширину крупних значків.

IconSize – отримує розміри (в точках) розміру значку програми за умовчанням.

IconSpacingSize – повертає розмір (в точках) елементу сітки, яка використовується для розміщення значків у режимі крупних значків.

IconVerticalSpacing – отримує висоту (в точках) комірки для розміщення значку у режимі крупних значків.

IsFlatMenuEnabled – повертає значення, що вказує чи має місце плоский зовнішній вигляд меню.

IsFontSmoothingEnabled – повертає значення, що вказує чи включений режим згладжування шрифту.

IsMenuAnimationEnabled – повертає значення, що вказує, чи будуть увімкнені функції анімації згасання або слайду меню.

IsMenuFadeEnabled – повертає значення, що вказує, чи включена анімація згасання меню.

MenuButtonSize – отримує розміри за умовчанням (в точках) для кнопок рядка меню.

MenuFont – отримує шрифт, що використовується для відображення тексту в меню.

MenuHeight – отримує висоту в точках однієї лінії меню.

MinimizedWindowSize – отримує розміри (в точках) для нормального згорнутого вікна.

MinimumWindowSize – повертає мінімальні ширину та висоту вікна (у пікселях).

MouseHoverSize – отримує розміри (в точках) прямокутника, в якому покажчик миші повинен залишатися під час наведення миші перед тим, як генерується повідомлення про наведення миші.

MouseHoverTime – повертає час у мілісекундах, протягом якого покажчик миші повинен залишатися у прямокутнику наведення перед тим, як генерується повідомлення про наведення миші.

MouseSpeed – повертає поточну швидкість миші.

MouseWheelPresent – повертає значення, що вказує, чи є миша з коліщатком.

Network – повертає значення, що вказує, чи є мережне підключення.

Secure – повертає значення, що вказує, чи є диспетчер безпеки в даній операційній системі.

SmallIconSize – отримує розміри (в точках) дрібного значку.

UserDomainName – повертає ім'я домену, до якого належить користувач.

VerticalScrollBarWidth – отримує ширину за умовчанням (в точках) вертикальної смуги прокрутки.

WorkingArea – повертає розмір у пікселях робочої області екрану.

Текст програми використання властивостей класу *SystemInformation* і результат її рішення наведено відповідно на лістингі 7 та рис. 7.

Лістинг 7. (exam07)

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

namespace exam7
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

        }
        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics gr = e.Graphics;
            SizeF sizef =
              gr.MeasureString("ArrangeStartingPosition", Font);
            int cySpace = Font.Height, sdvig = 300;
            int cxCol1, cxCol2, y;
            Brush brush = new SolidBrush(ForeColor);
```

```
            for (int i = 0; i < 2; i++)
            {
                y = 0;
                cxCol1 = i * sdvig;
                cxCol2 = cxCol1 + (int)sizef.Width;
                gr.DrawString("ArrangeDirection", Font, brush,
                              cxCol1, y);

gr.DrawString(SystemInformation.ArrangeDirection.ToString(),
                              Font, brush, cxCol2, y);
                y += cySpace;
      gr.DrawString("ArrangeStartingPosition", Font,
                brush, cxCol1, y);
      gr.DrawString(SystemInformation.ArrangeStartingPosition.
                    ToString(), Font, brush, cxCol2, y);
                y += cySpace;
      gr.DrawString("BootMode", Font, brush, cxCol1, y);
      gr.DrawString(SystemInformation.BootMode.ToString(), Font,
                brush, cxCol2, y);
                y += cySpace;
      gr.DrawString("Border3DSize", Font, brush, cxCol1, y);

gr.DrawString(SystemInformation.Border3DSize.ToString(), Font,
brush, cxCol2, y);
                y += cySpace;
      gr.DrawString("BorderSize", Font, brush, cxCol1, y);
      gr.DrawString(SystemInformation.BorderSize.ToString(),
                    Font, brush, cxCol2, y);
                y += cySpace;
     gr.DrawString("CaptionButtonSize", Font, brush, cxCol1, y);
     gr.DrawString(SystemInformation.CaptionButtonSize.
                   ToString(), Font, brush, cxCol2, y);
                y += cySpace;
     gr.DrawString("CaptionHeight", Font, brush, cxCol1, y);
     gr.DrawString(SystemInformation.CaptionHeight.
                   ToString(), Font, brush, cxCol2, y);
                y += cySpace;
     gr.DrawString("ComputerName", Font, brush, cxCol1, y);
     gr.DrawString(SystemInformation.ComputerName.
                    ToString(), Font, brush, cxCol2, y);
                y += cySpace;
     gr.DrawString("CursorSize", Font, brush, cxCol1, y);
     gr.DrawString(SystemInformation.CursorSize.
                   ToString(), Font, brush, cxCol2, y);
                y += cySpace;
     gr.DrawString("DbcsEnabled", Font, brush, cxCol1, y);
     gr.DrawString(SystemInformation.DbcsEnabled.
                   ToString(), Font, brush, cxCol2, y);
            }
        }
    }
}
```

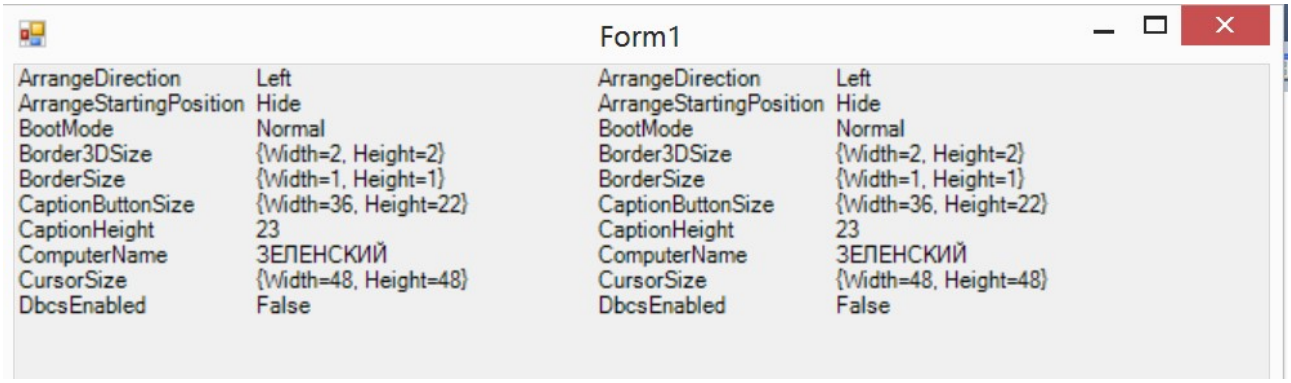


Рис. 7. До використання властивостей класу *SystemInformation*

Більш розширений приклад використання методів *DrawString*, *MeasureString* наводяться у лістингу 8. Результат рішення наводиться на рис. 8.

Лістинг 8. (exam08)

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

namespace exam8
{
    public partial class Form1 : Form
    {
        string[] str_is = { "A", "B",
        SystemInformation.Network.ToString(), Form1.dd() };

        public Form1()
        {
            InitializeComponent();
        }
        static string dd()
        {
            return "Привет";
        }
       private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics gr = e.Graphics;
            int y = 0;
            Brush br = new SolidBrush(ForeColor);
            foreach (string str in SysInfoStrings.Labels)
            {
                gr.DrawString(str, Font, br, 0, y);
                y += Font.Height;
```

```
        }
        gr.DrawString("Максимум - " + SysInfoStrings.
        MaxLabelWidth(gr, Font).ToString(),Font, br, 0, y);

        y += 2*Font.Height;

        foreach (string str in SysInfoStrings.Values)
        {
            gr.DrawString(str, Font, br, 0, y);
            y += Font.Height;
        }

        gr.DrawString("Максимум - " + SysInfoStrings.
        MaxValueWidth(gr, Font).ToString(),Font, br, 0, y);

        y += 2*Font.Height;
       foreach (string el_str in str_is)
        {
            gr.DrawString(el_str, Font, br, 0, y);
            y += Font.Height;
        }
    }
}

class SysInfoStrings
{
    public static string[] Labels
    {
        get
        {
           return new string[]
                      {"aaaaaaaa", "bbbb","cccc","dddd"};
        }
    }

    public  static string[] Values
    {
        get
        {
           return new string[]
                      { "5555555", "6666","7777", "8888" };
        }
    }

 public static float MaxLabelWidth(Graphics grfx, Font font)
  {
    return Maxwidth(Labels, grfx, font);
  }

 public static float MaxValueWidth(Graphics grfx, Font font)
  {
    return Maxwidth(Values, grfx, font);
  }
```

```
  static float Maxwidth(string[] astr, Graphics grfx, Font font)
   {
     float fMax = 0;
     foreach (string str in astr)
     fMax = Math.Max(fMax, grfx.MeasureString(str, font).Width);
     return fMax;
   }
  }
}
```

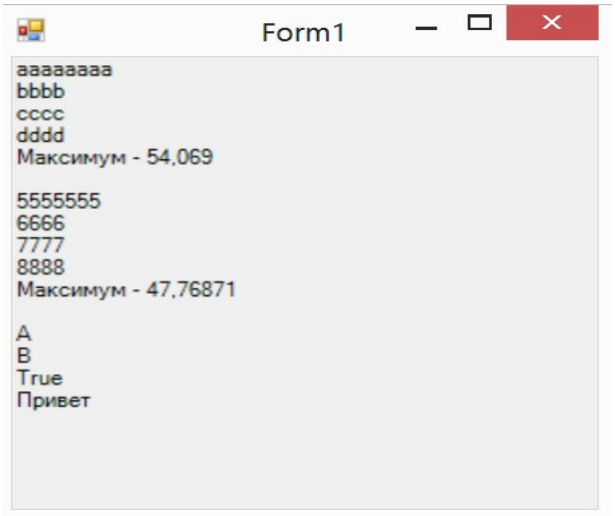


Рис. 8. Розширений приклад обробки рядкових даних

5. Робота зі скролінгом

Смуги прокрутки – важлива складова будь-якого графічного середовища. Вони покращують наочність та полегшують роботу користувача. Смуги прокрутки зручні при відображені довільних об'єктів (тексту, графіки, таблиць, записів баз даних, картинок, *Web*-сторінок), чиї розміри перевищують розміри клієнтської області програми. Смуги прокрутки можуть бути орієнтовані вертикально (для переміщення вверх-вниз) або горизонтально (вперед-назад). Клацання стрілок на кінцях смуги прокрутки переміщує документ на невелику відстань – зазвичай на один рядок вверх або вниз (для вертикальної смуги прокрутки). Клацання області, розташованої між стрілками, викликає переміщення на більшу відстань. Смуги прокрутки можна додати до форми двома способами.

Перший – створити елементи управління типу *VScrollBar* і *HScrollBar*. Робота смуг прокрутки налаштовується властивостями цих об'єктів. При використанні таких смуг прокрутки ініціюються певні події. Нижче ми будемо працювати з такими елементами управління.

Другий спосіб простіше. Його часто називають автопрокруткою (*autoscroll*), і саме його ми використаємо у даному розділі.

Засіб автопрокрутки в першу чергу призначений для програм, що розміщують у своїй клієнтській частині різні елементи управління (наприклад, кнопки і текстові поля). Включить автопрокрутку для форми можна, задавши

властивості *AutoScroll* значення *true*. Тоді, якщо розміру клієнтської області не вистачить для відображення усіх елементів управління, виникнуть смуги прокрутки і дозволять користувачу дістатися до інших елементів.

5.1. Побудова скролінгу по розташуванню дочірніх вікон

NET Framework містить масу цікавих елементів керування: від кнопок, списків та текстових полів, до календарів, дерев і таблиць. Але елемент «панель» не входить до їх числа. Панель не має візуального уявлення і не сильно впливає на інтерфейс користувача. Зазвичай панель застосовують у «архітектурних» цілях – щоб згрупувати елементи управління. Крім того, панелі бувають корисні при організації автопрокрутки. На рис. 9а у якості елементів керування використовуються панель і кнопка. Перший елемент управління визначає граничний кордон по горизонталі, другий – по вертикалі для автоматичного формування прокрутки. Текст програми і результат її рішення наведені відповідно на лістингах 9а, 9б та рис. 10. У лістингу 9а (файл *Form1.Designer.cs*) наводиться створення та ініціалізація об'єктів. У лістингу 9б (файл *Form1.cs*) наводиться програма, в якій об'єкт *Form1* (похідний від базового класу *Form*) і елементи управління (*panel*1, *button*1). В подальшому опис <имя >*Designer.cs* буде наводитись тільки у випадках необхідності.

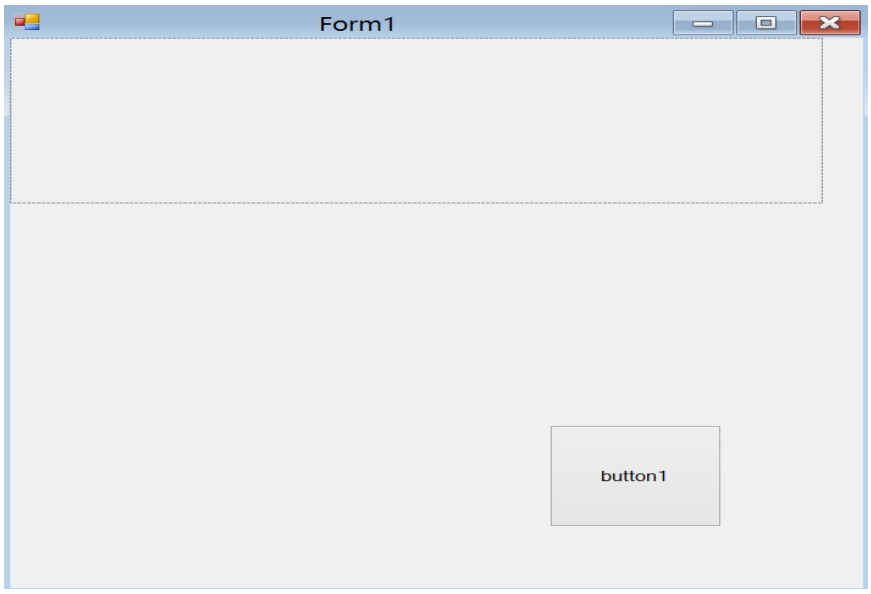


Рис. 9а. Розташування елементів управління

Лістинг 9а (exam09)

```
namespace exam9
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components =
null;

        protected override void Dispose(bool disposing)
        {
```

```
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

 private void InitializeComponent()
  {
     this.panel1 = new System.Windows.Forms.Panel();
     this.button1 = new System.Windows.Forms.Button();
     this.SuspendLayout();

                          // panel1
     this.panel1.Location = new System.Drawing.Point(0, 0);
     this.panel1.Name = "panel1";
     this.panel1.Size = new System.Drawing.Size(444, 125);
     this.panel1.TabIndex = 0;
     this.panel1.Paint += new
     Sytem.Windows.Forms.PaintEventHandler(this.PanelOnPaint);
            //
            // button1
            //
  this.button1.Location = new System.Drawing.Point(295, 292);
  this.button1.Margin = new System.Windows.Forms.Padding(2);
  this.button1.Name = "button1";
  this.button1.Size = new System.Drawing.Size(94, 77);
  this.button1.TabIndex = 1;
  this.button1.Text = "button1";
  this.button1.UseVisualStyleBackColor = true;
            //
            // Form1
            //
  this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
  this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
  this.ClientSize = new System.Drawing.Size(466, 415);

  this.Controls.Add(this.button1);
  this.Controls.Add(this.panel1);
  this.Name = "Form1";
  this.Text = "Form1";

  this.ResumeLayout(false);
  }
        #endregion

        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.Button button1;
    }
}
```

Лістинг 9б (exam09)

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

namespace exam9
{
    public partial class Form1 : Form
    {
        readonly float cxCol;
        readonly int cySpace;

        public Form1()
        {
            InitializeComponent();
            Text = "System Information: Panel";
            BackColor = SystemColors.Window;
            ForeColor = SystemColors.WindowText;
            Graphics grfx = this.CreateGraphics();
            SizeF sizef = grfx.MeasureString(" ", Font);
            cySpace = Font.Height;
            // отступы при скроллинге
            AutoScrollMargin = new Size(0, 0);
            // скроллинг по элементам управления
            AutoScroll = true;
            HorizontalScroll.SmallChange = (int)sizef.Width;
            VerticalScroll.SmallChange = cySpace;
            HorizontalScroll.LargeChange = 50;
            VerticalScroll.LargeChange = 50;
            cxCol = SysInfoStrings.MaxLabelWidth(grfx, Font);
            panel1.BackColor = Color.FromArgb(255,0,0);
            panel1. Location = Point. Empty;
            panel1. Size = new Size((int) Math.Ceiling(cxCol +
            SysInfoStrings.MaxValueWidth(grfx, Font)),
         (int) Math.Ceiling(1.0*cySpace *
                                SysInfoStrings.Labels.Count()));

        }

        void PanelOnPaint(object obj, PaintEventArgs pea)
        {
            Graphics grfx = pea.Graphics;
            Brush brush = new SolidBrush(ForeColor);
            int iCount = SysInfoStrings.Labels.Count();

            string[] astrLabels = SysInfoStrings.Labels;
            string[] astrValues = SysInfoStrings.Values;
```

```
            for (int i = 0; i < iCount; i++)
            {
                grfx.DrawString(astrLabels[i], Font, brush,
                0, i * cySpace);
                grfx.DrawString(astrValues[i], Font, brush,
                cxCol, i * cySpace);
            }
        }

        class SysInfoStrings
        {
            public static string[] Labels
            {
                get
                {
                    return new string[]
                    {"11111111","22222222", "333333333",
                    "4444444444", "555555555555",
                    "66666666666666666666666666666666666666K"};
                }
            }

            public static string[] Values
            {
                get
                {
                    return new string[]
                    {"11111111","22222222", "333333333",
                    "4444444444", "555555555555",
                    "66666666666666666666666666666666666666K"};
                }
            }

      public static float MaxLabelWidth(Graphics grfx, Font font)
      {
       return Maxwidth(Labels, grfx, font);
      }

      public static float MaxValueWidth(Graphics grfx, Font font)
       {
         return Maxwidth(Values, grfx, font);
       }

  static float Maxwidth(string[] astr, Graphics grfx, Font font)
   {
     float fMax = 0;
     foreach (string str in astr)
     fMax = Math.Max(fMax, grfx.MeasureString(str, font).Width);
     return fMax;
   }
  }
 }
}
```

При автопрокрутці діапазон значень задається різницею ширини і висоти клієнтської частини та ширини і висоти області розташування елементів управління (у нашому випадку, елементів *Panel* і *Button*). До цього значення додається різниця між величиною *AutoScrollMargin* та шириною і высотою клієнтської частини. Смуги прокрутки, створені як окремі елементи управління, генерують подію *Scroll*, коли користувач працює із смугою прокрутки. При автопрокрутці така подія не виникає, отримати до неї доступ з додатку неможна. Розглянута програма напряму не обробляє події *Scroll*, вона обробляє події *Paint* для панелі. Коли програма виводить інформацію на елемент управління, промальовується лише видима область цього елементу.

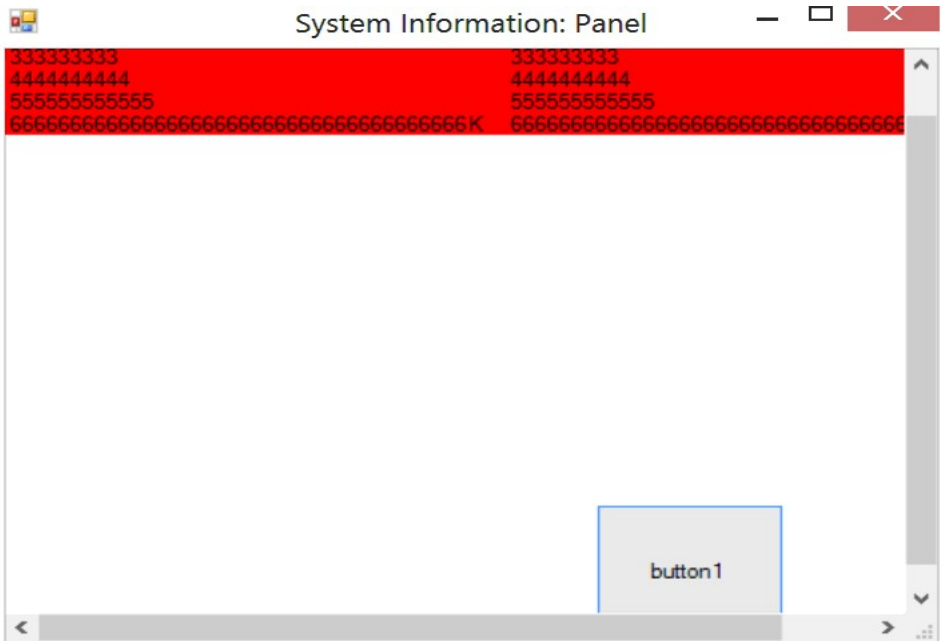


Рис.10. До організації скролінгу по розташуванню дочірніх вікон

Серед багатьох класів, які спадкують клас *Form*, є ***ScrollableControl***, властивості якого наведені в табл. 6. Тут перші дві описані вище при програмуванні автопрокрутки

Таблиця 6.

Властивості класу *ScrollableControl*

Тип	Ім’я	Опис
bool	*AutoScroll*	Наявність автоматичного скролінгу по елементам управління
Size	*AutoScrollMargin*	Встановлює відступ праворуч і знизу від елементу управління
bool	*HScroll*	Наявність горизонтальної смуги прокрутки
bool	*VScroll*	Наявність вертикальної смуги прокрутки.
Size	*AutoScrollMinSize*	Задає мінімальну область прокрутки
Point	*AutoScrollPosition*	Вказує положення смуги прокрутки

5.2. Створення автопрокрутки без створення дочірніх елементів управління

Ми розглядали створення автопрокрутки по розташуванню дочірніх елементів шляхом привласнення властивості *AutoScroll* значення *true*. Створення автопрокрутки без створення дочірніх елементів управління виконується використанням властивості *AutoScrollMinSize.* Цій властивості привласнюється розмір відображення всієї виведеної інформації.

AutoScrollPosition містить поточне положення прокрутки у від'ємних координатах.

У даному випадку нульові координати будуть відповідати лівій та верхній границі виведеного документу, а не клієнтського вікна.

На лістингу 10 рис. 11 наводиться програма і результат рішення задачі створення автопрокрутки без використання елементів управління. Організація скролінгу виконана на основі використання властивостей *AutoScrollMinSize* і *AutoScrollPosition*

Лістинг 10 (exam10)

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

namespace exam10{
    public partial class Form1 : Form
    {
        float cxCol;
        int cySpace;

        public Form1()
        {
            InitializeComponent();
        }

        void UpdateAll()
        {
            Graphics gr = this.CreateGraphics();

          cxCol = SysInfoStrings.Maxwidth(SysInfoStrings.Labels,
                  gr, Font);
            cySpace = Font.Height;
            this.AutoScrollMinSize = new Size(
            (int)cxCol + (int)SysInfoStrings.Maxwidth
            (SysInfoStrings.Values, gr, Font),
            (int)(cySpace * SysInfoStrings.Values.Count()));
            gr.Dispose();
        }
```

```
   private void Form1_Paint(object sender, PaintEventArgs e)
   {
     Graphics gr = e.Graphics;
     Brush brush = new SolidBrush(ForeColor);
     Point pt = AutoScrollPosition;

     for (int i = 0; i < SysInfoStrings.Labels.Count(); i++)
      {
        gr.DrawString(SysInfoStrings.Labels[i], Font, brush,
            0 + pt.X, i * cySpace + pt.Y);
        gr.DrawString(SysInfoStrings.Values[i], Font, brush,
                cxCol + pt.X, i * cySpace + pt.Y);
            }
        }
}

class SysInfoStrings
{
   public static string[] Labels
    {
        get
        {
            return new string[]
            {
                "ArrangeDirection",
                "ArrangeStartingPosition",
                "BootMode",
                "Border3DSize",
                "BorderMultiplierFactor",
                "BorderSize",
                "CaptionButtonSize",
                "CaptionHeight",
                "CaretBlinkTime",
                "CaretWidth",
                "ComputerName",
                "CursorSize",
                "DbcsEnabled",
                "DebugOS",
                "DoubleClickSize",
                "DoubleClickTime",
                "DragFullWindows",
                "DragSize",
                "FixedFrameBorderSize",
                "FontSmoothingContrast",
                "MenuButtonSize",
                "MenuHeight"
             };
         }
    }
```

```
    public static string[] Values
    {
     get
      {
        return new string[]
        {
            SystemInformation.ArrangeDirection.ToString(),
            SystemInformation.ArrangeStartingPosition.ToString(),
            SystemInformation.BootMode.ToString(),
            SystemInformation.Border3DSize.ToString(),
            SystemInformation.BorderMultiplierFactor.ToString(),
            SystemInformation.BorderSize.ToString(),
            SystemInformation.CaptionButtonSize.ToString(),
            SystemInformation.CaptionHeight.ToString(),
            SystemInformation.CaretBlinkTime.ToString(),
            SystemInformation.CaretWidth.ToString(),
            SystemInformation.ComputerName.ToString(),
            SystemInformation.CursorSize.ToString(),
            SystemInformation.DbcsEnabled.ToString(),
            SystemInformation.DebugOS.ToString(),
            SystemInformation.DoubleClickSize.ToString(),
            SystemInformation.DoubleClickTime.ToString(),
            SystemInformation.DragFullWindows.ToString(),
            SystemInformation.DragSize.ToString(),
            SystemInformation.FixedFrameBorderSize.ToString(),
            SystemInformation.FontSmoothingContrast.ToString(),
            SystemInformation.MenuButtonSize.ToString(),
            SystemInformation.MenuHeight.ToString()
         };
       }
      }

    public static float Maxwidth(string[]astr,Graphics grfx,
                                 Font font)
      {
        float fMax = 0;
        foreach (string str in astr)
      fMax = Math.Max(fMax, grfx.MeasureString(str, font).Width);
      return fMax;
         }
     }
}
```

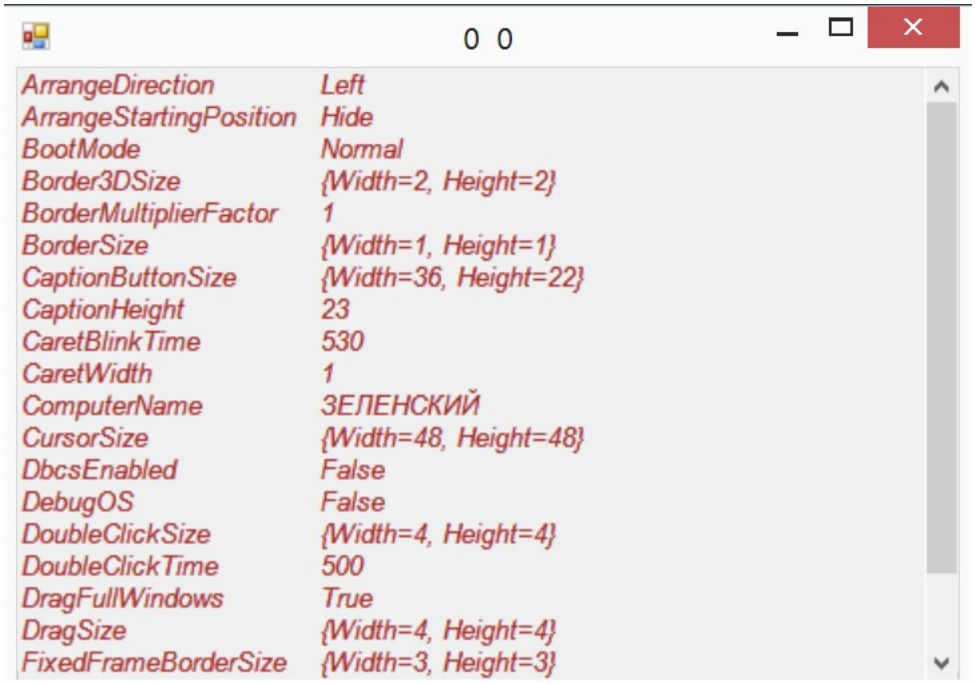


Рис. 11. До організації скролінгу по розміру інформації у документі

5.3. Управління скролінгом клавіатурним інтерфейсом

Властивість *AutoScrollPosition* реалізовано у об'єкті *ScrollableControl* (нащадком якого є *Form)* як засіб підтримки автоматичної прокрутки. Значенням *AutoScrollPosition* є структура *Point,* яка вказує положення двох смуг прокрутки. При отримання значення *AutoScrollPosition* значення координат від'ємні та вказують положення віртуальної клієнтської області відносно верхнього лівого кута фізичної клієнтської області. Однак при встановленні *AutoScrollPosition* координати повинні бути додатними. Для цього потрібно додати до програми два рядки:

```
pt. X = - pt.X;
pt. Y = - pt.Y;
```

А можна і інакше: просто налаштувати координати на основі будь-якої клавіші керування курсором. При натисканні лівої або правої стрілки одночасно з *Ctrl* зсувається клієнтська область на її ширину або, якщо клавіша *Ctrl* не натиснута, – на величину висоти символу. Дія інших клавіш керування курсором зроблено незалежним від стану модифікаторів. Клавіша *Home* повертає курсор на початок списку, *End* – переміщує його в кінець списку, не змінюючи його положення по горизонталі. Програма наведена у лістингу 11, результат виконання програми наведено на рис. 12.

Лістинг 11. (examp 11)

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

namespace exam11
{
    public partial class Form1 : Form
    {
        int w,ws;
        public Form1()
        {
            InitializeComponent();
            w = 400;
            Graphics grfx = this.CreateGraphics();
            ws = (int)grfx.MeasureString("9", Font).Width;
            grfx.Dispose();
            //Size = new Size(w,w);
            AutoScrollMinSize = new Size(w,w);
         }

protected override void  OnKeyDown(KeyEventArgs kea)
        {
            base.OnKeyDown(kea);
            Point pt = AutoScrollPosition;
            pt.X = -pt.X;
            pt.Y = -pt.Y;
            Text = pt.X.ToString() + "  " + pt.Y.ToString();
          switch (kea.KeyCode)
            {
                case Keys.Right:
                    if (kea.Modifiers == Keys.Control)
                        pt.X = w;
                    else
                        pt.X += ws;
                    break;
                case Keys.Left:
                    if (kea.Modifiers == Keys.Control)
                        pt.X = 0;
                    else
                        pt.X -= ws;
                    break;
                case Keys.Down:
                    pt.Y += Font.Height; break;
                case Keys.Up:
                    pt.Y -= Font.Height; break;
                case Keys.PageDown:
```

```
                    pt.Y += ClientSize.Height;
                    break;
                case Keys.PageUp:
                    pt.Y -= ClientSize.Height;
                    break;
                case Keys.Home:
                    pt = Point.Empty; break;
                case Keys.End:
                    pt.X = pt.Y = w;
                    break;
            }
            AutoScrollPosition = pt;
        }

  private void Form1_Paint(object sender, PaintEventArgs e)
  {
    Graphics grfx = e.Graphics;
    Point pt = this.AutoScrollPosition;
    grfx.DrawLine(new Pen(Color.Black,5.0f), pt.X, pt.Y,
                  pt.X + w, pt.Y + w);
    grfx.DrawLine(new Pen(Color.Black, 5.0f), w+pt.X, pt.Y,
                  pt.X, pt.Y + w);
    //Text = pt.ToString();
  }
 }
}
```

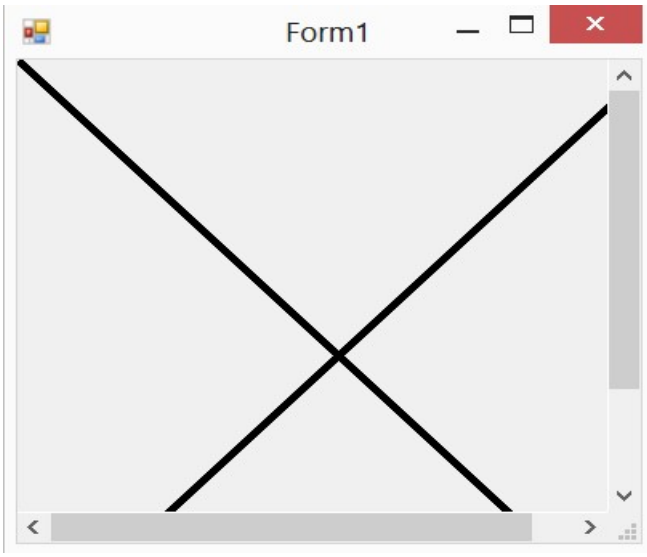


Рис. 12. До керування скролінгом клавіатурним інтерфейсом

6. Програмування друку

6.1. Технологія організації друку

Розглянемо виведення на друк посередині сторінки текст "Hello Printer!". Для цього використовується елемент управління *PrintDocument*, який показано на рис. 13. і має ім'я *printDocument1*. Текст програми наводиться на лістингу 12.

Після натиснення кнопки викликається метод *PrintDocument*1.*Print()*. У результаті виконання методу викликається обробник події *PrintPage* (названий тут *PrintDocument*1*_PrintPage*).

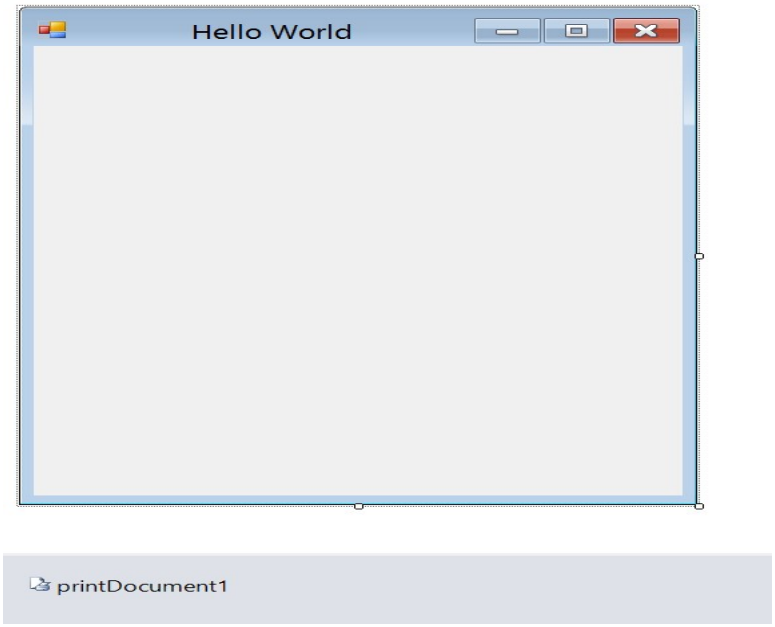


Рис. 13. До організації друку

Значенням параметру *object* обробника *PrintDocument*1*_PrintPage* є раніше створений об'єкт *PrintDocument.* Параметр *PrintPageEventArgs* визначає властивості, що надають відомості про принтер, найважливіша з яких – *Graphics* – аналогічно однойменній властивості параметру *PaintEventArgs* за виключенням того, що *PrintPageEventArgs* надає об'єкт *Graphics* для сторінки принтеру, а не для клієнтської області форми. Об'єкт *Graphics* викликає методи, що виводять графіку на сторінку принтера. Якщо треба надрукувати декілька сторінок, слід встановити властивість *HasMorePages* об'єкту *PrintPageEventArgs* як *true.* Але оскільки друкується лише одна сторінка, залишимо значення властивості за умовчанням – *false.* Метод *DoPage,* викликається як обробником *OnPaint,* так і *PrintDocumentOnPrintPage.*

Як відомо, властивість *ClientSize* форми надає розміри її клієнтської області, виражені у пікселах. Створення графіки в межах клієнтської області вікна відрізняється від виведення графіки на принтер. Сторінка принтера визначається трьома різними областями.

По-перше, повним розміром сторінки. Цю інформацію надає властивість *PageBounds* класу *PrintPageEventArgs.* Його значенням є структура *Rectangle,* її властивості *X* і *Y* дорівнюють 0, а властивість *Width* і *Height* надають розміри паперу, які за умовчанням, вимірюються в сотих долях дюйма. Наприклад, для листа розміром 8,5 и 11 дюймів значення властивостей *Width* і *Height* об'єкту *PageBounds* дорівнюватиме 850 і 1100. Якщо у принтері за умовчанням задана альбомна, а не книжкова орієнтація сторінки, то вони дорівнюватимуть відповідно 1100 і 850.

По-друге, є *область друку* сторінки. Її розмір зазвичай дуже близький до повного розміру сторінки, окрім полів, недоступних друкуючій головці принтера. Ширина верхнього, нижнього, лівого та правого полів може бути різною. Значенням властивості *VisibleClipBounds* класу *Graphics* є структура *RectangleF,* що надає розмір області друку сторінки. Властивості *X і Y* цієї структури встановлюються в 0, а її властивості *Width* і *Height* дорівнюють горизонтальному та вертикальному розмірам області друку сторінки.

Розмір третьої області розраховується з урахуванням 1-дюймового відступу по периметру сторінки і являє собою границі, в межах яких воліє друкувати користувач. Ця інформація повертається у вигляді структури *Rectangle* властивістю *MarginBounds* об'єкту *PrintPageEventArgs.*

Лістинг 12. (exam12)

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

namespace exam12{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
            printDocument1.DocumentName = "Doc1";
             Text = "Hello Printer!";
        }

        private void Form1_Click(object sender, EventArgs e)
        {
            printDocument1.Print(); // Печать
        }

        private void printDocument1_PrintPage(object sender,
```

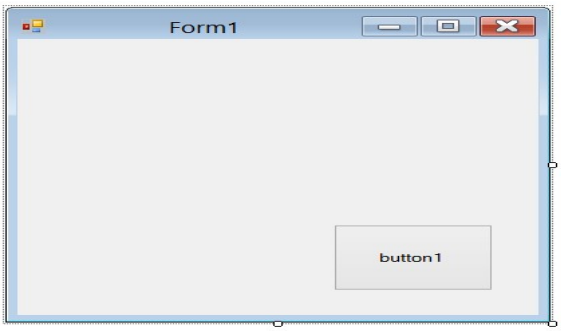
```
            PrintPageEventArgs e)
        {
            DoPage(e.Graphics, Color.Black,
            (int)e.Graphics.VisibleClipBounds.Width,
                   (int)e.Graphics.VisibleClipBounds.Height);
        }
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
          DoPage(e.Graphics, Color.FromArgb(255, 0, 0),
          ClientSize.Width/6, ClientSize.Height/6);
        }
        private void DoPage(Graphics grfx, Color col,int w,int h)
        {
          Pen pen = new Pen(col);
          Rectangle rect = new Rectangle(1,1,w-2, h-2);
          StringFormat ss = new StringFormat();
          ss.Alignment = StringAlignment.Center;
          ss.LineAlignment = StringAlignment.Center;
          grfx.DrawRectangle(pen, rect);
          grfx.DrawString(Text, Font, Brushes.Black, rect, ss);
        }
    }
}
```

6.2. Виведення на друк текстового файлу

У лістингі 12 наводиться програма зчитування даних з текстового файлу та виведення цієї інформації на принтер.

Розглянемо виведення на друк даних текстового файлу. Для цього використовується елемент управління *PrintDocument*, який показано на рис. 14 і має ім’я *printDocument*1. Текст програми зчитування даних з текстового файлу і виведення цієї інформації на принтер наводиться у лістингу 13.

Після натиснення кнопки на принтер виводяться дані текстового файлу 1.txt.



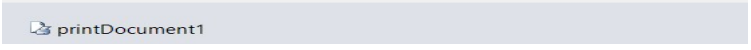


Рис. 14. До виведення на друк текстового файлу

Лістинг 13. (exam13)

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

namespace exam13
{
    public partial class Form1 : Form
    {
        Font printFont;
        StreamReader streamToPrint;
        int kod;

        public Form1()
        {
            InitializeComponent();
            printFont = new Font("Arial Cyr", 10);
            kod = 1;
            try
            {
                streamToPrint = new StreamReader("1.txt",
                Encoding.Default);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                kod = 0;
            }
        }
  private void printDocument1_PrintPage(object sender,
     System.Drawing.Printing.PrintPageEventArgs ev)
   {
    float linesPerPage = 0;
    float yPos = 0;
    int count = 0;
    float leftMargin = 100 + ev.Graphics.VisibleClipBounds.Left;
    float topMargin =  ev.Graphics.VisibleClipBounds.Top;
    string line = null;
    linesPerPage = ev.Graphics.VisibleClipBounds.Height /
            printFont.GetHeight(ev.Graphics);
         while (count < linesPerPage &&
               ((line = streamToPrint.ReadLine()) != null))
         {
          yPos = topMargin + (count *
          printFont.GetHeight(ev.Graphics));
          ev.Graphics.DrawString(line, printFont, Brushes.Black,
             leftMargin, yPos);
```

```
            count++;
          }
            ev.HasMorePages = line != null;
        }
          void button1_Click(object sender, EventArgs e)
          {
              if (kod == 0) return;
                    try
                  {
                        streamToPrint.BaseStream.Position = 0;
                        printDocument1.Print();
                  }
                    catch (Exception ex)
                  {
                      MessageBox.Show(ex.Message);
                  }
          }
          private void Form1_FormClosed(object sender,
                                        FormClosedEventArgs e)
          {
              if(kod==1)streamToPrint.Close();
          }
    }
}
```

Будь-який текстовий редактор повинен мати можливість друку на принтері. Наступна програма має скромні можливості: відкрити у стандартному діалозі текстовий файл, переглянути його у текстовому полі без можливості зміни тексту (*ReadOnly*) і при можливості вивести цей текст на принтер. Використані наступні елементи управління: текстове поле *TextBox*, меню *MenuStrip* з пунктами меню: «Открыть», «Печать» и «Выход», а також елементи управління *OpenFileDialog*, *PrintDialog* (див. рис. 15). Текст програми надано у лістингі 14.

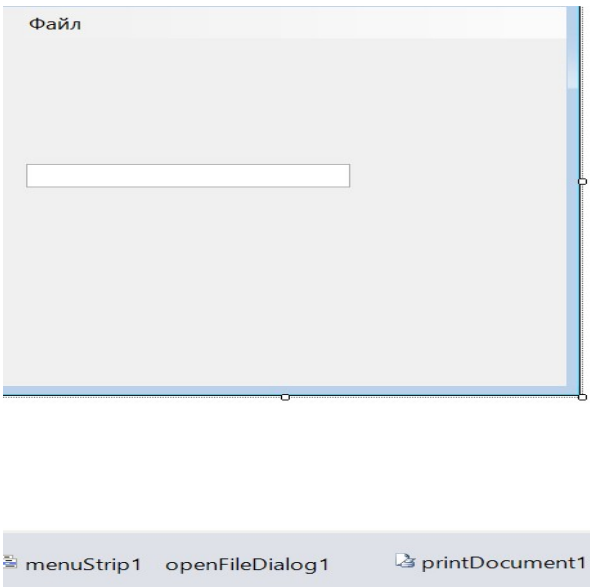


Рис. 15. До друку текстового документа з використанням меню и елементу *openFileDialog*

Лістинг 14. (exam14)

```
using System;
using System.Drawing;
using System.Windows.Forms;
// Інші директиви using видалені, оскільки вони
// не використовуються у даній програмі

namespace exam14
{
    public partial class Form1 : Form
    {
        System.IO.StreamReader Читатель;
        public Form1()
        {
          InitializeComponent();
          base.Text = "Открытие текстового файла и его печать";
          textBox1.Multiline = true;
          textBox1.Clear();
          textBox1.Size = new System.Drawing.Size(268, 112);
          textBox1.ScrollBars = ScrollBars.Vertical;
          textBox1.ReadOnly = true;
         // До тих пір, поки файл не прочитаний у текстове поле,
         // не повинен бути видно пункт меню "Печать..."
            печатьToolStripMenuItem.Visible = false;
            openFileDialog1.FileName = null;
        }
      private void открытьToolStripMenuItem_Click(object sender,
                                                  EventArgs e)
        { // Клацання на пункті меню "Открыть":
           openFileDialog1.Filter =
            "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
            openFileDialog1.ShowDialog();
            if (openFileDialog1.FileName == null) return;
          try
           { //Створення потоку StreamReader для читання з файлу
             Читатель = new
                System.IO.StreamReader(openFileDialog1.FileName,
                System.Text.Encoding.GetEncoding(1251));
                // - тут замовлення кодової сторінки Win1251 для
                // кирилиці
                textBox1.Text = Читатель.ReadToEnd();
                Читатель.Close();
                печатьToolStripMenuItem.Visible = true;
            }
           catch (System.IO.FileNotFoundException Exc)
           {
             MessageBox.Show(Exc.Message + "\nНет такого файла",
                             "Ошибка", MessageBoxButtons.OK,
                              MessageBoxIcon.Exclamation);
            }
            catch (Exception Exc)
            {
```

```
                // Звіт про інші помилки:
                MessageBox.Show(Exc.Message, "Ошибка",
                    MessageBoxButtons.OK,
                     MessageBoxIcon.Exclamation);
            }
        }
        private void печатьToolStripMenuItem_Click(object sender,
                                                   EventArgs e)
        { // Пункт меню "Печать"
            try
            {
                Читатель = new
                System.IO.StreamReader(openFileDialog1.FileName,
                System.Text.Encoding.GetEncoding(1251));
                // - тут замовлення кодової сторінки Win1251 для
                // кирилиці
                try { printDocument1.Print(); }
                finally { Читатель.Close(); }
            }
            catch (Exception ex) { MessageBox.Show(ex.Message);
 }
}

        private void printDocument1_PrintPage(object sender,
                System.Drawing, Printing.PrintPageEventArgs e)
        {
            Single linesPerPage = 0;
            Single yPos = 0; int count = 0;
            Single leftMargin = e.MarginBounds.Left;
            Single topMargin = e.MarginBounds.Top;
            topMargin = 0;
            String line = null;
            Font printFont = new Font("Times New Roman", 12.0F);
            // Обчислюємо кількість рядків на одній сторінці
            linesPerPage = e.MarginBounds.Height /
            printFont.GetHeight(e.Graphics);
            // Друкуємо кожний рядок файлу
            while (count < linesPerPage)
            {
                line = Читатель.ReadLine();
                if (line == null) break; // вихід з циклу
                yPos = topMargin + count *
                       printFont.GetHeight(e.Graphics);
                // Друк рядку
                e.Graphics.DrawString(line, printFont,
           Brushes.Black, leftMargin, yPos, new StringFormat());
                count += 1;
            }
            // Друк наступної сторінки, якщо є ще рядки файлу
            if (line != null) e.HasMorePages = true;
            else e.HasMorePages = false;
        }
```

```
        private void выходToolStripMenuItem_Click(object sender,
                                                  EventArgs e)
        { // Вихід з програми
            base.Close();
        }
    }
}
```

Тут при обробці події форми *Form1_Load* забороняється користувачеві редагувати текстове поле: *ReadOnly = true*. Також призначаємо властивості друк *ToolStripMenuItem.Visible = false* (пункт меню «Печать»), тобто на початку програми пункт меню «Печать» користувачеві не видно. При натисканні пункту меню «Открыть» викликається стандартний діалог *OpenDialog* і виконується читання текстового файлу з використанням потоку *StreamReader*. Після зчитування файлу стає видимим пункт «Печать». Зверніть увагу на обробку події *PrintDocument*1.*PrintPage*. Тут у змінній *count* відбувається підрахунок рядків на сторінці *linesPerPage*. При перевищенні кількості рядків на сторінці відбувається вихід з циклу, оскільки сторінка роздрукована. Якщо є ще сторінки (l*ine*!= *null*), то змінній *e.HasMorePages* призначаємо *true*, що ініціює знову подію *PrintPage* і підпрограма *PrintDocument*1.*Print* починає свою роботу знову. І так, поки не роздрукуються всі сторінки (*e.HasMorePages = false*). На рис.16. показано результат роботи програми.

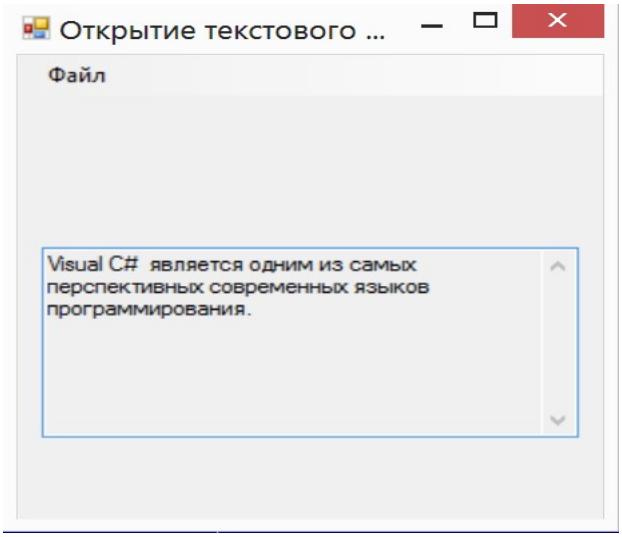


Рис.16. Результат робото програми друку текстового файлу

6.3. Виведення на друк 10 сантиметрової лінійки

У лістингу 15 наводиться програма, яка дозволяє з високою точністю вивести на принтер 10 сантиметрову лінійку. Така ж лінійка виводиться на екран. Однак якщо виміряти на екрані довжину лінійки, вона більш ніж 10 см. Чим менше роздільна здатність екрану, тим довше буде лінійка на екрані. При самій максимальній роздільній здатності екрану розміри лінійки наближаються до 10 см. Було б непогано знати роздільну здатність дисплея,

виражене у загальних одиницях виміру, наприклад в точках на дюйм (*dpi*), Але якщо для принтерів цей параметр визначений цілком точно (зазвичай його можна знайти навіть на упаковці принтеру), то для моніторів він залишається досить розпливчастим. Реальна роздільна здатність дисплея в *dpi* визначається двома параметрами: фізичним розміром екрану (зазвичай його вимірюють в дюймах по діагоналі) і роздільною здатністю екрану у пікселях. Оскільки Windows не знає розміру монітору, вона не може повідомити реальну роздільну здатність екрану. Навіть маючи такі відомості, ОС опиниться у скруті. Тому роздільна здатність екрану оцінюється у кількості пікселів на логічний дюйм. Цими показниками для екрану є властивості класу *Graphics DpiX* і *DpiY* (відповідно по горизонталі і вертикалі).

У лістингу 15 для екрану наводиться конвертування з міліметрів у пікселі:

```
PointF Conv(Graphics grfx, PointF pointf)
 {
   pointf.X*= grfx.DpiX  / 25.4f;
   pointf.Y *= grfx.DpiY / 25.4f;
   return pointf;
 }
```

Якщо за умовчанням виведення на екран виконується у пікселях, то на принтер у сотих долях реального дюйма. Для принтера конвертування з міліметрів у соті долі дюйма виконується в наступній функції:

```
PointF Conv_pr(Graphics grfx, PointF pointf)
 {
   pointf.X *= 100 / 25.4f;
   pointf.Y *= 100 / 25.4f;
   return pointf;
 }
```

Результат рішення наводиться на рис. 17.

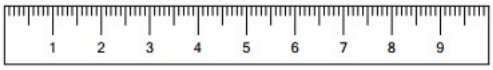
Лістинг 15. (exam15)

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

  namespace exam15
  {
    public partial class Form1 : Form
    {
```

```
        public Form1()
        {
          InitializeComponent();
          Text = "Печать";
        }

       private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics grfx = e.Graphics;
            Pen pen = new Pen(ForeColor);
            Brush brush = new SolidBrush(ForeColor);
            const int xOff = 10;
            const int yOff = 10;
            grfx.DrawPolygon(pen, new PointF[]
              {
                Conv(grfx, new PointF(xOff, yOff)),
                Conv(grfx, new PointF(xOff + 100, yOff)),
                Conv(grfx, new PointF(xOff + 100, yOff + 12)),
                Conv(grfx, new PointF(xOff, yOff + 12))
              }
        );

StringFormat  fmt = new StringFormat();
fmt.Alignment =  StringAlignment.Center;
//fmt.LineAlignment = StringAlignment.Near;

for (int i=1; i < 100; i++)
{
    if (i % 10 == 0) // Сантиметрові ділення
     {
        grfx.DrawLine(pen,
        Conv(grfx, new PointF(xOff + i, yOff)),
        Conv(grfx, new PointF(xOff + i, yOff + 6)));

        grfx.DrawString((i/10).ToString(), Font, brush,
        Conv(grfx, new PointF(xOff + i, yOff + 7)),fmt);
     }
}

   PointF Conv(Graphics grfx, PointF pointf)
    {
        pointf.X*= grfx.DpiX  / 25.4f;
        pointf.Y *= grfx.DpiY / 25.4f;
        return pointf;
    }

   PointF Conv_pr(Graphics grfx, PointF pointf)
    {
        pointf.X *= 100 / 25.4f;
        pointf.Y *= 100 / 25.4f;
        return pointf;
    }
```

```
  private void printDocument1_PrintPage(object sender,
  System.Drawing.Printing.PrintPageEventArgs e)
  {
    Graphics grfx = e.Graphics;
    Pen pen = new Pen(ForeColor);
    Brush brush = new SolidBrush(ForeColor);
    const int xOff = 10;
    const int yOff = 10;
    grfx.DrawPolygon(pen, new PointF[]
    {
      Conv_pr(grfx, new PointF(xOff, yOff)),
      Conv_pr(grfx, new PointF(xOff + 100, yOff)),
      Conv_pr(grfx, new PointF(xOff + 100, yOff + 12)),
      Conv_pr(grfx, new PointF(xOff, yOff + 12))
    }
    );

    StringFormat fmt = new StringFormat();
    fmt.Alignment = StringAlignment.Center;
    for (int i = 1; i < 100; i++)
    {
        if (i % 10 == 0) // Сантиметрові ділення
        {
            grfx.DrawLine(pen,
            Conv_pr(grfx, new PointF(xOff + i, yOff)),
            Conv_pr(grfx, new PointF(xOff + i, yOff + 6)));
            grfx.DrawString((i / 10).ToString(), Font, brush,
            Conv_pr(grfx, new PointF(xOff + i, yOff + 7)), fmt);
        }
        else if (i % 5 == 0) // Ділення по 5 мм
        {
            grfx.DrawLine(pen,
            Conv_pr(grfx, new PointF(xOff + i, yOff)),
            Conv_pr(grfx, new PointF(xOff + i, yOff + 4)));
        }
        else // Міліметрові ділення
        {
            grfx.DrawLine(pen,
            Conv_pr(grfx, new PointF(xOff + i, yOff)),
            Conv_pr(grfx, new PointF(xOff + i, yOff + 2)));
        }
    }
}
      private void button1_Click(object sender, EventArgs e)
      {
        this.printDocument1.Print();
      }
   }
}
```

а)



б)

Рис. 17. Виведення лінійки на екран (а) і принтер (б)

7. Одиниці вимірювання та масштабування сторінки

7.1. Одиниці вимірювання

Щоб уникнути написання методів конвертації даних, таких як *Conv* і *Conv_pr,* які були наведені у попередньому прикладі, включений механізм автоматичного масштабування до заданого розміру. Цей механізм масштабує координати за допомогою привласнення різних значень властивостям *PageUnit* та *PageScale* класу *Graphics:*

Властивості *PageUnit привласнюється* одне із значень перелічення *GraphicsUnit:*

Таблиця 7

Перелічення *GraphicsUnit*

Член	Значення	Опис
Display	1	Те ж, що *Pixel* для дисплею; для принтера дорівнює 1/100 дюйма (задано за умовчанням для принтера).
Pixel	2	Одиниці вимірювання – пікселі (задано за умовчанням)
Point	3	Одиниці вимірювання – пункти (1/72 дюйма)
Inch	4	Одиниці вимірювання – дюйми
Document	5	Одиниці вимірювання – 1/300 дюйма
Millimeter	6	Одиниці вимірювання – міліметри

Наприклад, якщо ви хочете малювати в координатах, обчислюваних сотими долями дюйма, цій парі властивостей треба задати такі значення:

```
grfx.PageUnit = GraphicsUnit.Inch;
grfx.PageScale = 0.01,
```

що еквівалентно вказівкою рахувати одне ділення координатної вісі рівним 0,01 дюйма. Після цього наступний виклик методу *DrawLine* намалює лінію довжиною в 1 дюйм:

```
grfx,DrawLine(pen , 0, 0, 100, 0);
```

Якщо надрукувати таку лінію на принтері, то її довжина дійсно дорівнює 1 дюйму; на дисплеї цей дюйм дорівнює *grfx.DpiX* пікселів. Аналогічний результат можна отримати так:

grfx.PageUnit = GrapnicsUnit.Document;
grfx.PageScale = 3;

або:

grfx.PageUnit = GraphicsUnit.Point;
grfx.PageScale = 0.72;

За умовчанням для дисплея задано значення *GraphicsUnit.Pixel,* а для принтера *GraphicsUnit.Display,* для обох пристроїв значення *PageScale* дорівнює 1.

У зв'язку із запропонованими властивостями і методами на лістингі 16 наведено більш ефективний код виведення на екран і принтер лінійки довжиною 10 сантиметрів. Виведення лінійки на екран показано на рис. 18.

Лістинг 16 (exam16)

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

namespace exam16
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void DoPage(Graphics grfx)
        {
            grfx.PageUnit = GraphicsUnit.Millimeter;
            grfx.PageScale = (float)1;
            Pen pen = new Pen(ForeColor, 0.5f);
            Brush brush = new SolidBrush(ForeColor);
            const int xOff = 10;
            const int yOff = 10;
            grfx.DrawPolygon(pen, new PointF[]
        {
         new PointF(xOff, yOff),
         new PointF(xOff + 100, yOff),
         new PointF(xOff + 100, yOff + 12),
```

```
          new PointF(xOff, yOff + 12)
         }
         );

          StringFormat fmt = new StringFormat();
          fmt.Alignment = StringAlignment.Center;
          for (int i = 1; i < 100; i++)
          {
              if (i % 10 == 0) // Сантиметрові ділення
              {
                grfx.DrawLine(pen,
                new PointF(xOff + i, yOff),
                new PointF(xOff + i, yOff + 6));
                grfx.DrawString((i / 10).ToString(), Font, brush,
                     new PointF(xOff + i, yOff + 7), fmt);
              }
              else if (i % 5 == 0) // Ділення по 5 мм
              {
                  grfx.DrawLine(pen,
                  new PointF(xOff + i, yOff),
                  new PointF(xOff + i, yOff + 4));
              }

                 else // Міліметрові ділення
                 {
                     grfx.DrawLine(pen,
                     new PointF(xOff + i, yOff),
                     new PointF(xOff + i, yOff + 2));
                 }
             }
          }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
           DoPage(e.Graphics);
        }
         private void printDocument1_PrintPage(object sender,
                  System.Drawing.Printing.PrintPageEventArgs e)
         {
             DoPage(e.Graphics);
         }
         private void button1_Click(object sender, EventArgs e)
         {
             printDocument1.Print();
         }
    }
}
```

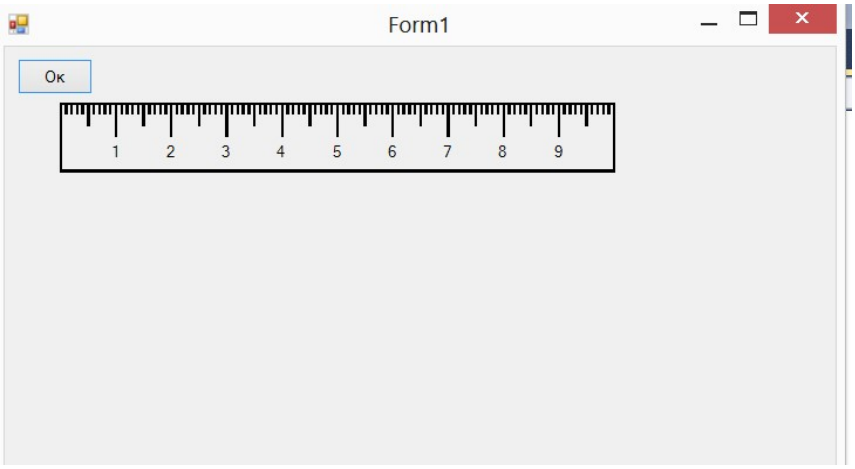


Рис. 18. До виведення лінійки з використанням міліметрів у якості одиниць вимірювання

Використання властивості *PageScale* об'єкту *Graphics* дозволяє створювати користувацьку систему координат. Скажімо, для найменшої сторони клієнтського вікна встановимо значення, рівне 1000 одиниць. При цьому побудуємо коло з діаметром, рівним 1000. Текст програми і результат рішення показані відповідно на лістингу 17. та рис. 19.

Лістинг 17. (exam17)

```
using System;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace exam17
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
        }
       private void Form1_Paint(object sender, PaintEventArgs e)
        {
          Graphics grfx = e.Graphics;
          SizeF sizef = grfx.VisibleClipBounds.Size;
          grfx.PageScale = Math.Min(
          sizef.Width  / 1000f,
          sizef.Height / 1000f);
          Text = sizef.ToString();
          grfx.DrawEllipse(new Pen(ForeColor), 0, 0, 1000f - 4,
                           1000f - 4);
        }
    }
}
```

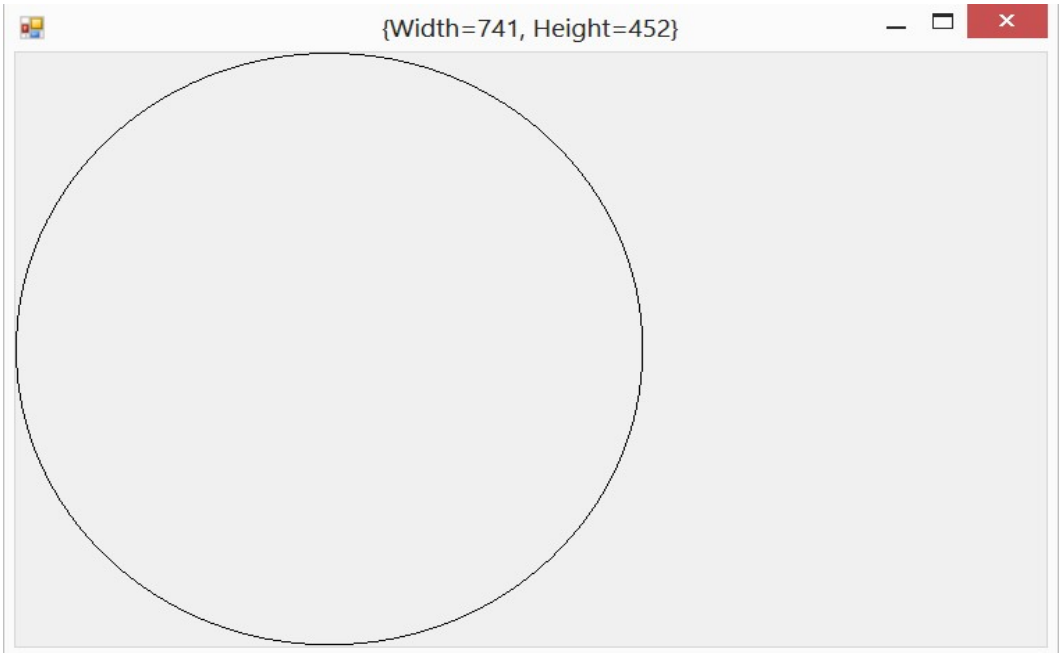


Рис. 19. До використання властивості *PageScale* объекта *Graphics*

7.2. Метричні розміри

Зміна значень властивостей *PageUnit* і *PageScale* об'єкту *Graphics* суттєво впливає на вид виведеної графіки. Іноді значення цих або іншихх властивостей об'єкту *Graphics* змінюють на час, щоб намалювати декілька фігур, після чого виникає необхідність повернутися до вихідних значень властивостей. У класі *Graphics* є два методи, *Save* і *Restore*, які слугують саме цій меті: зберігають властивості об'єкту *Graphics* і відновлюють їх згодом. Ці методи використовують клас *GraphicsState*.

У класі *GraphicsState* немає відкритих атрибутів. При виклику:

GraphicsState gs = grfx.Save();

поточні значення усіх змінюваних властивостей об'єкту *Graphics* зберігаються у об'єкті *GraphicsState*, після чого можна змінити властивості об'єкту *Graphics*. Для відновлення збережених властивостей слугує виклик:

grfx.Restore(gs);

Розміри клієнтської області форми доступні через її властивість *ClientSize* і завжди обчислюються у пікселах. Задав інше перетворення сторінки, краще виражати розміри клієнтської області не в пікселах, а в одиницях, що використовуються для методів малювання. Існує мінімум два способи отримання розмірів клієнтської області у метричних одиницях. Можливо, самий зручний – використання властивості *VisibleClipBounds* об'єкту *Graphics.* Вона завжди повертає розміри клієнтської області в одиницях, які відповідають поточним значенням властивостей *PageUnit* і *PageScale*. На лістингу 18 наводиться програма, яка за допомогою *VisibleClipBounds* показує розмір клієнтської області, виражений у різних одиницях. Реалізація програми наводиться на рис. 20.

Лістинг 18. (exam18)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace exam18
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;

        }

       private void Form1_Paint(object sender, PaintEventArgs e)
        {
          Brush brush = new SolidBrush(Color.FromArgb(255,0,0));
          int y = 2;
          Graphics grfx = e.Graphics;

             DoIt(grfx, brush, ref y, GraphicsUnit.Pixel);
             DoIt(grfx, brush, ref y, GraphicsUnit.Display);
             DoIt(grfx, brush, ref y, GraphicsUnit.Document);
             DoIt(grfx, brush, ref y, GraphicsUnit.Inch);
             DoIt(grfx, brush, ref y, GraphicsUnit.Millimeter);
             DoIt(grfx, brush, ref y, GraphicsUnit.Point);
       }
    void DoIt(Graphics grfx, Brush brush, ref int y,
                GraphicsUnit gu)
        {
            // Меняется вывод на экран, а свойства связанные с
            // геометр. параметрами только в пикселях
            GraphicsState gs = grfx.Save();
            int x = 2;

            grfx.PageUnit = gu;
            grfx.PageScale = 1;

            SizeF sizef = grfx.VisibleClipBounds.Size;
            grfx.Restore(gs);
            grfx.DrawString(gu + ": " + sizef,Font, brush,x,y);
            y += 2*Font.Height;
        }
    }
}
```

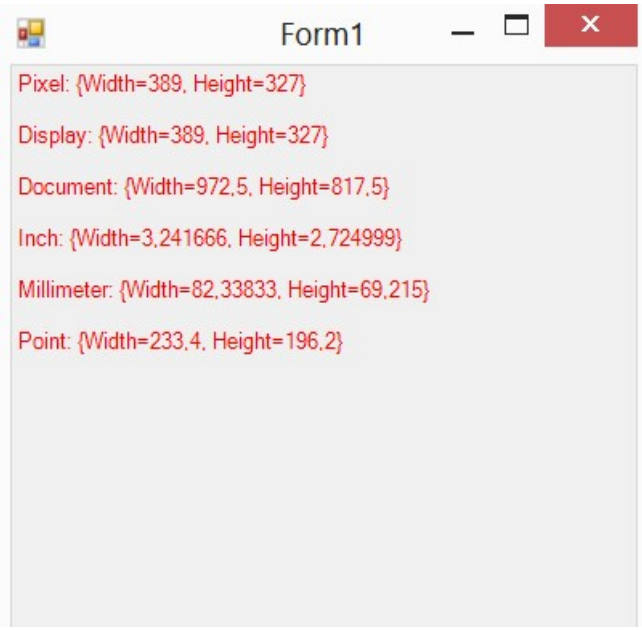


Рис. 20. До визначення розмірів клієнтської області в різних одиницях вимірювання

7.3. Друк тексту з різними одиницями вимірювання шрифту

Вище розглядався клас *Font* та його конструктори з використанням розміру шрифту в пунктах. Є можливість використовувати інші одиниці вимірювання. Наступні два конструктори *Font* мають у якості одного з аргументів *GraphicsUnit:*

Font(string strFamily, float fSize, GraphicsUnit gu)
Font(string strFamily, float fSize, FontStyle fs, GraphicsUnit gu)

Перелічення *GraphicsUnit*

Член	Значення	Опис
World		У одиницях глобальних координат
Pixel	2	У пікселах
Point	3	У 1/72 дюйма
Inch	4	У дюймах
Document	5	У 1/300 дюйма
Millimeter	6	У міліметрах

Конструктор:

new Font(strFamily, float fSize)

ідентичний конструктору:

new Font(strFamily, float fSize, GraphicsUnit.Point)

Наступні конструктори еквівалентні:

new Font(strFamily, 72)
new Font(strFamily, 72, GraphicsUnit.Point)
new Font(strFamily, 1 , GraphicsUnit.Inch)
new Font(strFamily, 25.4f, GraphicsUnit.Millimeter)
new Font(strFamily, 300, GraphicsUnit.Document)

Всі ці конструктори призводять до створення ідентичних 72-пунктних шрифтів. Це очевидно: один дюйм рівний 72 пт або 25,4 мм.

Для *GraphicsUnit.Pixel* і *GraphicsUnit.World* можна використовувати такі конструктори для створення 72-пунктних шрифтів:

new Font(strFamily, grfx.DpiY, GraphicsUnit.Pixel)
new Font(strFamily, grfx.DpiY, GraphicsUnit. World)

У якості другого аргументу задається кількість пікселів у одному дюймі по вертикалі.

Щоб створити 72-пунктний шрифт для виведення на друк з перетворенням сторінки за умовчанням, використовуючи *GraphicsUnit.Pixel* або *GraphicsUnit.World,* застосовуються конструктори:

new Font(strFamily, 100, GraphicsUnit.Pixel)
new Font(strFamily, 100, GraphicsUnit.World)

Щоб створити 24-пунктний шрифт для виведення на принтер, треба використовувати 1/3 від 100. У лістингу 19 та на рис. 21 наводяться програма і результат рішення виведення на друк одні й ті ж рядки рівної висоти з використанням різних одиниць вимірювання.

Лістинг 19. (exam19)

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

namespace exam19
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
private void printDocument1_PrintPage(object sender,
        System.Drawing.Printing.PrintPageEventArgs e)
{

  Graphics gr = e.Graphics;

  string strFont = "Times New Roman";
  Brush br = new SolidBrush(Color.FromArgb(255, 0, 0));
  float y = 0;
  Font font;

  font = new Font(strFont, 24);
  gr.DrawString("No GraphicsUnit, 24 points",
  font, br, 0, y);
  y += font.GetHeight(gr);
  font.Dispose();

  font = new Font(strFont, 24, GraphicsUnit.Point);
  gr.DrawString("GraphicsUnit.Point, 24 units",
  font, br, 0, y);
  y += font.GetHeight(gr);
  font.Dispose();

  font = new Font(strFont, 1 / 3f, GraphicsUnit.Inch);
  gr.DrawString("GraphicsUnit.Inch, 1/3 units",
  font, br, 0, y);
  y += font.GetHeight(gr);
  font.Dispose();
  font = new Font(strFont, 25.4f / 3,
                  GraphicsUnit.Millimeter);
  gr.DrawString("GraphicsUnit.Millimeter,
                 25.4/3 units",font, br, 0, y);
  y += font.GetHeight(gr);
  font.Dispose();
  font = new Font(strFont, 100, GraphicsUnit.Document);
  gr.DrawString("GraphicsUnit.Document, 100 units",
  font, br, 0, y);
  y += font.GetHeight(gr);
 font.Dispose();

 font = new Font(strFont, 100f / 3, GraphicsUnit.Pixel);
    gr.DrawString("GraphicsUnit.Pixel " + gr.DpiY / 3 +
                  " units", font, br, 0, y);
 y += font.GetHeight(gr);
 font.Dispose();
 font = new Font(strFont, 100f / 3, GraphicsUnit.World);
 gr.DrawString("GraphicsUnit.World " + gr.DpiY / 3 + "
               units",font, br, 0, y);
    y += font.GetHeight(gr);
    font.Dispose();
}
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            printDocument1.Print();
        }
}
```

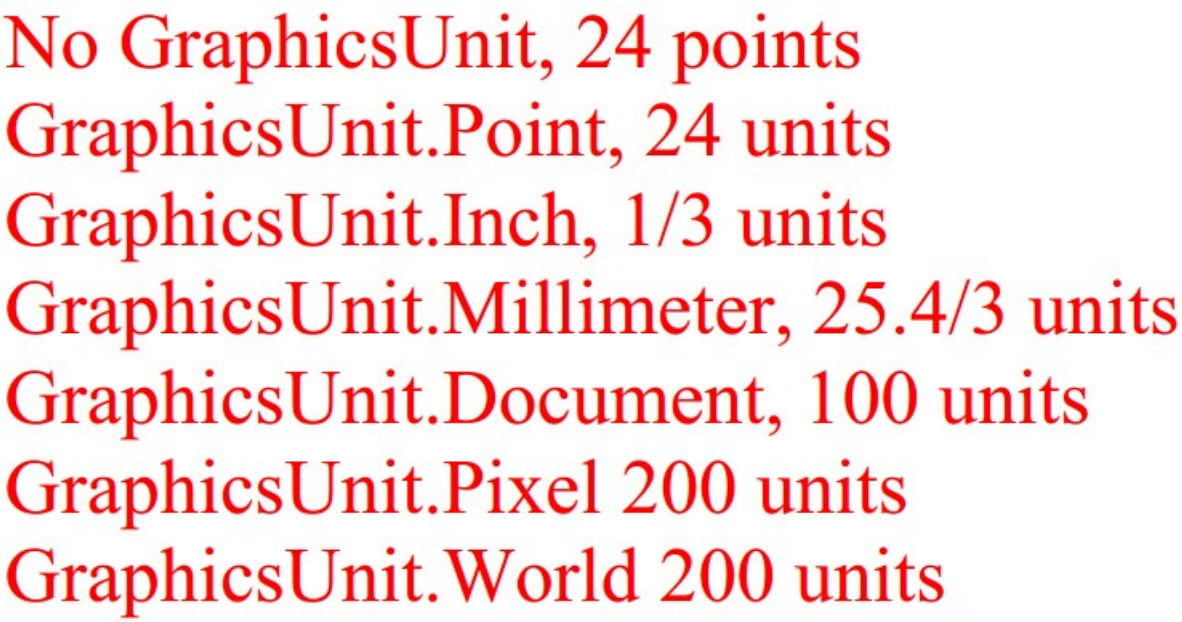


Рис. 21. Виведення тексту однакової висоти при різних одиницях вимірювання

8. Лінії, криві, згладжування та заливання областей

8.1. Лінії

Як уже зазначалося, клас *Graphics* містить багато методів для малювання графічних фігур, наприклад, ліній, кривих, прямокутників, еліпсів і растрових зображень. Першим аргументом всіх, що розглядаються в цьомц розділі методів *Draw* є об'єкт *Pen,* а першим аргументом методів *Fill* – об'єкт *Brush.*

Конструктори *Pen*:

Pen(Color color);
Pen(Color color, float fWidth);

Для малювання прямих ліній використовуються методи *DrawLine* класу *Graphics.* Є чотири перевантажені версії *DrawLine,* але всі вони вимагають один и той же набір аргументів: координати початкової та кінцевої точок лінії, а також перо, яким вона малюється:

void DrawLine(Pen pen, int x1, int y1 , int x2, int y2);
void DrawLine(Pen pen, float x1, float y1, float x2, float y2);
void DrawLine(Pen pen, Point pointl, Point point2);
void DrawLine(Pen pen, PointF pointl, PointF point2).

Для малювання ломаної (декількох з'єднаних ліній) використовується метод *DrawLines.* Його перевантажені версії

void DrawLines (Pen pen, Point[ ] apt);
void DrawLines (Pen pen, PointF[ ] aptf).

Ці методи вимагають масив цілочисельних координат *(Point)* або координат з плаваючою точкою *(PointF).*

Приклад малювання прямокутника по периметру клієнтської області програми:

```
Graphics gr = this.CreateGraphics();
Point[ ] apt = {new Point(0, 0), new Point(0, cy - 1),
new Point(cx-1, cy-1), new Point(cx - 1, 0), new Point(0, 0)};
gr. DrawLines(new Pen( clr ), apt);
```

8.2. Прямокутники

Прямокутники є найпоширенішою формою об'єктів, створених людиною. Безсумнівно, прямокутник можна намалювати за допомогою методів *DrawLine* або *DrawLines* (що ми і робили, коли окреслювали клієнтську область), але простіше зробити це методом *DrawRectangle.* У кожній з трьох його версій прямокутник визначається точкою, що задає положення верхнього лівого кута, шириною та висотою. Аналогічно визначається структура *Rectangle,* яку використовує одна із версій *DrawRectangle:*

void DrawRectangle(Pen pen, int x, int y, int ex, int cy)
void DrawRectangle(Pen pen, float x, float y, float cx, float cy)
void DrawRectangle(Pen pen, Rectangle rect)

8.3. Багатокутники

З точки зору математики, багатокутник – це замкнена фігура з трьома і більше сторонами, наприклад, трикутники, квадрати, п'яти-, шестикутники тощо. Ось два методи класу *Graphics,* що малюють багатокутники:

void DrawPolygon(Pen pen, Point[ ] point);
void DrawPolygon(Pen pen, PointF[ ] point).

По функціональності *DrawPolygon* дуже схожий на *DrawLines* за виключенням того, що він автоматично замикає контур фігури, з'єднуючи її першу та останню точки лінією. Приклад малювання прямокутника по периметру клієнтської області програми:

```
Graphics gr = this.CreateGraphics();
Point[ ] apt = {  new Point(0, 0), new Point(0, cy - 1),
new Point(cx-1, cy-1), new Point(cx - 1, 0)};
gr. DrawLines(new Pen( clr ), apt);
```

8.4. Елінси

Методи DrawEllipse класу *Graphics*:

void DrawEllipse (Pen pen, int x, int y, int ex, int cy)
void DrawEllipse (Pen pen, float x, float y, float cx, float cy)
void DrawEllipse (Pen pen, Rectangle rect)
void DrawEllipse (Pen pen, RectangleF rectF)

8.5. Дуги та сектори

Дуга (принаймні у випадку *Windows Forms*) – це сегмент еліпсу. Щоб визначити дугу, необхідно вказати ті ж відомості, що і для визначення еліпсу плюс кути для початкової та кінцевої точок дуги. В силу цих причин кожна з чотирьох версій методу *DrawArc* вимагає на два аргументи більше, ніж їх треба для *DrawEllipse*

Методи *DrawArc* класу *Graphics:*

void DrawArc (Pen pen, int x, int y, int cx, int cy, int iAngleStart, int iAngleSweep)
void DrawArc (Pen pen, float x, float y, float cx, float cy, float fAngleStart, float fAngleSweep)
void DrawArc (Pen pen, Rectangle rect, float fAngleStart, float fAngleSweep)
void DrawArc (Pen pen, RectangleF rectF, float fAngleStart, float fAngleSweep)

Останні аргументи задають кути: початок і кінець дуги (див. рис. 22).

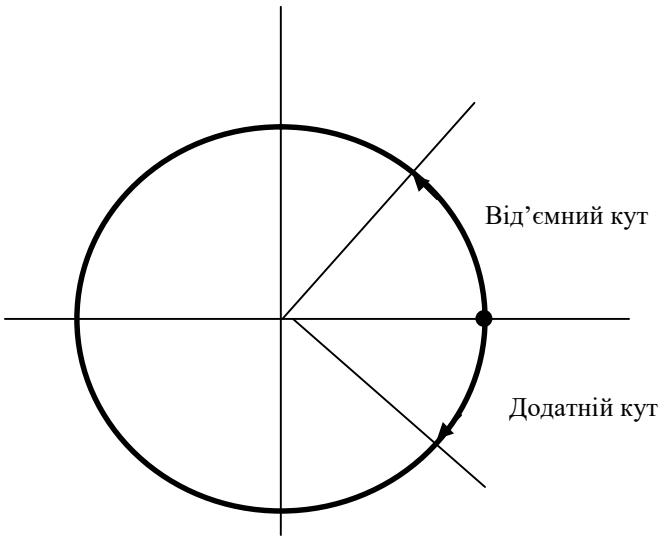


Рис. 22. До використання методу *DrawArc*

Ці кути (які можуть бути як додатними, так і від'ємними) вимірюються за часовою стрілкою, в градусах і відкладаються щодо горизонтальної вісі, що проходить від центру еліпса – праворуч.

У діловій графіці часто використовується кругова діаграма. Для її побудови можна використати методи малювання дуг и ліній (*DrawLine* і *DrawArc).* Для простоти побудови кругової діаграми запропонований метод *DrawPie*. У методів *DrawPie* ті ж аргументи, що і у *DrawArc,* але вони відріз-

няються тим, що з'єднують кінці дуги з центром еліпса лініями, створюючи замкнену область.

Методи *DrawPie* класу *Graphics:*

void DrawPie (Pen pen, int x, int y, int cx, int cy, int iAngleStart, int iAngleSweep)
void DrawPie (Pen pen, float x, float y, float cx, float cy, float fAngleStart, float fAngleSweep)
void DrawPie (Pen pen, Rectangle rect, float fAngleStart, float fAngleSweep)
void DrawPie (Pen pen, RectangleF rectF, float fAngleStart, float fAngleSweep)

8.6. Властивості класу Graphics, згладжування

У класі *Graphics* є властивості, що істотно впливають на вигляд графічних фігур:

• *PageScale* і *PageUnit* визначають одиниці розмірів при малюванні (за умовчанням одиницею розміру при малюванні на екрані є піксель);

• *Transform* – це об'єкт типу *Matrix,* що визначає матричне перетворення для всієї графічної інформації, що виводиться; перетворення транслює, масштабує, обрізує та обертає точки координат;

• *Clip* – це вирізана область; будучи встановленою, вона обмежує область виведення для будь-якої з функцій малювання, що викликаються.

Крім чотирьох перелічених вище, у класу *Graphics* є й інші властивості, вплив яких менш помітний, наприклад *SmoothingMode* і *PixelOffsetMode*. Ці властивості використовуються для згладжування.

Згладжування – це спроба приховати різкі нерівності виведеної графіки за допомогою напівтонів. Властивості класу *Graphics* для згладжування наведені в табл. 8.

Таблиця 8

Властивості класу *Graphics* для згладжування

Властивість	Доступ	Опис
SmoothingMode	Читання/запис	Згладжування ліній
PixelOffsetMode	Читання/запис	Покращене згладжування

Властивість *SmoothingMode* є переліченням та визначається і в просторі імен *System.Drawing.Drawing2D.*

Таблиця 9

Перелічення SmoothingMode

Член	Значення	Коментарі
Default	0	Без згладжування
Highspeed	1	Без згладжування
HigfoQuality	2	Згладжування включено
None	3	Без згладжування
AntiAlias	4	Згладжування включено

Згладжування можна розглянути на прикладі рис. 23.

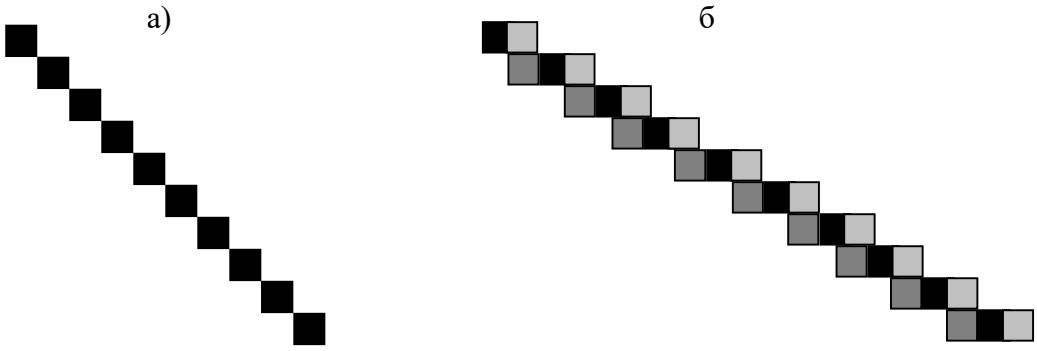


Рис. 23. Незгладжена лінія (а) та згладжена лінія (б)

Лінія 23б розпливається, якщо розглядати її поблизу, але на більш далекій відстані вона виглядає більш гладко. Можна трохи покращити згладжування, встановив для властивості *PixelOffsetMode* одне із значень перелічення *PixelOffsetMode* (див. табл. 10).

Таблиця 10

Перелічення *PixelOffsetMode*

Член	Значення	Коментарі
Default	0	Зміщення пікселів не включено
Highspeed	1	Зміщення пікселів не включено
HigfoQuality	2	Включено зміщення на половину піксела
None	3	Зміщення пікселів не включено
Half	4	Включено зміщення на половину піксела

Тут лише дві реальні можливості вибору, причому *Half* або *HighQuality* *рівнозначні*. Ефективність використання властивості *PixelOffsetMode* (*Half* або *HighQuality*) можна оцінити сприйняттям користувача.

Текст програми і результат рішення програми наводяться відповідно у лістингу 20 та рисунку 24.

Лістинг 20. (exam20)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace exam 20
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
        }

     protected override void OnPaint(PaintEventArgs e)
     {
       base.OnPaint(e);

       int cx,cy,y1,y2;
       double x;

       Graphics gl = e.Graphics;
       gl.DrawLine(new Pen(ForeColor,6.0F),200,0,214,200);

       //gl.SmoothingMode = SmoothingMode.HighQuality;
       gl.SmoothingMode = SmoothingMode.AntiAlias;
       gl.DrawLine(new Pen(ForeColor, 6.0F), 300, 0, 314, 200);

       Point[] ap = {new Point(10, 10),new Point(100, 10),
       new Point(100, 100),new Point(10, 104),new Point(10, 8)};
       gl.DrawLines(new Pen(ForeColor, 5.0F),ap);
       //График
       cx = ClientSize.Width;
       cy = ClientSize.Height;

       //for (int i = 0; i < cx - 1; i++)
       for (int i = 0; i < 10; i++)
        {
          x = Math.PI*2/cx*i;
          y1 = (int)((1 - Math.Sin(x))* cy/2);
          y2 = (int)((1 - Math.Sin(x + Math.PI*2/cx))*cy/2);
          // y2 = y1;
          gl.DrawLine(new Pen(ForeColor, 5.0F),i,y1,i+1,y2);
        }
       gl.DrawLine(new Pen(ForeColor, 5.0F), 0, cy/2,cx,cy/2);
     }
  }
}
```

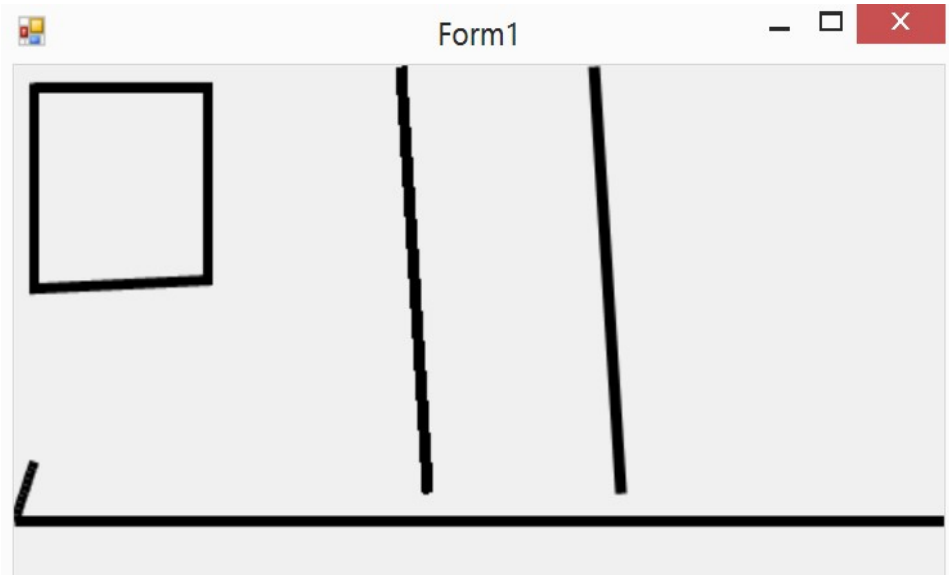


Рис. 24. До згладжування прямих

8.7. Приклади використання методів і властивостей класу Graphics

У даному розділі наводяться 4 приклади використання методів і властивостей класу *Graphics*.

Приклад 1. Використання окремих методів та властивості згладжування прямих класу *Graphics* (Лістинг 21, рис. 25).

Лістинг 21 (exam21)

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace exam 21
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
        }
       private void Form1_Paint(object sender, PaintEventArgs e)
        {
           Graphics gl = e.Graphics;
           gl.SmoothingMode = SmoothingMode.HighQuality;
           //gl.PixelOffsetMode = PixelOffsetMode.Half;
           Point[] ap = {new Point(10, 10),new Point(100, 10),
           new Point(100, 100),new Point(10, 104)};
           //gl.DrawLines(new Pen(ForeColor, 5.0F), ap);
          gl.DrawPolygon(new Pen(ForeColor, 5.0F), ap);
          gl.DrawEllipse(Pens.Red, 1, 1, ClientSize.Width - 2,
```

```
                        ClientSize.Height - 2);
          gl.DrawRectangle(new Pen(ForeColor, 5.0F),100,100,
                        100,100);
            for (int i = 0; i < 360; i+=10)
                 gl.DrawArc(new Pen(ForeColor, 1.0F),
                        new Rectangle(100,100,100,100),i,5);
        }
    }
}
```

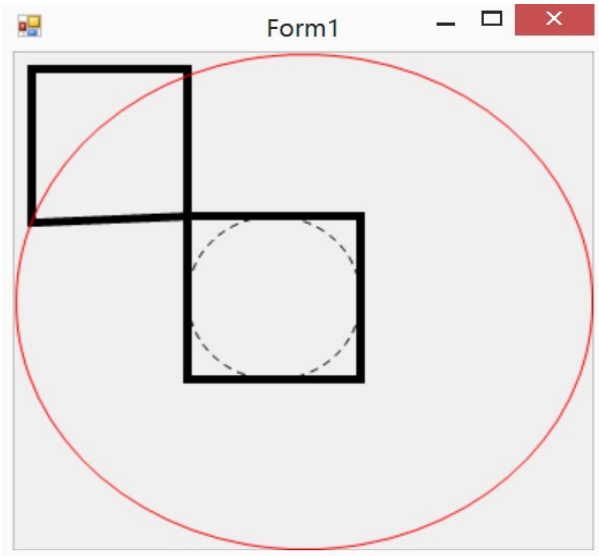


Рис. 25. Малювання та згладжування методами та властивістю класу *Graphics*

Приклад 2. Малювання прямокутників на відгук натискання кнопок. На листингу 22 та рис. 26 наводиться програма і результат рішення задачі.

Лістинг 22. (exam22)

```
namespace exam22
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
     private void button_Click(object sender, EventArgs e)
     {
       Graphics gr = CreateGraphics();
       int height = 3 * (int)gr.MeasureString("M", Font).Height;
       gr.Clear(this.BackColor);
       if (((Button)sender).Name == "button1")
       gr.DrawRectangle(new Pen(Color.Green), button1.Left,
                    height, 100, 100);
       else gr.DrawRectangle(new Pen(Color.Red), button2.Left,
                           height, 100, 100);
            gr.Dispose();
         }
    }
}
```

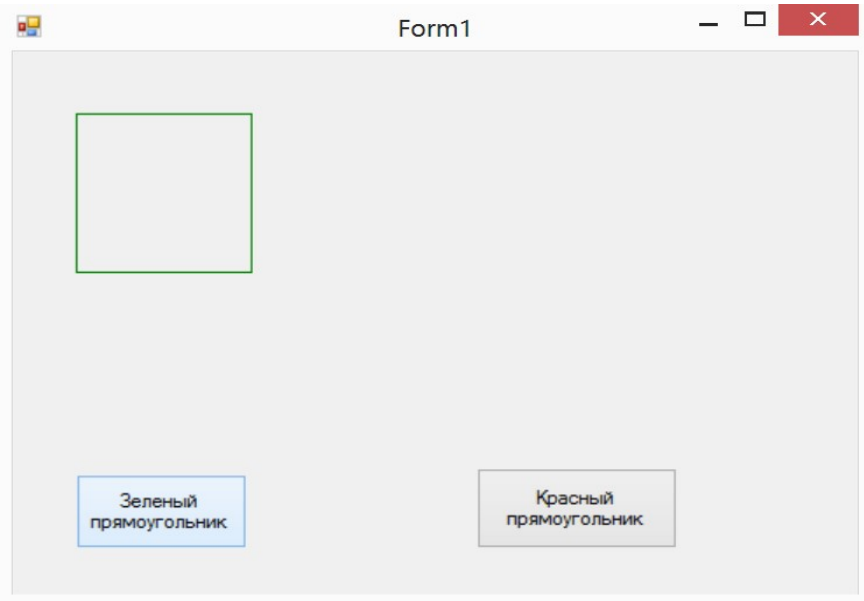


Рис. 26. Малювання прямокутників на відгук натиснення кнопок

Приклад 3. У даному прикладі розглядається побудова кругової діаграми методами малювання та заливання областей. Текст програми і результат рішення наводяться відповідно у лістингу 23 та рисунку 27.

Лістинг 23. (exam23)

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

namespace exam23
{
    public partial class Form1 : Form
    {
        Pen pen;
        float[] Values = { 50, 100, 25, 150, 100 };
        int tol_lin = 4;
        Random rd = new Random();
        Rectangle rect;
        Brush br;

        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
        }
```

```
        private void Form1_Paint(object sender, PaintEventArgs e)
         {
             float Total =  0.0F;
             float f2, f1 = 0.0F;
             foreach(float lval in Values)
             Total+=lval;
             foreach(float lval in Values)
             {
                 br = new SolidBrush(Color.FromArgb(rd.Next(255),
                 rd.Next(255), rd.Next(255)));
                 f2 = 360*lval/Total;
                 e.Graphics.FillPie(br, rect, f1, f2);
                 e.Graphics.DrawPie(pen, rect, f1, f2);
                 f1 += f2;
                 br.Dispose();
             }
         }

         private void button1_Click(object sender, EventArgs e)
         {
           Invalidate( false);
         }

         private void Form1_Load(object sender, EventArgs e)
         {
             pen = new Pen(Color.Black, tol_lin);
             button1.Location = new Point(ClientSize.Width -
             button1.ClientRectangle.Width, tol_lin);
             rect = new Rectangle(tol_lin, tol_lin,
        ClientSize.Width-2*tol_lin, ClientSize.Height-2*tol_lin);
         }
    }
}
```

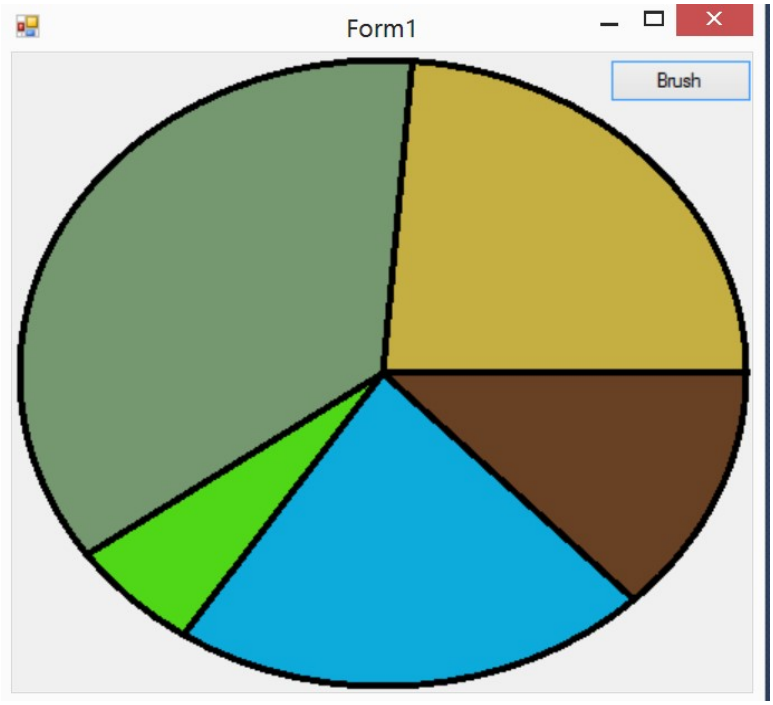


Рис. 27. Побудова кругової діаграми

Приклад 4. У даному прикладі розглядається режим заливання багатокутників. Текст програми і результат рішення наводяться відповідно у лістингу 24 та рис. 28.

Лістинг 24. (exam24)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
namespace exam24
{
    public partial class Form1 : Form
    {
        public Form1()
       {
         InitializeComponent();
         ClientSize = new System.Drawing.Size(600, 200);
         ClientSize = new Size(2 * ClientSize.Height,
         ClientSize.Height);
         this.ResizeRedraw = true;
       }

       private void Form1_Paint(object sender, PaintEventArgs e)
        {
           Graphics gr = e.Graphics;
           Point[] apt = new Point[6];
           double Ang;
           int k;
           for (int i = 0; i < 6; i++)
           {
               Ang = i*2F*Math.PI/5;
               k = i < 3 ? i * 2 : i * 2 - 5;
               apt[k] = new Point(ClientSize.Height / 2 -
               (int)(ClientSize.Height/2 * Math.Sin(Ang)) ,
               ClientSize.Height/2 -
               (int)(ClientSize.Height/2 * Math.Cos(Ang)));
            }
           gr.FillPolygon(Brushes.Red, apt, FillMode.Alternate);
           gr.DrawPolygon(new Pen(Color.Black, 2), apt);
           for (int i = 0; i < apt.Length; i++)
           apt[i].X += ClientSize.Width / 2;
           gr.FillPolygon(Brushes.Red, apt, FillMode.Winding);
           gr.DrawPolygon(new Pen(Color.Black, 2), apt);
        }
    }
}
```

Від інших областей, що зафарбовуються, багатокутники відрізняються тим, що визначальні їх лінії можуть перетинатися і накладатися одна на іншу. Це ускладнює справу, оскільки є два різні способи заливання багатокутників.

Існують чотири методи *FillPolygon*:

void FillPolygon(Brush brush, Point[] apt)
void FillPolygon(Brush brush, PointF[] apt)
void FillPolygon(Brush brush, Point[] apt, FillMode fm)
void FillPolygon(Brush brush, PointF[] apt, FillMode fm)

Вони у всьому схожі на метод *DrawPolygon*, крім одного необов'язкового аргументу. *FillMode* – це перелічення, що визначається у просторі імен *System.Drawing.Drawing2D.* У нього всього два можливих значення:

Alternate – задано за замовчуванням, чергує зафарбовані та незафарбовані області фігури.

Winding – замальовує максимально можливе число внутрішніх областей.

Класичний приклад – п'ятикутна зірка з різними *FillMode*.

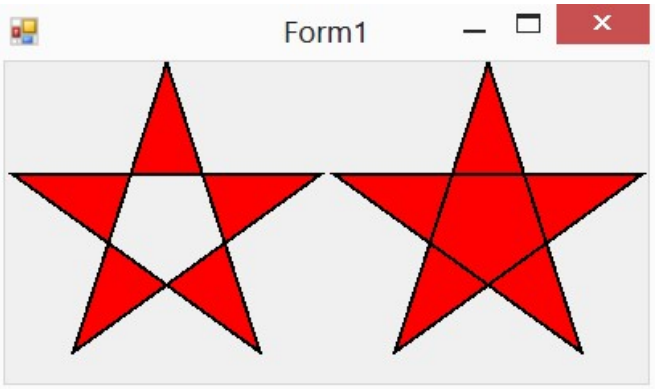


Рис. 28. Заливання багатокутників

9. Відомості про мишку

Програмі *Windows Forms* може знадобитися визначити, чи встановлена мишка і, якщо так, скільки у неї кнопок. І тут на допомогу приходить клас *SystemInformation* (див. табл. 11).

Таблиця 11

Статичні властивості SystemInformation

Тип	Властивість	Опис
bool	*MousePresent*	Вказує, чи встановлена мишка.
int	*MouseButtons*	Вказує кількість кнопок мишки.
bool	*MouseButtonsSwapped*	повертає *true,* якщо користувач змінив функції правої та лівої кнопок мишки.

Додаток *Windows Forms* отримує відомості щодо активності мишки у вигляді подій. Клас *Control* визначає 9 подій мишки і 9 відповідних захищених методів. Всі класи, породжені від *Control* (включаючи *Form),* також спа-

дкують ці 9 методів. Будь-яка подія мишки може бути отримана лише одним елементом управління. Елемент управління отримує події мишки, лише коли він активізований (увімкнений) і видимий, тобто його властивості *Enabled* і *Visible* дорівнюють *true.* Зазвичай події мишки отримує лише елемент управління, що знаходиться прямо під курсором мишки. Якщо провести курсор над дочірнім елементом управління, який активізований і видимий, то подія мишки отримає саме цей дочірній об'єкт, а не його батьківський об'єкт. Якщо дочірній елемент управління деактивізований (відключений) або невидимий, то подія мишки отримує його батьківський об'єкт. Будь-який об'єкт, який є потомком або екземпляром *Form,* отримує події мишки, лише коли курсор знаходиться над клієнтською областю форми. Об'єкт *Form* не отримує події мишки, якщо курсор знаходиться над границею, заголовком, системним меню, керуючими кнопками вікна або смугами прокрутки. В табл. 12 наведені чотири основні події мишки.

Таблиця 12

Події *Control*

Подія	Метод	Аргумент
MouseDown	*OnMouseDown*	*MouseEventArgs*
MouseUp	*OnMouseUp*	*MouseEventArgs*
MouseMove	*OnMouseMove*	*MouseEventArgs*
MouseWheel	*OnMouseWheel*	*MouseEventArgs*

Лише ці чотири події пов'язані з об'єктами типу *MouseEventArgs.* У класі *MouseEventArgs* є п'ять незмінних властивостей (табл. 13).

Таблиця 13

Властивості *MouseEventArgs*

Тип	Властивість	Опис
int	*X*	Позиція мишки по горизонталі
int	*Y*	Позиція мишки по вертикалі
MouseButtons	*Button*	Кнопка (кнопки) мишки
int	*Clicks*	У випадку подвійного клацання повертає 2
int	*Delta*	Обертання коліщатка мишки

Властивість *Button* вказує кнопку або кнопки, що беруть участь у даній події. Ця властивість не дійсна для подій *MouseWheel.* У властивість *Button* заноситься одне із значень перелічення *MouseButtons* (табл. 14).

Таблиця 14

Перелічення *MouseButtons*

Член	Значення
None	0x00000000
Left	0x0010000
Right	0x0020000
Middle	0x0040000
XButton1	0x0080000
XButton2	0x0100000

9.1. Рух мишки

Приклад руху мишки, що викликає подію *MouseMove* наведено на лістингу 25. Тут перевизначається метод *OnMouseMove,* щоб намалювати «павутинку» із ліній, з'єднуючих поточну позицію мишки з кутами і краями клієнтської області.

Лістинг 25. (exam25)

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

namespace exam25
{

    public partial class Form1 : Form
    {
        Point ptMouse = new Point(0,0);
        public Form1()
        {
            InitializeComponent();
            Text = "House Web";
            BackColor = SystemColors. Window;
            ForeColor = SystemColors.WindowText;
            ResizeRedraw = true;
        }

    private void Form1_MouseMove(object sender,MouseEventArgs e)
    {
        ptMouse.X  =  e.X;
        ptMouse.Y  =  e.Y;
        Invalidate();
    }
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
```

```
        Graphics gr = e.Graphics;
        Pen pen = new Pen(ForeColor);
        int cx = ClientSize.Width;
        int cy = ClientSize.Height;
        for (int i = 0; i < 2; i++)
          for (double j = 0; j < 1.1; j += 0.25)
            {
              gr.DrawLine(pen, ptMouse, new Point(i * cx,
                          (int)(j * cy)));
                if (j > 0.1 && j < 0.9)
                  gr.DrawLine(new Pen(Color.Red,4), ptMouse,
                  new Point((int)(j * cx), i * cy));
                  //System.Threading.Thread.Sleep(50);
            }
        Font ob = new Font("Arial", 30);
        gr.DrawString("П Р И В Е Т", ob, Brushes.Black, 20, 20);
        }
    }
}
```

На рис. 29 показано побудову павутини відносно положення мишки.

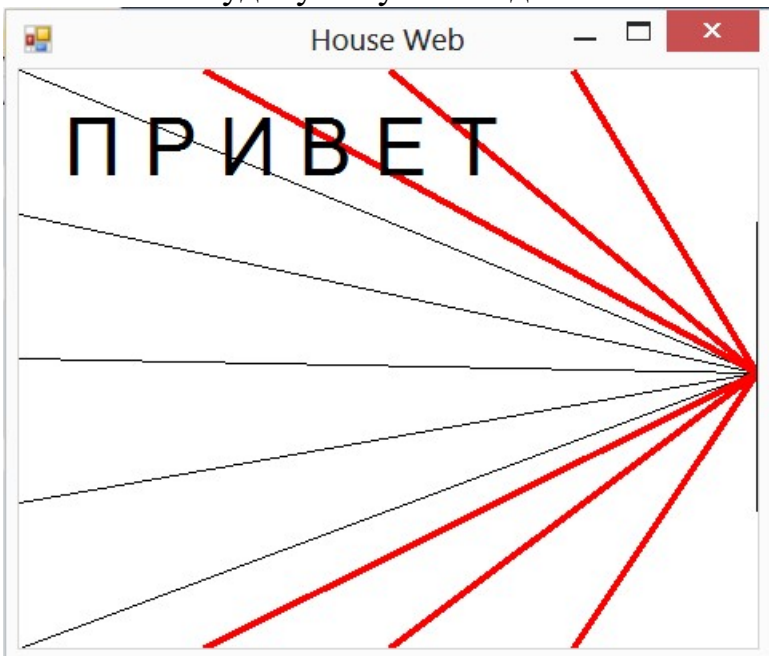


Рис. 29. Побудова павутини відносно положення мишки

9.2. Події при переміщенні курсору мишки

У таблиці 15 наведені події переміщення мишки:

Таблиця 15

Події переміщення курсору мишки

Подія	Метод	Аргумент
MouseEnter	*OnMouseEnter*	*EventArgs*
Mou.seLeave	*OnMouseLeave*	*EventArgs*
WouseHover	*QnMouseHover*	*EventArgs*

Подія *MouseEnter* вказує елементу управління (або клієнтській області форми) щодо входження курсору мишки у область над цим елементом управління. Можливо, у відповідь на цю подію повинна змінитися зовнішність елементу управління. Подія *MouseLeave* повідомляє елементу управління, що курсор мишки покинув область над цим елементом управління. Подія *MouseHover* відбувається, коли курсор зупинився, увійшовши в область над елементом управління або в клієнтську область форми. Зазвичай врна має місце між подіями *MouseEnter* і *MouseLeave.* На лістингу 26 наведена програма, що візуалізує ці три події. Після виклику *OnMouseEnter* клієнтська область зафарбовується зеленим, а після виклику *OnMouseLeave* відновлюється її звичайний колір – колір фону. У відповідь на *OnMouseHover* область фарбується у червоний колір. Результат рішення задачі наведено на рис. 30.

Лістинг 26. (exam26)

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

namespace exam 26
    public partial class Form1 : Form
    {
        bool bInside = false;

        public Form1()
        {
            InitializeComponent();
        }
        //Перехід всередину
       private void Form1_MouseEnter(object sender, EventArgs e)
       {
           Text = "MouseEnter";
           bInside = true;
           Invalidate();
       }
       //Покинув, перехід ззовні
       private void Form1_MouseLeave(object sender, EventArgs e)
        {
            Text = "MouseLeave";
            bInside = false;
            Invalidate();
        }

        // зупинка при переході
```

```
        private void Form1_MouseHover(object sender, EventArgs e)
        {
            Text = "MouseHover";
            Graphics grfx = CreateGraphics();
            grfx.Clear(Color.Red);
            Thread.Sleep(500);
            grfx.Clear(Color.Green);
            grfx.Dispose();
        }
        int d = 0;
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            d++;
            Text = d.ToString();
            e.Graphics.Clear(bInside ? Color.Green : BackColor);
        }
    }
}
```

Рис. 30. До подій при переміщенні курсору

9.3. Робота з коліщатком мишки

Програма роботи з коліщатком мишки наведена у лістингу 27.

Властивість *MouseEventArgs Delta* дійсна для подій *MouseWheel*. Якщо повернути коліщатко мишки на одне клацання вперед, властивість *Delta* стане рівною 120, а якщо на одне клацання назад, то йому буде привласнено -120. Для спрощення у методі *OnMouseWheel* ці цифри відповідно змінюються на +1 і -1.

У вихідній рядковій змінній, впроваджені символи переведення рядку. Програма підраховує кількість рядків під час виконання конструктору форми і зберігає результат у полі *TextLines*. Конструктор також отримує значення міжрядкового інтервалу, викликаючи метод *Font.GetHeight(gr)*. Значення, що повертається методом, заноситься у поле *cyText*. Завершальна частина ініціа-

лізації полягає у виконанні методу *OnResize.* Перший раз конструктор повинен викликати цей метод явно, потім *OnResize* будт викликаться кожного разу, коли користувач змінить розміри форми. *OnResize* використовує значення *cyText* для розрахунку значення *Clientlines,* що визначає кількість рядків, яке може вмістити клієнтська область. Змінна *StartLine* – це рядок тексту, який повинен з'явитися у верхній частині клієнтської області. При ініціалізації цій змінній привласнюється нульове значення. Змінна *StartLine* визначається як при зміні клієнтського вікна, так і при обертанні коліщатка мишки. Очевидно, що прокручивати текст можна до тих пір, поки останній рядок у нижнього краю клієнтської області не буде останнім рядком у вихідному тексті. Крім того, значення *StartLine* не повинно бути від'ємним числом. Ці обмеження враховані при використанні методів *Math.Min* и *Math.Max*. Очевидно, що значення *StartLine* не повинно бути більше різниці *Textlines – Clientlines*. На рис. 31 наводиться результат рішення задачі.

Лістинг 27. (exam27)

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

namespace exam27
{
    public partial class Form1 : Form
    {

    int Textlines = 0; // кількість рядків у документі
    int Clientlines ; // кількість рядків у клієнтнській області
    // Початкова лінія виведення в клієнтській області
    int StartLine = 0;
    float cyText;      // Висота шрифту
    string str;

    public Form1()
    {
        InitializeComponent();
        Text = "Annabel Lee by Edgar Allan Poe";
        this.BackColor = SystemColors.Window;
        this.ForeColor = SystemColors.WindowText;
        this.ResizeRedraw = true; //Відгук для усіх сторін вікна
        str = "00001\n" +
              "111\n" +
              "2222\n" +
```

```
"33333\n" +
"44444\n" +
"55555\n" +
"66666\n" +
"77777\n" +
"88888\n" +
"99999\n" +
"00002\n" +
"11111\n" +
"22222\n" +
"33333\n" +
"44444\n" +
"55555\n" +
"66666\n" +
"77777\n" +
"88888\n" +
"99999\n" +
"00003\n" +
"11111\n" +
"22222\n" +
"33333\n" +
"44444\n" +
"55555\n" +
"66666\n" +
"77777\n" +
"88888\n" +
"99999\n" +
"00004\n" +
"11111\n" +
"22222\n" +
"33333\n" +
"44444\n" +
"55555\n" +
"66666\n" +
"77777\n" +
"88888\n" +
"99999\n" +
"00005\n" +
"11111\n" +
"22222\n" +
"33333\n" +
"44444\n" +
"55555\n" +
"66666\n" +
"77777\n" +
"88888\n" +
"99999\n" +
"00006\n" +
"11111\n" +
"22222\n" +
```

```
                "33333\n" +
                "44444\n" +
                "55555\n" +
                "66666\n" +
                "77777\n" +
                "88888\n" +
                "99999\n";
            int index = -1;
            while ((index = str.IndexOf('\n', index + 1)) != -1)
            Textlines++;
            Graphics gr = CreateGraphics();
            cyText = Font.GetHeight(gr);
            gr.Dispose();

           //виклик відгуку Resize
            Form1_Resize(this, EventArgs.Empty);
        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            Clientlines = (int)(ClientSize.Height / cyText);
            StartLine = Math.Max(0,Math.Min(StartLine,
                                 Textlines - Clientlines));
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
          Graphics grfx = e.Graphics;
          grfx.DrawString(str, Font, new SolidBrush(ForeColor),
                          0, -StartLine* cyText);
        }

        protected override void OnMouseWheel(MouseEventArgs e)
        {
            base.OnMouseWheel(e);
            int Scroll = e.Delta / Math.Abs(e.Delta);
            Text = e.Delta.ToString();
            StartLine -= Scroll;
            StartLine =   Math.Max(0,Math.Min(StartLine,
                                    Textlines - Clientlines));
            Text = (Textlines - Clientlines).ToString();
            Invalidate();
        }
    }
}
```

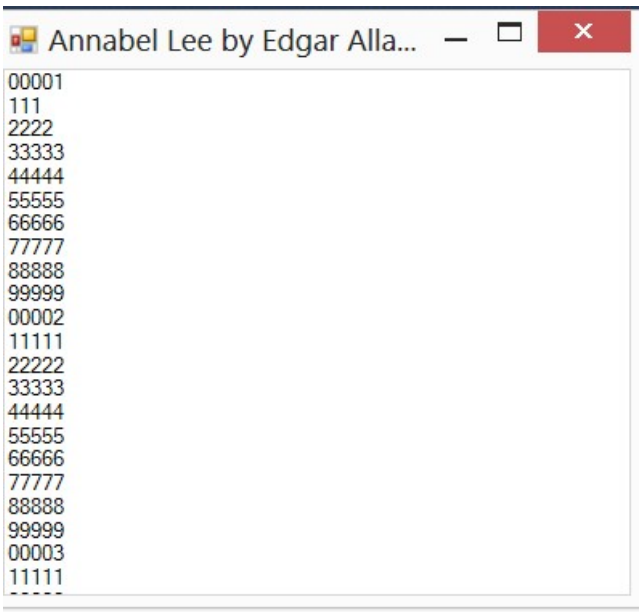


Рис. 31. До роботи з коліщатком мишки

10. Робота з гумовим контуром

У 2D-графіці часто доводиться виділяти прямокутну область для виділення показників, частини рисунку. Зазвичай це виконується "гумовим" контуром, тобто при переміщенні мишки і натиснутій лівої кнопки будується новий прямокутник та видаляється старий.

Ефективно використовувати в *OnMouseMove* методику *малювання за принципом "виключаюче АБО"* (*exclusive-OR* або *XOR*), *XOR*-малювання не просто малює кольорові пікселі на екрані, а змінює кольори існуючих пікселів. Лінія, намальована в такий спосіб на чорному фоні, виглядає білою, а на голубому фоні – червоною. Перевага цієї методики у тому, що друга *XOR*-лінія, задана тими ж координатами, стирає повністю першу. Тобто повністю відновлює колір пікселів, які були до малювання першої лінії. У старих версіях *C# GDI+* не підтримується *XOR*-малювання і для стирання попереднього прямокутника використовується колір фону. Очевидно, що це не кращий спосіб. З’являються сліди фонових ліній, від яких треба позбутися. Для усунення цього обробка *OnMouseMove* закінчується викликом *Invalidate,* генеруючим подію *Paint.* У даному випадку повністю перемальовується малюнок. У випадку *XOR*-малювання необхідність у виклику *Invalidate* просто відпала б. Нижче наведені три приклади роботи з гумовим контуром. Перший і другий приклад виконані без використання *XOR*-малювання. У третьому прикладі програма виконана з використанням *XOR*-малювання.

Приклад 1.

У лістингу 28 та на рис. 32 відповідно наведені текст програми і результат її рішення без використання *XOR*-малювання. Детальний опис програми недоцільно, оскільки студенти мають досвід програмування резинового контуру при вивченні *Visual C++*.

Лістинг 28. (exam28)

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

namespace exam28
{
    public partial class Form1 : Form
    {
        Point ptbeg,ptend;
        Graphics grfx;

        public Form1()
        {
         InitializeComponent();
         Text = "Blockout flectangle with House";
         BackColor = SystemColors.Window;
         ForeColor = SystemColors.WindowText;
         ptbeg = ptend = Point.Empty;
         grfx = CreateGraphics();
        }

    private void Form1_MouseDown(object sender,MouseEventArgs e)
    {
      ptbeg = ptend = e.Location;
    }

   private void Form1_MouseMove(object sender, MouseEventArgs e)
   {
     if (e.Button != MouseButtons.Left) return;
     for (int i = 0; i < 2; i++)
      {
        grfx.DrawRectangle(new Pen((i == 0) ?
        BackColor : ForeColor),
        ptbeg.X, ptbeg.Y, ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
        ptend = e.Location;
       }
   }

    private void Form1_MouseUp(object sender, MouseEventArgs e)
      {
        if (e.Button == MouseButtons.Left) Invalidate();
      }

    protected override void  OnPaint(PaintEventArgs e)
    {
        //return;
```

```
    if (ptbeg.Equals(Point.Empty)) return;
    if (Math.Abs(ptbeg.X - ptend.X) < 2 ||
            Math.Abs(ptbeg.Y - ptend.Y) < 2) return;
    e.Graphics.DrawRectangle(new Pen(ForeColor),ptbeg.X,ptbeg.Y,
                         ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
    }
  }
}
```

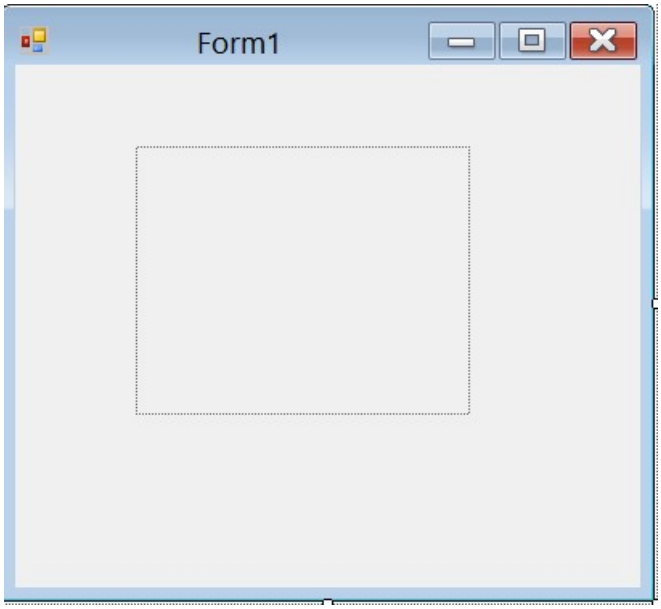


Рис. 32. До формування гумового контуру без використання *XOR*

Приклад 2.

У прикладі використаний інтерфейс, який використовується для двох класів відповідно в *Form*1 та *Form*2 (лістинги 29а і 29б). У першій формі виводиться гумовий прямокутник. У другій формі виводиться еліпс у вигляді гумового контуру. Інтерфейс схожий на визначення класу. Він містить методи, властивості і індексатор. Але інтерфейс містить лише сигнатури (*signatures*) цих членів, але не їх тіла. Як відомо, клас може бути спадкоємцем іншого класу, а клас, який не виглядає нічиїм спадкоємцем, насправді є спадкоємцем *Object.*

Лістинг 29а. (exam29)

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

namespace exam29
{
```

```
interface ICapture
{
    void OnLostCapture();
    void Down(Point a);
    void Move(Point a);
}
public partial class Form1 : Form, ICapture
{
    public Point ptbeg, ptend;
    Form2 f = null;

    public Form1()
    {
        InitializeComponent();
        Text = "Blockout";
        BackColor = SystemColors.Window;
        ForeColor = SystemColors.WindowText;
        ptbeg = ptend = Point.Empty;
        CaptureLoss win = new CaptureLoss();
        win.control = this;
        win.AssignHandle(Handle);
    }

   private void Form1_Paint(object sender, PaintEventArgs e)
   {
       if (Math.Abs(ptbeg.X - ptend.X) < 5 ||
            Math.Abs(ptbeg.Y - ptend.Y) < 5)
       ptbeg.X = ptend.X = ptbeg.Y = ptend.Y;
       e.Graphics.DrawRectangle(new Pen(ForeColor), ptbeg.X,
          ptbeg.Y, ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
    }

public void OnLostCapture()
    {
       // Text ="OnLostCapture";
        vr++;
        Text = vr.ToString() + "C  ";

        Invalidate();
    }
    public void Down(Point a)
    {
        ptbeg = ptend = a;
    }

    int vr = 0;

   public void Move(Point a)
   {
     vr++;
     Text = vr.ToString() + "M  ";
    Graphics grfx = CreateGraphics();
    grfx.DrawRectangle(new Pen(BackColor), ptbeg.X, ptbeg.Y,
```

```
            ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
        ptend = a;
        grfx.DrawRectangle(new Pen(ForeColor), ptbeg.X, ptbeg.Y,
            ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
        grfx.Dispose();
       }

     private void form2ToolStripMenuItem_Click(object sender,
                                                EventArgs e)
        {
            if (f == null)
            {
                f = new Form2();
                f.Show();
            }
            else
            {
                f.Close();
                f = null;
            }
        }
    }

class CaptureLoss:NativeWindow
{
    public ICapture control;

    protected override void  WndProc(ref Message m)
{
    // WM_LBUTTONDOWN - 513
     if (m.Msg == 513)
    {
        short a = (short)m.LParam;
        short b = (short)(((int)m.LParam)/65536);
        Point pp = new Point(a, b);
        control.Down(pp);
    }
      // WM_MOUSEMOVE - 512
     if (m.Msg == 512&&((int)m.WParam)==1)
     {
         short a = (short)m.LParam;
         short b = (short)(((int)m.LParam) / 65536);
         Point pp = new Point(a, b);
         control.Move(pp);
     }

     // WM_CAPTURECHANGED - 533
     if (m.Msg == 533)
     control.OnLostCapture();
     base.WndProc(ref m);
      }
    }
 }
```

Лістинг 29б (exam29)

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

namespace exam29
{
    public partial class Form2 : Form, ICapture
    {
        public Point ptbeg, ptend;

        public Form2()
        {
            InitializeComponent();
            Text = "Blockout 2";
            BackColor = SystemColors.Window;
            ForeColor = SystemColors.WindowText;
            ptbeg = ptend = Point.Empty;
            CaptureLoss win = new CaptureLoss();
            win.control = this;
            win.AssignHandle(Handle);
        }

       private void Form2_Paint(object sender, PaintEventArgs e)
       {
           if (Math.Abs(ptbeg.X - ptend.X) < 5 ||
                Math.Abs(ptbeg.Y - ptend.Y) < 5)
           ptbeg.X = ptend.X = ptbeg.Y = ptend.Y;
           e.Graphics.DrawEllipse(new Pen(ForeColor), ptbeg.X,
             ptbeg.Y, ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
        }

        public void OnLostCapture()
        {
            Invalidate();
        }

        public void Down(Point a)
        {
            ptbeg = ptend = a;
        }
```

```
     public void Move(Point a)
      {
         Graphics grfx = CreateGraphics();
         grfx.DrawEllipse(new Pen(BackColor), ptbeg.X, ptbeg.Y,
             ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
         ptend = a;
         grfx.DrawEllipse(new Pen(ForeColor), ptbeg.X, ptbeg.Y,
             ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
         grfx.Dispose();
      }
    }
}
```

Клас також може бути спадкоємцем одного або декількох інтерфейсів. Якщо клас спадкує інтерфейс, у ньому повинні бути реалізовані всі методи і властивості, визначені у інтерфейсі. Інтерфейси допомагають підвищувати ступінь абстрактності класів, так як для визначення змінної замість імені класу або структури може бути використано ім'я відповідного інтерфейсу, після чого клас може викликати методи і властивості, визначені в цьому інтерфейсі. У нашому випадку, використовуючи інтерфейс, для кожної з двох форм підключаються різні функції:

void OnLostCapture();
void Down(Point a);
void Move(Point a);

Функція *void OnLostCapture*() підключається при генерації відгуку WM_CAPTURECHANGED (код = 533). Це відбувається коли вікно втрачає захоплення мишки штатним (при відпусканні кнопки мишки) або нештатним чином. Для цього знадобиться клас *NativeWindow*. Дві інші функції підключаються на відгук натискання і переміщення мишки.

Ці функції підключаються по відгукам у функції *WndProc* класу *NativeWindow*.

Прив'язка двох форм до єдиної функції *WndProc* виконується у кожній формі з використанням операторів

CaptureLoss win = new CaptureLoss();
win.control = this;
win.AssignHandle(Handle);

Тут виконується створення об'єкту класу *CaptureLoss*(), похідного від класу *NativeWindow*. Потім інтерфейсу (як базовому класу) привласнюється об'єкт класу *Form*1 або *Form*2. У функції *AssignHandle* здійснюється прив'язка через дескриптор вікна об'єкту класу *CaptureLoss*() до *Form*1 або *Form*2. На рис. 33. наведено результат рішення задачі.

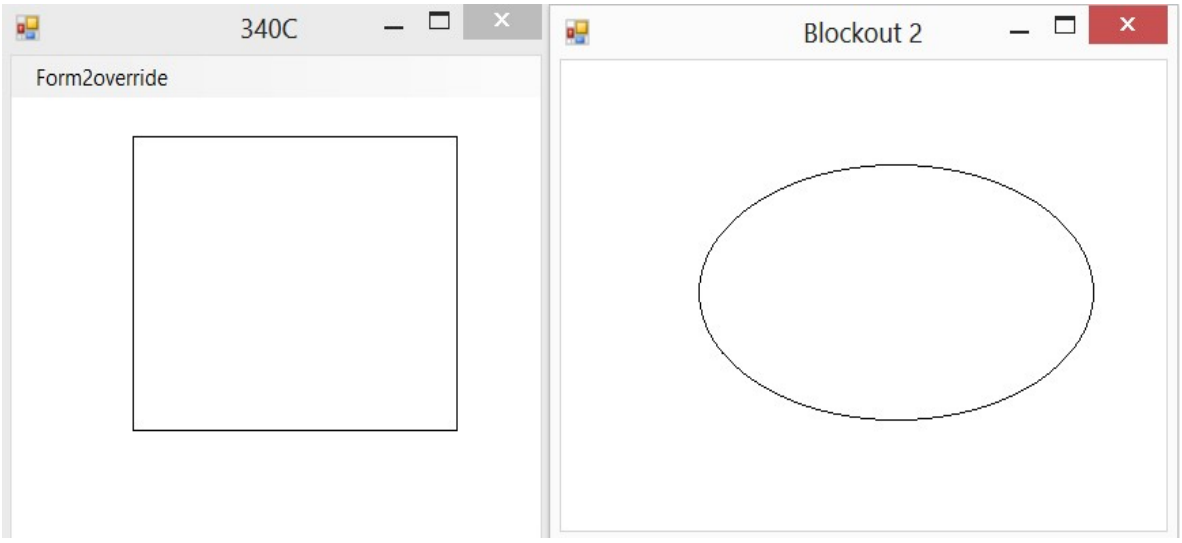


Рис. 33. Використання інтерфейсу при малюванні гумових контурів

Приклад 3. Цей приклад зроблено на основі *прикладу 2*. Однак є дві суттєві відмінності. По-перше, при виході мишки за межі клієнтського вікна будується прямокутник, утворений перетином клієнтського вікна і прямокутником, утворений мишкою. При цьому припиняється захоплення мишки. По-друге, у програмі малювання виконується з використанням *XOR*-малювання. Як вже зазначалося перевага цієї методики у тому, що друга *XOR*-лінія, задана тими ж координатами що і перша, стирає повністю першу лінію. У прикладі використаний інтерфейс, який використовується для двох класів відповідно до *Form*1 та *Form*2 (лістинги 30а і 30б). Результат рішення наведено на рис. 34.

Лістинг 30а (exam30)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace exam30
{
  interface ICapture
    {
        void OnLostCapture();
        void Down(Point a);
        void Move(Point a);
    }

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

```
 public Point ptbeg, ptend;
 Form2 f = null;
 public Form1()
 {
     InitializeComponent();
     Text = "Blockout";
     BackColor = SystemColors.Window;
     ForeColor = SystemColors.WindowText;
     ptbeg = ptend = Point.Empty;
     CaptureLoss win = new CaptureLoss();
     win.control = this;
     win.AssignHandle(this.Handle);
 }

private void Form1_Paint(object sender, PaintEventArgs e)
{
  Graphics gr = CreateGraphics();
  gr.DrawRectangle(new Pen(ForeColor), ptbeg.X, ptbeg.Y,
  ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
  gr.Dispose();
}

 public void OnLostCapture()
 {
    Invalidate();
 }

 public void Down(Point a)
 {
     ptbeg = ptend = a;
 }

 public void Move(Point a)
 {
     Rectangle rect;
     Point pt1, pt2;
     pt1 = PointToScreen(ptbeg);
     Peres(ref pt1);
     pt2 = PointToScreen(ptend);
     rect = new Rectangle(pt1.X, pt1.Y, pt2.X - pt1.X,
                          pt2.Y - pt1.Y);
     ControlPaint.DrawReversibleFrame(rect,
         Color.FromArgb(255, 255, 0), FrameStyle.Dashed);
     ptend = a;
     Peres(ref ptend);
     pt2 = PointToScreen(ptend);
     rect = new Rectangle(pt1.X, pt1.Y, pt2.X - pt1.X,
                          pt2.Y - pt1.Y);
     ControlPaint.DrawReversibleFrame(rect,
         Color.FromArgb(255, 255, 0), FrameStyle.Dashed);
 }
```

```
     private void Peres(ref Point pt)
     {
      pt.X = (pt.X < ClientRectangle.Left) ?
              ClientRectangle.Left : pt.X;
      pt.X = (pt.X > ClientRectangle.Right) ?
              ClientRectangle.Right : pt.X;
      pt.Y = (pt.Y < ClientRectangle.Top + menuStrip1.Height) ?
              ClientRectangle.Top + menuStrip1.Height : pt.Y;
      pt.Y = (pt.Y > ClientRectangle.Bottom) ?
              ClientRectangle.Bottom : pt.Y;
        }

     private void form2ToolStripMenuItem_Click(object sender,
                                               EventArgs e)
        {
            if (f == null)
            {
                f = new Form2();
                f.Show();
            }
            else
            {
                f.Close();
                f = null;
            }
        }
    }

class CaptureLoss : NativeWindow
{
  public ICapture control;

  protected override void  WndProc(ref Message m)
   {
    // WM_LBUTTONDOWN - 513
     if (m.Msg == 513)
    {
        short a = (short)m.LParam;
        short b = (short)(((int)m.LParam)/65536);
        Point pp = new Point(a, b);
        control.Down(pp);
    }

     // WM_MOUSEMOVE - 512
     if (m.Msg == 512&&((int)m.WParam)==1)
     {
         short a = (short)m.LParam;
         short b = (short)(((int)m.LParam) / 65536);
         Point pp = new Point(a, b);
         control.Move(pp);
     }
```

```
          // WM_CAPTURECHANGED - 533

          if (m.Msg == 533)
          control.OnLostCapture();
          base.WndProc(ref m);
    }
   }
}
```

Лістинг 30б (exam30)

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

namespace exam30
{
    public partial class Form2 : Form, ICapture
    {
        public Point ptbeg, ptend;

        public Form2()
        {
            InitializeComponent();
            Text = "Blockout 2";
            BackColor = SystemColors.Window;
            ForeColor = SystemColors.WindowText;
            ptbeg = ptend = Point.Empty;
            CaptureLoss win = new CaptureLoss();
            win.control = this;
            win.AssignHandle(Handle);
        }

       private void Form2_Paint(object sender, PaintEventArgs e)
       {
           if (Math.Abs(ptbeg.X - ptend.X) < 5 ||
               Math.Abs(ptbeg.Y - ptend.Y) < 5)
           ptbeg.X = ptend.X = ptbeg.Y = ptend.Y;
           e.Graphics.DrawEllipse(new Pen(ForeColor), ptbeg.X,
               ptbeg.Y,ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
        }

        public void OnLostCapture()
        {
          Invalidate();
        }
```

```
        public void Down(Point a)
        {
            ptbeg = ptend = a;
        }

      public void Move(Point a)
      {
          Graphics grfx = CreateGraphics();
          grfx.DrawEllipse(new Pen(BackColor), ptbeg.X, ptbeg.Y,
          ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
          ptend = a;
          grfx.DrawEllipse(new Pen(ForeColor), ptbeg.X, ptbeg.Y,
          ptend.X - ptbeg.X, ptend.Y - ptbeg.Y);
          grfx.Dispose();
      }
  }
}
```

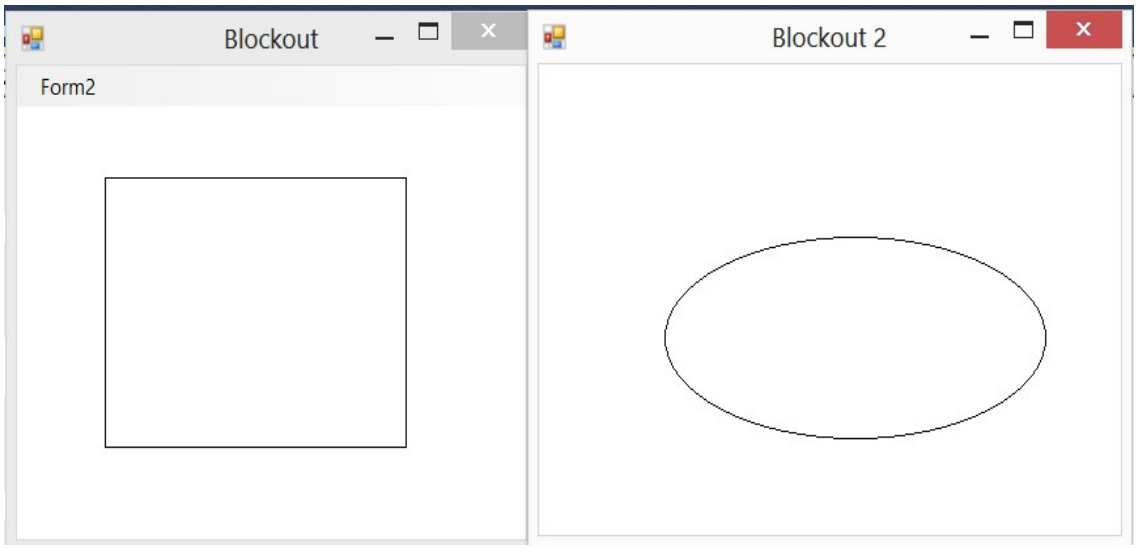


Рис. 34. До *XOR*-малювання гумових контурів

У програмі перетин прямокутників виконується функцією *Peres*(). *XOR*-малювання виконується при використанні статичного методу класу *ControlPaint*:

ControlPaint.DrawReversibleFrame().

Приклад використання цієї функції у програмі:

ControlPaint.DrawReversibleFrame(rect, Color.FromArgb(255, 255, 0), FrameStyle.Dashed).

11. Робота з точковою 2D-графікою

Робота з двовимірною графікою докладно розглядалась при вивчнні *Visual C++* в посібнику "Інструментальні засоби прикладного програмування з використанням мови *Visual C++* (Частина I)" у розділі "Універсальний приклад роботи з двовимірною графікою з використанням гумового контуру" [1]. Такий же алгоритм використано при програмуванні на мові C#. Можливості програми полягають у наступному. На екран дисплея виводяться точки, що характеризуються координатами *X*, *Y* та показником (показниками). При цьому за допомогою гумового контуру можна деталізувати рисунок, правою кнопкою мишки переміщувати рисунок в різних напрямах, змінювати колір і шрифт виведених показників, відносно яких масштабується графік. На лістингу 31 показаний текст програми, фрагмент рішення задачі показаний на рис. 35.

Лістинг 31 (exam31)

```
//Для визначення правильної довжини рядку у MeasureString –
//StringFormat.GenericTypographic.
//SizeF pr_po =
//gr.MeasureString("9", font_os, PointF.Empty,
//StrinFormat.GenericTypographic);

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

namespace exam31
{
    public partial class Form1 : Form
    {
        Point PointBegin = Point.Empty;
        Point PointEnd   = Point.Empty;
        int KOL = 100;
        int KDX = 10;
        int KDY = 6;
        double CD = 1.0; // будь-яке додатне число
        double N = 2; // кратність ціни поділу
        double[] X;//
        double[] Y;//  координати вихідних точок
        double[] Z;//
        double Xmin, Xmax, Ymin, Ymax;     // поточні межі
        // межі вихідних даних
        double Xmin_d, Xmax_d, Ymin_d, Ymax_d;
        int ots_lev1, ots_lev2, ots_pra1, ots_pra2;
```

```
int ots_ver1, ots_ver2, ots_niz1, ots_niz2;
double K_e_px, K_e_py;
bool flag = false; // для коліщатка
Font font_os;
Font font_nad;
Font font_pok;
StringFormat sf;

public Form1()
{
    InitializeComponent();
    DoubleBuffered = true; // подвійна буферизація
    Cursor = Cursors.Arrow; // курсор із стрілкою
    sf = new StringFormat();
    ResizeRedraw = true;
    X = new double[KOL];
    Y = new double[KOL];
    Z = new double[KOL];
    font_os = new Font("Arial",  14);
    font_nad = new Font("Arial", 20);
    font_pok = new Font("Arial", 10);

    // Размеры экрана
    Size ek = SystemInformation.PrimaryMonitorSize;
    Size = new Size((int)(0.8*ek.Width),
                    (int)(0.8*ek.Height));
    Form_mas();
    Nastroyka(); // Налаштування
    Max_Min();    //Max-Min
    Top =  (int)(0.1 * ek.Height);
    Left = (int)(0.1 * ek.Width);
}
//////////////////////////////////////
void Form_mas()
{
    //Формування масиву показників
    Random rnd = new Random();
    int[,] XY = new int[KOL + 1, KOL + 1];
    int xx, yy;
    int kol = 0;
    for (int i = 0; i < KOL + 1; i++)
    for (int j = 0; j < KOL + 1; j++)
    XY[i,j] = 0;

      // формування точок з різними координатами
      while (true)
        {
            xx = rnd.Next(KOL) + 1;
            yy = rnd.Next(KOL) + 1;
            if (XY[xx,yy] == 1) continue;
            XY[xx, yy] = 1;
            X[kol] = xx;
            Y[kol] = yy;
```

```
            Z[kol] = rnd.Next(100) + 1;
            if (kol == KOL - 1) break;
            kol++;
        }
    }

 void Nastroyka()
 {
     Graphics gr = CreateGraphics();
     //Для визначення правильної довжини рядку у
     //MeasureString - StringFormat.GenericTypographic

     SizeF pr_po_os =  gr.MeasureString("9", font_os,
     PointF.Empty, StringFormat.GenericTypographic);
     SizeF pr_po_nad = gr.MeasureString("9", font_nad,
     PointF.Empty, StringFormat.GenericTypographic);
     SizeF pr_po_pok = gr.MeasureString("9", font_pok,
     PointF.Empty, StringFormat.GenericTypographic);

     ots_lev1 = (int)(4 * pr_po_os.Width);
     ots_lev2 = (int)(2 * pr_po_pok.Width);
     ots_pra1 = (int)(2 * pr_po_pok.Width);
     ots_pra2 = (int)(2 * pr_po_pok.Width);
     ots_ver1 = (int)(2 * pr_po_nad.Height) +
                      menuStrip1.Height;
     ots_ver2 = (int)(2 * pr_po_pok.Width);
     ots_niz1 = (int)(2 * pr_po_os.Height);
     ots_niz2 = (int)(0.5 * pr_po_os.Height);
     gr.Dispose();
 }

 void Max_Min()
 {
     Xmax = Xmin = X[0];
     Ymax = Ymin = Y[0];
     for (int i = 1; i < KOL; i++)
     {
         if (X[i] > Xmax) Xmax = X[i];
         if (X[i] < Xmin) Xmin = X[i];
         if (Y[i] > Ymax) Ymax = Y[i];
         if (Y[i] < Ymin) Ymin = Y[i];
     }
     Xmax = Xmax_d = Math.Ceiling(Xmax);
     Ymax = Ymax_d = Math.Ceiling(Ymax);
     Xmin = Xmin_d = Math.Floor(Xmin);
     Ymin = Ymin_d = Math.Floor(Ymin);
 }

private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics gr = e.Graphics;
    int i, koor_x, koor_y;
    double pr;
```

```
            string buf;

            if (flag == false)
            {
                Xmin = Math.Floor(Xmin / N) * N;
                Ymin = Math.Floor(Ymin / N) * N;
            }
             if (CD >= 0)
             {
                 pr = CD = 0;
                 while ((Xmin + CD * KDX) < Xmax ||
                        (Ymin + CD * KDY) < Ymax)
                 {
                     pr++;
                     CD = N * pr;
                 }
                 if (CD == 0) CD = N;
             }
             else
                 CD *= -1.0;

             Xmax = Xmin + KDX * CD;
             Ymax = Ymin + KDY * CD;

             K_e_px = (Xmax - Xmin) /
(ClientSize.Width  - ots_lev1 - ots_lev2 - ots_pra1 - ots_pra2);
             K_e_py = (Ymax - Ymin) /
(ClientSize.Height - ots_niz1 - ots_niz2 - ots_ver1 - ots_ver2);

              // Виведення заголовку
        sf.Alignment = sf.LineAlignment = StringAlignment.Center;
        gr.DrawString("Работа с двумерной графикой", font_nad,
              Brushes.Black,ClientSize.Width / 2,
              (menuStrip1.Height + ots_ver1) / 2, sf);

        //   Виведення осей координат
        Pen pen = new Pen(Color.Black, 1);
        pen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
        sf.LineAlignment = sf.Alignment = StringAlignment.Center;

        // Виведення горизонтальних осей

         for (i = 0; i < KDY + 1; i++)
         {
             koor_y = (int)((Ymax - Ymin - CD * i) / K_e_py +
                            ots_ver1 + ots_ver2);
             buf = string.Format("{0:F0}", Ymin + CD * i);

             gr.DrawString(buf, font_os, Brushes.Black,
                           ots_lev1/2,  koor_y, sf);
             gr.DrawLine(pen,ots_lev1,koor_y,ots_lev1+ots_lev2+
                     (int)((Xmax - Xmin) / K_e_px), koor_y);
             }
```

```
    // Виведення вертикальних осей
         sf.LineAlignment = StringAlignment.Near;
         sf.Alignment = StringAlignment.Center;

        for (i = 0; i < KDX + 1; i++)
        {
          koor_y = ots_ver1 +ots_ver2;
          koor_x = (int)((Xmax - Xmin - CD * i) / K_e_px +
                   ots_lev1 + ots_lev2);
          gr.DrawLine(pen, koor_x, koor_y, koor_x,koor_y +
                   (int)((Ymax - Ymin) /
          K_e_py + ots_niz2));
          koor_y += (int)((Ymax - Ymin) / K_e_py + ots_niz2);
          if ((i & 1) == 0)//четные
          {
           buf = string.Format("{0:F0}", Xmax - CD * i);
           gr.DrawString(buf, font_os, Brushes.Black, koor_x,
                        koor_y, sf);
             }
         }

       //Виведення показників
         sf.LineAlignment = StringAlignment.Far;
         sf.Alignment = StringAlignment.Center;
     for (i = 0; i < KOL; i++)
     {
        if (X[i] > Xmax || X[i] < Xmin || Y[i] > Ymax ||
            Y[i] < Ymin) continue;
        koor_x = (int)((X[i]-Xmin)/K_e_px+ots_lev1+ots_lev2);
        koor_y = (int)((Ymax-Y[i])/K_e_py+ots_ver1+ots_ver2);
        buf = string.Format("{0:F0}", Z[i]);
        gr.DrawString(buf, font_pok, Brushes.Black,
        koor_x, koor_y , sf);
        gr.FillEllipse(new SolidBrush(Color.Red), koor_x - 3,
                       koor_y - 3, 6, 6);
         }
 }

 private void Form1_MouseDown(object sender,MouseEventArgs e)
{
     PointEnd = PointBegin = e.Location;
     Cursor = Cursors.Cross;
}

private void Form1_MouseUp(object sender, MouseEventArgs e)
 {
     int x1, x2, y1, y2;
     Cursor = Cursors.Arrow;
     if (e.Button == MouseButtons.Left)
     {
      x1 = PointBegin.X<PointEnd.X ? PointBegin.X:PointEnd.X;
      x2 = PointBegin.X>PointEnd.X ? PointBegin.X:PointEnd.X;
      y1 = PointBegin.Y<PointEnd.Y ? PointBegin.Y:PointEnd.Y;
```

```
        y2 = PointBegin.Y>PointEnd.Y ? PointBegin.Y:PointEnd.Y;

        if (x1 == x2 || y1 == y2) return;
        Xmax = (x2 - ots_lev1 - ots_lev2) * K_e_px + Xmin;
        Xmin = (x1 - ots_lev1 - ots_lev2) * K_e_px + Xmin;
        Ymin = Ymax - (y2 - ots_ver1 - ots_ver2) * K_e_py;
        Ymax = Ymax - (y1 - ots_ver1 - ots_ver2) * K_e_py;
        Invalidate();
        return;
       }
        if (e.Button != MouseButtons.Right) return;
        if (PointBegin == PointEnd)         return;
           Xmin = Xmin - (PointEnd.X - PointBegin.X) * K_e_px;
           Ymin = Ymin - (PointBegin.Y - PointEnd.Y) * K_e_py;
           CD = -1 * CD; // для сохранения CD в функции SCROLL
           Invalidate();
    }

private void Form1_MouseMove(object sender,MouseEventArgs e)
    {
        double x, y;
        Point pt = e.Location;
        x = (pt.X - ots_lev1 - ots_lev2) * K_e_px + Xmin;
        y = Ymax - (pt.Y - ots_ver1 - ots_ver2) * K_e_py;
        Text = string.Format("X = {0:F2} Y = {1:F2} ", x,y);
        if (flag)
        {
            flag = false;
            Cursor = Cursors.Arrow;
            Invalidate();
        }
        if (e.Button == MouseButtons.Left)
        {
            Rectangle rect;
            Point pt1, pt2;
            pt1 = PointToScreen(PointBegin);
            Peres(ref PointEnd);
            pt2 = PointToScreen(PointEnd);
            rect = new Rectangle(pt1.X, pt1.Y, pt2.X -
                   pt1.X, pt2.Y - pt1.Y);
            ControlPaint.DrawReversibleFrame(rect,
            Color.FromArgb(255, 255, 0), FrameStyle.Dashed);
            PointEnd = e.Location;
            Peres(ref PointEnd);
            pt2 = PointToScreen(PointEnd);
            rect = new Rectangle(pt1.X, pt1.Y, pt2.X -
                   pt1.X, pt2.Y - pt1.Y);
            ControlPaint.DrawReversibleFrame(rect,
            Color.FromArgb(255, 255, 0), FrameStyle.Dashed);
            return;
        }
```

```
      if (e.Button == MouseButtons.Right)
       {
                Peres(ref PointEnd);
      ControlPaint.DrawReversibleLine(PointToScreen(PointBegin),
          PointToScreen(PointEnd), Color.FromArgb(255, 255, 0));
      PointEnd = e.Location;
      Peres(ref PointEnd);
      ControlPaint.DrawReversibleLine(PointToScreen(PointBegin),
          PointToScreen(PointEnd), Color.FromArgb(255, 255, 0));
      return;
      }
  }

//Функція перерахунку координат точок при виході за межі вікна
  private void Peres(ref Point pt)
  {
      pt.X = (pt.X < ClientRectangle.Left) ?
              ClientRectangle.Left :pt.X;
      pt.X = (pt.X > ClientRectangle.Right) ?
              ClientRectangle.Right : pt.X;
      pt.Y = (pt.Y < ClientRectangle.Top + menuStrip1.Height) ?
              ClientRectangle.Top + menuStrip1.Height : pt.Y;
      pt.Y = (pt.Y > ClientRectangle.Bottom) ?
              ClientRectangle.Bottom : pt.Y;
   }

       protected override void OnMouseWheel(MouseEventArgs e)
       {
           base.OnMouseWheel(e);
           double x, y, k_uv, vr_cd;
           Text = "Работает колесико";
           Cursor = Cursors.Cross;
           flag = true;
           Point pt = e.Location;
           x = (pt.X - ots_lev1 - ots_lev2) * K_e_px + Xmin;
           y = Ymax - (pt.Y - ots_ver1 - ots_ver2)* K_e_py;
           k_uv = (e.Delta > 0) ? 1.1 : 0.9;
           vr_cd = Math.Floor(CD * k_uv / N) * N;
           if (Math.Abs(CD - vr_cd) < 0.001)
           vr_cd = (k_uv > 1) ? vr_cd+=N : vr_cd-=N;
           if (vr_cd <= N) vr_cd = N;
           if (vr_cd > (Xmax_d - Xmin_d) &&
               vr_cd > (Ymax_d - Ymin_d))
               return;
           Xmin = x - (x - Xmin) / CD * vr_cd;
           Ymin = y - (y - Ymin) / CD * vr_cd;
           CD = -vr_cd;
           Invalidate();
        }
```

```
        //Сформувати дані
        private void сформироватьДанныеToolStripMenuItem_Click
                    (object sender, EventArgs e)
        {
            Form_mas();
            Nastroyka();  // Налаштування
            Max_Min();    // Max-Min
            Invalidate();
        }

        //Оновити
        private void обновитьToolStripMenuItem_Click
                     (object sender, EventArgs e)
        {
            Nastroyka();  // Налаштування
            Max_Min();    // Max-Min
            Invalidate();
        }
    }
}
```

Рис. 35. Робота з двовимірною графікою

12. Курсор мишки

Курсор мишки – це невелика растрова картинка, що показує позицію мишки на екрані. Як відомо, в залежності від положення на екрані його зовнішній вигляд може змінюватися. Часто це стрілка, але якщо навести її на границю форми, вона перетвориться у двоголову стрілку. У полі для введення тексту курсор стає *I*-образним. Курсор мишки – це об'єкт типу *Cursor,* визначений у просторі імен *System.Windows.Forms.* Взагалі найлегше отримати курсор за допомогою класу *Cursors* (зверніть увагу на множину). Клас *Cursors* також визначається у просторі імен *System.Windows.Forms* і складається виключно із 28 статичних властивостей.

Три найбільш часто використовувані властивості класу *Cursor*:

Position (повернення *Point*)
Clip (повернення *Rectangle*)
Current (повернення *Cursor*)

Cursor.Position рідко встановлює позицію курсору. Властивість *Cursor.Clip* обмежує область руху курсору мишки заданим прямокутником. Цю властивість можна встановити лише при захопленні мишки. Значення властивостей *Position* і *Clip* обчислюються в координатах екрану, тому після отримання їх значення, необхідно використати метод *PointToClient,* а перед їх установкою – метод *PointToScreen.* Поточний курсор мишки можна встановлювати за допомогою властивості *Cursor.Current*. Як відомо, програми, що тривало обробляють великі завдання, зазвичай виводять курсор у вигляді пісочного годинника за допомогою оператору:

Cursor.Current = Cursors.WaitCursor

Після цього, закінчивши тривалу обробку, програма може відновити попередній вигляд курсору:

Cursor.Current = Cursors.Arrow

Приклад використання пісочного годинника наведено на лістингу 32.

Лістинг 32. (exam32)

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam32
{
    public partial class Form1 : Form
    {

        public Form1()
```

```
        {

            InitializeComponent();
            Cursor = Cursors.Cross;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            //Текущий курсор во время выполнения функции
            Cursor.Current = Cursors.WaitCursor;
            for (int i = 0; i < 100000; i++)
                Text = i.ToString();
        }
    }
}
```

13. Вправи визначення позиції курсору

Зображуючи на формі графічні фігури і текст, зазвичай спочатку визначають координати всіх елементів, після чого малюють їх, викликаючи відповідні методи. Але часто в програмах застосовують інтерфейс на основі мишки, щоб дати користувачеві можливість вибирати графічні об'єкти та маніпулювати ними. Отже, програма повинна працювати за зворотним принципом, визначаючи координати покажчика, щоб з'ясувати, на який графічний елемент вказує мишка. Цей процес називається *визначенням позиції курсору* (*hit-testing*). Він може бути досить складним, особливо якщо клієнтське вікно містить фігури, що перекриваються, і текст, набраний пропорційним шрифтом, але іноді визначити позицію курсору досить просто. Фактично наведена нижче програма на лістингу 33 визначає позицію курсору. Програма малює у своїй клієнтській області кілька прямокутників. Якщо клацнути намальований програмою прямокутник, з'явиться хрестик, який зникне при повторному клацанні.

Лістинг 33. (exam 33)

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

namespace exam33
{
    public partial class Form1 : Form
    {
    int xNum = 5; // Кількість прямокутників по горизонталі
```

```
 int yNum = 4; // Кількість прямокутників по вертикалі
 bool[,] abChecked;
 int cxBlock, cyBlock;

     public Form1()
     {
         abChecked = new bool[xNum, yNum];
         InitializeComponent();
         Text = "Gheck";
         BackColor = SystemColors. Window;
         ForeColor = SystemColors.WindowText;
         ResizeRedraw = true;
         OnResize(EventArgs.Empty);
     }
protected override void  OnResize(EventArgs e)
{
    //MessageBox.Show("OnResize");
    cxBlock = ClientSize.Width/xNum;
    cyBlock = ClientSize.Height/yNum;
    base.OnResize(e);
}

  protected override void  OnMouseUp(MouseEventArgs e)
 {
     int x = e.X/cxBlock;
     int y = e.Y/cyBlock;
     if (x < xNum && y < yNum)
       {
         abChecked[x, y] = true;
         Invalidate(new Rectangle(x * cxBlock, y * cyBlock,
                    cxBlock, cyBlock));
        }
        base.OnMouseUp(e);
 }

   protected override void  OnPaint(PaintEventArgs e)
 {
        Graphics gr = e.Graphics;
        Pen pen = new Pen(ForeColor);
        for(int i = 0; i < xNum;i++)
        for(int j = 0; j < yNum;j++)
        {
          gr.DrawRectangle(pen,i*cxBlock, j*cyBlock,
                           cxBlock, cyBlock);
          if(abChecked[i,j])
          {
            gr.DrawLine(pen,i*cxBlock, j*cyBlock,
                        (i+1)*cxBlock, (j+1)*cyBlock);
            gr.DrawLine(pen,(i+1)*cxBlock, j*cyBlock,
                         i*cxBlock, (j+1)*cyBlock);
          }
       }
   base.OnPaint(e);
```

```
    }

      protected override void OnKeyDown(KeyEventArgs kea)
       {
         Point ptCursor = PointToClient(Cursor.Position);
         int x = Math.Max(0, Math.Min(xNum - 1,
                          ptCursor.X /cxBlock));
         int y = Math.Max(0, Math.Min(yNum - 1,
                          ptCursor.Y /cyBlock));
            switch(kea.KeyCode)
            {
             case Keys.Up: y--; break;
             case Keys.Down: y++; break;
             case Keys.Left: x--; break;
             case Keys.Right: x++; break;
             case Keys.Home: x = y = 0; break;
             case Keys.End: x = xNum - 1;
             y = yNum - 1; break;
             case Keys.Enter:
             case Keys.Space:
             abChecked[x,y] = true;
             Invalidate(new Rectangle(x*cxBlock,y*cyBlock,
                        cxBlock,cyBlock));
             return;

             default:
             return;
            }
            x = (x + xNum)%xNum;
            y = (y + yNum)%yNum;
            Cursor.Position = PointToScreen(new Point(x *
            cxBlock + cxBlock / 2, y * cyBlock + cyBlock / 2));
            base.OnKeyDown(kea);
       }
    }
}
```

При будь-якій зміні розмірів форми програма перераховує значення *cxBlock* і *cyBlock*, що задають розміри кожного прямокутника. Програма *також* підтримує масив *abChecked*, що зберігає значення типу *bool*, кожне з яких вказує, чи є хрестик у деякому прямокутнику. Метод *OnPaint* малює контур кожного прямокутника і, якщо значення *abChecked* для цього прямокутника – *true*, малює у ньому хрестик. Визначення позиції курсору відбувається при виконанні методу *OnMouseUp*. Програма ділить значення координат мишки на *cxBlock* та *cyBlock*, щоб отримати значення індексів у масиві *abChecked.*

Результат роботи програми показаний на рис. 36.

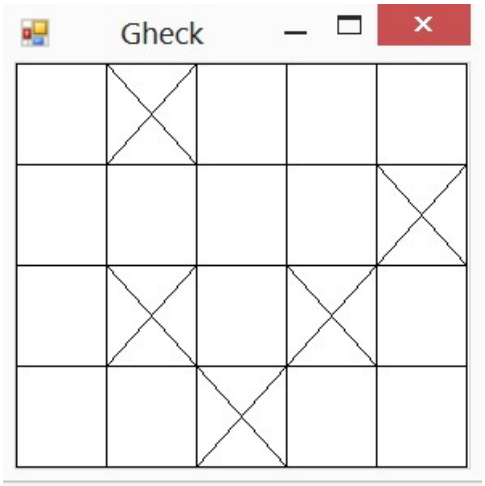


Рис. 36. До визначення позицій курсору

Розглянутий вище приклад можна вирішити простіше. Не обов'язково використовувати готові елементи управління, коли можна зробити свої, створивши підкласи класу *Control. Control,* – це основа всіх зумовлених елементів управління в Windows Forms. Однак при створенні власних елементів управління рекомендується використовувати у якості батьківського класу *UserControl*, який спадкує *Control* через *ScrollableControl* і *ContainerControl*. У лістингу 34 кожний прямокутник являє собою дочірній клас *CheckerChild,* похідний від *UserControl.* У нього єдине поле типу *bool,* значення якого змінюється у відповідь на виклик функції *OnMouseDown*() при натисненні кнопки мишки. При виконанні методу *OnPaint* програма малює контур прямокутника і, якщо значення змінної типу *bool* дорівнює *true*, малює в прямокутнику хрестик. Цей клас також реагує на натискання клавіш *Enter* та пробіл.

У даній програмі не визначається позиція курсору. Дочірньому елементу управління абсолютно не важливо, де його клацають. *Windows* сама визначає позицію курсору у конкретному дочірньому об’єкті.

Лістинг 34. (exam34)

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

namespace exam34 // Дочірні об’єкти
{
    public partial class Form1 : Form
    {
        int xNum = 5;
        int yNum = 6;
```

```
    CheckerChild[,] ch;

    public Form1()
    {
        ch = new CheckerChild[xNum, yNum];
        for (int i = 0; i < xNum; i++)
          for (int j = 0; j < yNum; j++)
            {
                ch[i, j] = new CheckerChild();
                ch[i, j].Parent = this;
            }
        InitializeComponent();
      }
    protected override void OnResize(EventArgs e)
    {
        int cx = ClientSize.Width/xNum;
        int cy = ClientSize.Height / yNum;
        for (int i = 0; i < xNum; i++)
        for (int j = 0; j < yNum; j++)
        {
             ch[i,j].Location = new Point(i*cx, j*cy);
             ch[i,j].Size = new Size(cx,cy);
        }
    }
 }

class CheckerChild : UserControl
{
    bool Check = false;
        void OnClick()
      {
        Check = !Check;
        Invalidate();

      }

    protected override void OnMouseDown(MouseEventArgs e)
    {
        base.OnMouseUp(e);
        OnClick();

    }
    protected override void OnKeyDown(KeyEventArgs e)
    {
       switch (e.KeyCode)
        {
            case Keys.Enter:
            case Keys.Space:
                OnClick();
                break;
        }
     }
    protected override void OnPaint(PaintEventArgs e)
```

```
        {
            Graphics gr = e.Graphics;
            Pen pen = new Pen(ForeColor);
            gr.DrawRectangle(pen, ClientRectangle);
            if (Check)
             {
                gr.DrawLine(pen, 0, 0, ClientSize.Width,
                            ClientSize.Height);
                gr.DrawLine(pen, ClientSize.Width, 0, 0,
                            ClientSize.Height);
            }
        }
    }
}
```

Результат роботи програми показаний на рис. 37.

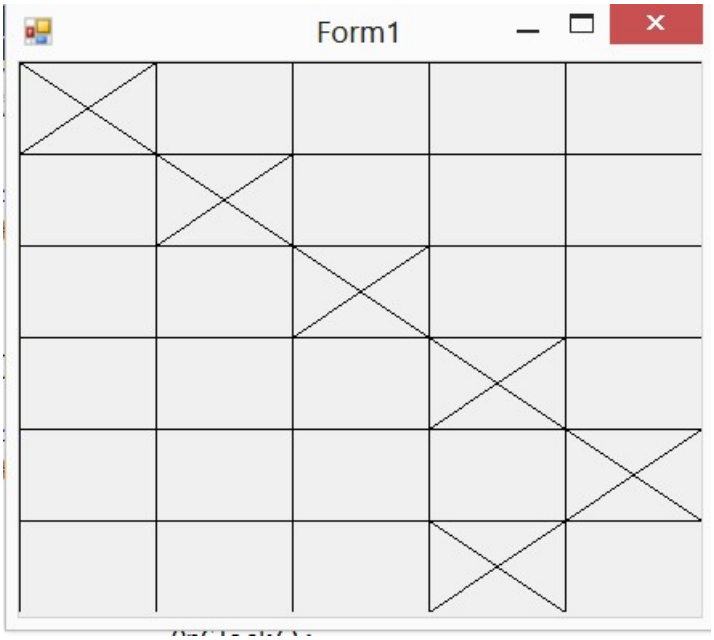


Рис. 37. Робота з дочірніми вікнами

14. Відображення символів, що набираються на екрані

Елементи управління або форми, що отримують введену з клавіатури інформацію, зазвичай якось показують, що вони знаходяться у фокусі. Елементи управління і форми, що дозволяють вводити текст, зазвичай відзначають місце появи наступного символу на екрані короткою горизонтальною (вертикальною) лінією або квадратиком. Цей індикатор називається *кареткою.* Якщо треба написати власний код для введення тексту, виникає проблема. Серед функцій, яких немає у бібліотеках класів *Windows Forms*, є функції створення і управління кареткою. Виникає необхідність писати код з використанням *DLL Windows*. У роботі [2] Петцольдом Ч. розроблений клас *Caret*. Він заснований на п’яти *API* функціях *Windows*, розташованих у бібліотеці User32.dll.

У лістингах 35а и 35б наводиться програма, яка дозволяє вводити і редагувати текст за допомогою класу *Caret.* По функціональності вона дуже близька до елемента управління *TextBox,* працюючому у режимі обробки одного рядку. Першочергово завантажується форма, кодова частина якої містить клас *Aw* (лістинг 35а). Цей клас є похідним від класу *Aw_1*, який знаходиться у лістингу 35б. У цьому ж лістингу знаходиться клас *Caret.* Клас *Aw_1* призначений для керування текстом з клавіатури (введення, видалення символів з кінця тексту). Клас *Aw_1* за допомогою курсору мишки дозволяє встановлювати каретку всередині тексту для наступного введення або видалення символу відносно встановленого положення каретки. Класи *Aw_1* і *Aw* використовують клас *Caret.*

Клас *Aw_1* створює об'єкт типу *Caret* у своєму конструкторі та ініціалізує його розмір і положення. При малюванні на формі програма повинна лише сховати каретку до події *Paint* і знову показати її після цієї події, а також встановити позицію каретки в клієнтській області. Рядок символів, що вводиться і редагується користувачем, зберігається в полі *strText.* Поле in*sert* зберігає точку вставки у рядок. Наприклад, після набору трьох символів значення *insert* дорівнює 3. Якщо потім натиснути "стрілку-ліворуч", *insert* буде дорівнювати 2. Метод *PositionCaret* відповідає за перетворення позиції символу у позицію пікселів. Результат цього перетворення слугує для установки властивості *Position* об'єкту *Caret*. Розглянемо метод *OnKeyPress*, який при натисненні клавіш формує текст. Програма стирає текст форми за допомогою методів *MeasureString* і *FillRectangle.* Це необхідно, якщо точка вставки знаходиться не в кінці рядка. Метод *OnKeyPress* обробляє натиснення клавіші *BackSpace*, видаляючи з рядку символ, розташований перед точкою вставки. Цей метод ігнорує керуючі символи повернення каретки і переведення рядку, і поміщає всі інші символи в рядок у точці вставки. Після він виводить весь рядок повністю і викликає метод *PositionCaret*. Зауважте, що метод ховає каретку під час малювання на формі.

Метод *OnKeyDown* обробляє лише клавіші керування курсором, переміщуючи точку вставки, і клавішу *Delete*, імітуючи дію *BackSpace*. Він також викликає *PositionCaret.*

Як уже зазначалося, метод *PositionCaret* відповідає за перетворення позиції точки вставки *(insert)* у координати пікселів каретки. Він робить це за допомогою *MeasureString.* На жаль, версія *MeasureString* за умовчанням не забезпечує точності, необхідної для додатку такого роду. При розріхунку довжини рядку дана версія *MeasureString* зазвичай не враховує кінцеві пробіли. Щоб вийти із ситуації, програма використовує версію *MeasureString* з аргументом *StringFormat* і заносить у властивість *FormatFlag* аргументу *StringFormat* значення перелічення *StringFormatFlags. MeasureTrailingSpaces.* Без цих змін каретка рухалася б при наборі букв слова, але залишалася б на місці при введенні пробілу, що поділяє слова.

Але одних цих заходів недостатньо, щоб точно вирівняти каретку і виведений текст. В методах *MeasureString* и *DrawString* є засоби компенсації спотворень, пов'язаних із незалежною від пристрою растеризацією контур-

них шрифтів. Для цього використовується об'єкт StringFormat, одержуваний із статичної властивості *StringFormat.GenericTypographic.*

Приклад:

StringFormat strfmt = StringFormat.GenericTypographic;
strfmt.FormatFlags |= StringFormatFlags.MeasureTrailingSpaces;

Лістинг 35а. (exam35)

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

namespace exam35
{
    public partial class Aw : Aw_1
    {
        public Aw()
        {
            InitializeComponent();
            Cursor =  Cursors.IBeam;
        }

        protected override void OnMouseDown(MouseEventArgs e)
        {
            if (strText.Length == 0) return;
            Graphics gr = CreateGraphics();
            float xP = 0;
            int i;
            for (i = 0; i < strText.Length; i++)
            {
                SizeF sf =
                gr.MeasureString(strText.
                Substring(0, i + 1), Font, PointF.Empty,
                StringFormat.GenericTypographic);
                if (Math.Abs(e.X - xP) <
                    Math.Abs(e.X - sf.Width)) break;
                xP = sf.Width;
            }
            this.insert = i;
            gr.Dispose();
            this.PositionCaret();
            base.OnMouseDown(e);
        }
    }
}
```

Лістинг 35б. (exam35)

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

namespace exam35
{
    public  class Aw_1 : Form
    {
        public Caret caret;
        public string strText = "";
        public int insert = 0;

        public Aw_1()
        {
            InitializeComponent();
            FontHeight = 24;
            caret = new Caret(this);
            caret.Size = new Size(2, Font.Height);
            caret.Position = new Point(0, 0);
        }
        protected override void OnKeyDown(KeyEventArgs e)
        {
            //PositionCaret();

            base.OnKeyDown(e);
            switch (e.KeyData)
            {
            case Keys. Left:
                if (insert > 0)
                    insert--;
                    break;
            case Keys.Right:
                if (insert < strText.Length)
                    insert++;
            break;
            case Keys.Home:
                insert = 0;
                break;
            case Keys.End:
                insert = strText. Length;
            break;
            case Keys.Delete:
            if (insert < strText.Length)
```

```
          {
            insert++;
            Aw_1_KeyPress(this, new KeyPressEventArgs ('\b'));
          }
             break;
             default:
             return;
          }
          PositionCaret();
      }

private void Aw_1_KeyPress(object sender,KeyPressEventArgs e)
    {
      caret.Hide();
      Graphics grfx = CreateGraphics();
      grfx.FillRectangle(new SolidBrush(BackColor),
      new RectangleF(Point.Empty,grfx.MeasureString(strText,
      Font, Point.Empty, StringFormat.GenericTypographic)));

    switch (e.KeyChar)
    {
     case '\b':
           if (insert > 0)
           {
               strText = strText.Substring(0, insert - 1) +
               strText.Substring(insert);
               insert --;
           }
      break;
 case '\r':
 case '\n':
      break;
 default:
      if (insert == strText.Length)
          strText += e.KeyChar;
      else
          strText = strText.Substring(0, insert) +
          e.KeyChar + strText.Substring(insert);
      insert++;
      break;
       }
     //grfx.TextRenderingHint = TextRenderingHint.AntiAlias;
     grfx.DrawString(strText, Font, new SolidBrush(ForeColor),
     0, 0, StringFormat.GenericTypographic);
     grfx.Dispose();
     PositionCaret();
     caret.Show();
  }
      private void Aw_1_Paint(object sender, PaintEventArgs e)
      {
         Graphics grfx = e.Graphics;
         grfx.TextRenderingHint = TextRenderingHint.AntiAlias;
         grfx.DrawString(strText,Font,new
```

```
            SolidBrush(ForeColor), 0, 0,
                        StringFormat.GenericTypographic);
        }

    protected void PositionCaret()
    {
       Graphics grfx = CreateGraphics();
       string str = strText.Substring(0, insert);
       // Работа с MeasureString
       StringFormat strfmt = StringFormat.GenericTypographic;
       strfmt.FormatFlags |=
              StringFormatFlags.MeasureTrailingSpaces;
       SizeF sizef = grfx.MeasureString(str, Font,
                                        Point.Empty, strfmt);
       caret.Position = new Point((int)sizef.Width, 0);
       grfx.Dispose();
    }

    private void InitializeComponent()
    {
        this.SuspendLayout();
        // Aw_1
        this.ClientSize = new System.Drawing.Size(292, 266);
        this.Name = "Aw_1";
        this.KeyPress += new System.Windows.Forms.
             KeyPressEventHandler(this.Aw_1_KeyPress);
        this.ResumeLayout(false);
    }
   }

   public class Caret
  {
   [DllImport("user32.dll")]
   public static extern int CreateCaret(IntPtr hwnd, IntPtr hbm,
   int cx, int cy);
   [DllImport("user32.dll")]
   public static extern int DestroyCaret();
   [DllImport("user32.dll")]
   public static extern int SetCaretPos(int x, int y);
   [DllImport("user32.dll")]
   public static extern int ShowCaret(IntPtr hwnd);
   [DllImport("user32.dll")]
   public static extern int HideCaret(IntPtr hwnd);
   // Поля
   Control ctrl;
   Size size;
   Point ptPos;
   bool bVisible;
    // Конструкторы
    // Конструктор ло умолчанию недоступен.
    //private Caret() { ;}
    // Только у допустимого конструктора есть аргумент Control.
```

```
  public Caret(Control ctrl)
  {
    this.ctrl = ctrl;
    Position = Point.Empty;
    Size = new Size(1, ctrl.Font.Height);
    Control.GotFocus += new EventHandler(ControlOnGotFocus);
    Control.LostFocus+=new EventHandler(ControlOnLostFocus);
   // Создать каретку, если элемент управления уже в фокусе. ,
    if (ctrl.Focused)
    ControlOnGotFocus(ctrl, new EventArgs());
  }

  //Свойства
public Control Control
{
  get
  {
    return ctrl;
  }
}
public Size Size
{
   get
   {
   return size;
   }
  set
  {
  size = value;
  }
}

public Point Position
{
  get
  {
    return ptPos;
  }
  set
  {
    ptPos = value;
    SetCaretPos(ptPos.X, ptPos.Y);
  }
}
public bool Visibility
{
   get
  {
  return bVisible;
  }
  set
  {
  if (bVisible = value)
```

```
            ShowCaret(Control.Handle);
        else
            HideCaret(Control.Handle);
        }
    }
      public void Show()
      {
       Visibility = true;
      }
      public void Hide()
      {
       Visibility = false;
      }
      public void Dispose()
      {

      // Знищити каретку при втраті фокусу.
     if (ctrl.Focused)
     {
       ControlOnLostFocus(ctrl, new EventArgs());
       Control.GotFocus  -=  new EventHandler(ControlOnGotFocus);
       Control.LostFocus -=  new EventHandler(ControlOnLostFocus);
     }
}

      // Обробники подій
      void ControlOnGotFocus(object obj, EventArgs sa)
      {
      CreateCaret(Control.Handle, IntPtr.Zero, Size.Width,
                  Size.Height);
      SetCaretPos(Position.X, Position.Y);
      Show();
      }

     void ControlOnLostFocus(object obj, EventArgs sa)
     {
       Hide();
       DestroyCaret();
     }
    }
}
```

Результат роботи програми показаний на рис. 38.

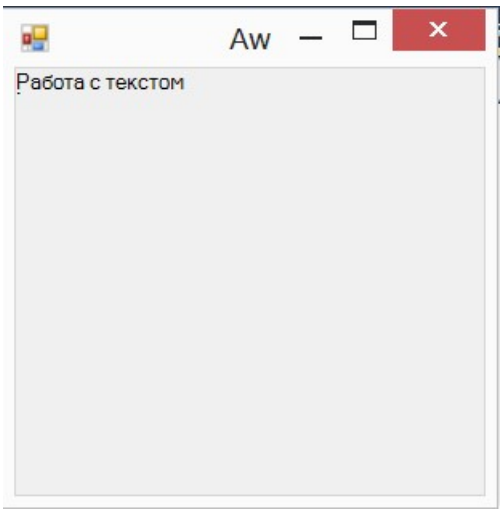


Рис. 38. Робота с текстом

15. Особливості роботи з текстом

15.1. Зображення шрифтів і визначення довжин рядків

Розглянемо більш детально роботу із шрифтами. У просторі імен *System.Drawing* визначені два важливих класи формування шрифтів:

• *FontFamily* визначений як рядок, наприклад "Times New Roman";

• *Font* – це комбінація із назви шрифту (объект *FontFamily* або символьний рядок, що визначає гарнітуру), атрибутів (таких як курсив або напівжирний) і кегля.

Почнемо з класу *Font.* Є три категорії конструкторів *Font,* засновані на:

1. Існуючому об’єкті *Font*.
2. Символьному рядку, що визначає гарнітуру.
3. Об’єкті *FontFamily.*

Найпростіший конструктор *Font* створює новий шрифт на базі існуючого. Новий шрифт відрізняється лише зображенням:

Font(Font font, FontStyle fs), де

FontStyle – це перелічення, що складається із серії однобітних прапорів:

Член	Значення
Regular	0
Bold	1
Italic	2
Underline	4
Strikeout	8

Припустимо, ми маємо шрифт *font,* отриманий із властивості *Font* форми:

Font font = Font;

Для даного шрифту можна створити курсивне зображення:

Font fontItalic = new Font (font, FontStyle.Italic);

Можна використати декілька членів переліченння:

Font fontBoldStrikeout = new Font(font, FontStyle.Bold | FontStyle.Strikeout);

На лістингу 36 наводяться різні зображення тексту, що виводиться. Програма використовує *MeasureString* для того, щоб визначити розміри кожного фрагменту тексту і розташувати текст горизонтально. Як видно із лістингу 36 та рис. 39, перші два рядки формуються шляхом з'єднання окремих фрагментів тексту та мають різні довжини. Це пояснюється різним способом використання методу *MeasureString* при визначенні довжин фрагментів.

У першому випадку метод *MeasureString* використовується без аргументу *StringFormat.* У даному випадку довжина визначається довжиною фрагменту плюс пробіл у кінці фрагменту у тому випадку, якщо у кінці фрагменті пробілу немає. У загальному випадку, якщо у конці рядка є декілька пробілів, довжина в *MeasureString* визначається лише для одного кінцевого пробілу.

У другому випадку, тобто. при виведенні другого рядка його довжина коротше. У даному випадку визначається довжина фрагменту без врахування кінцевого пробілу. Це досягається використанням переліченння *StringFormat.GenericTypographic* класу *StringFormat*.

Лістинг 36 (exam36)

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

namespace exam36
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

       protected override void OnPaint(PaintEventArgs e)
       {
         base.OnPaint(e);
         Graphics gr = e.Graphics;
         const string str1 = "This is some";
         const string str2 = "bold";
         const string str3 = " text and this is some";
         const string str4 = "italic";
```

```
        const string str5 = "text";
        float x;
        float y = 0;
        Brush brush = new SolidBrush(Color.FromArgb(255,0,0));
        Font fontRegular = Font;
        Font fontBold = new  Font(fontRegular, FontStyle.Bold);
        Font fontItalic = new Font(fontRegular,
                                   FontStyle.Italic);
        StringFormat strfmt = StringFormat.GenericTypographic;
        for (int ii = 0 ; ii < 2 ; ii++)
         {
          x = 0;
          gr.DrawString(str1, fontRegular, brush, x, y);
          fontRegular, Point.Empty, strfmt).Width;
          gr.DrawString(str2, fontBold, brush, x, y);
          x+= ii == 0 ? gr.MeasureString(str2, fontBold).Width:
      gr.MeasureString(str2,fontBold,Point.Empty,strfmt).Width;
        gr.DrawString(str3, fontRegular, brush, x, y);
        x+= ii == 0 ?gr.MeasureString(str3,fontRegular).Width:
        gr.MeasureString(str3, fontRegular, Point.Empty,
                         strfmt).Width;
        gr.DrawString(str4, fontItalic, brush, x, y);
        x+= ii == 0 ? gr.MeasureString(str4,fontItalic).Width:
        gr.MeasureString(str4, fontItalic, Point.Empty,
                         strfmt).Width;
        gr.DrawString(str5, fontRegular, brush, x, y);
        x+= ii == 0 ?gr.MeasureString(str5,fontRegular).Width:
        gr.MeasureString(str5, fontRegular, Point.Empty,
                         strfmt).Width;
        y += gr.MeasureString(str5, fontRegular, Point.Empty,
                         strfmt).Height;
        gr.DrawLine(Pens.Black, 0, y, x, y);
        y *= 2;
       }

    string[] aff = { "Courier New", "Arial","Times New Roman" };
    FontStyle[] afs = {FontStyle.Regular,
    FontStyle.Bold,FontStyle.Italic,
    FontStyle.Bold |FontStyle.Italic};
    y = 60;
    foreach (string sf in aff)
        foreach (FontStyle fs in afs)
    {
     Font font = new Font(sf, 18, fs);
     gr.DrawString(sf, font, brush,0,y);
     y += font.GetHeight(gr);
     }
   }
 }
}
```

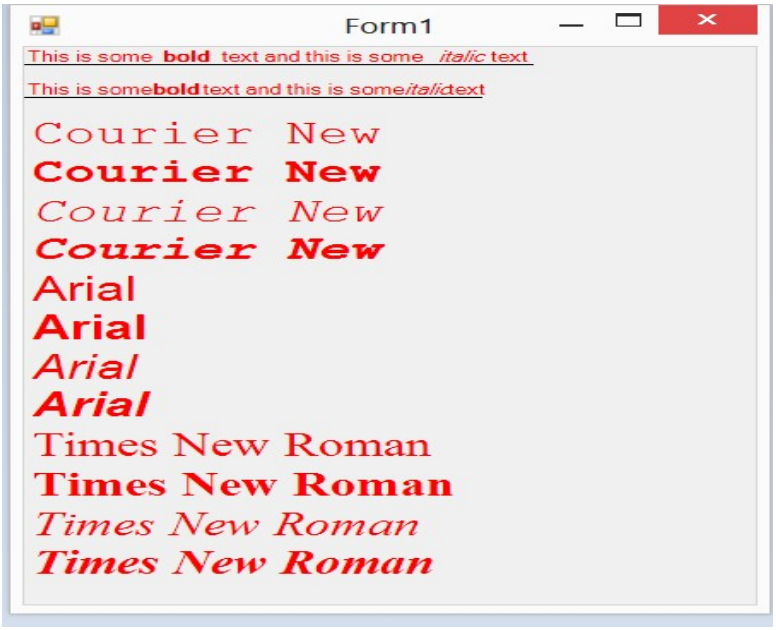


Рис. 39. До визначення довжин рядків та зображень шрифтів

15.2. Метрики дизайну

У *Windows Forms* не вистачає графічних засобів для роботи з метриками шрифтів. До сих пір ми мали справу лише з трьома параметрами, що відносяться до висот знаків: кеглем (який є поняттям друкарського дизайну і має лише непряме відношення до метричних розмірів знаків шрифту), максимальним розміром знаків, який можна отримати з *MeasureString*, і міжрядковим інтервалом, що рекомендується та визначається методом *GetHeight*. Потрібна ще одна метрика шрифту, що визначає положення базової лінії (*baseline*) – лінії, вище якої розташовуються надрядкові елементи, а нижче – підрядкові елементи. Ця інформація міститься у класі *FontFamily*, який містить чотири методи, що дозволяють отримати додаткову метричну інформацію про шрифт. У кожному з цих методів у якості аргументу виступає перелічення *FontStyle.*

Методи *FontFamily*:

int GetEmHeight (FontStyle fs);
int GetCellAscent (FontStyle fs);
int GetCellDescent (FontStyle fs);
int GetLineSpacing (FontStyle fs).

Ці величини називаються метриками дизайну, тому що задаються дизайнером шрифтів (принаймні для шрифтів *TrueType*). Вони не пов’язані з кінцевим розміром шрифту. Розглянемо приклад. Якщо створити *FontFamily* на основі гарнітури Times New Roman і викликати ці чотири методи з аргументом *FontStyle Regular* (або будь-якою іншою величиною *FontStyle),* вийдуть значення, наведені у табл. 16.

Таблиця 16

Метрики дизайну шрифту Times New Roman

Метрика	Значення
Висота ем (*Em Height*)	2048
Висота з надрядковим елементом (*Ascent*)	1825
Висота підрядкового елементу (*Descent*)	443
Висота з надрядковим елементом + висота підрядкового елементу	2268
Міжрядковий інтервал	2355

Висота «*ем*» характеризує сітку, що дозволяє дизайнеру шрифту задавати координати ліній і кривих, що визначають зовнішній вигляд знаків шрифту. Зазвичай використовується значення 2048. Висота з надрядковим елементом – це висота знаків шрифту над базовою лінією шрифту (включаючи діакритичні знаки), а висота підрядкового елемента – це висота знаків під базовою лінією шрифту. Для гарнітури Times New Roman сума висоти з надрядковим елементом і висоти підрядкового елемента (яка представлена в таблиці) представляє дійсну висоту знаків шрифту. Міжрядковий інтервал складається з трьох компонентів: висота з надрядковим елементом над базовою лінією, висота підрядкового елемента під базовою лінією і додаткове місце під підрядковим елементом, як показано на рис. 40.

Рис. 40. Графічна інтерпретація метрики дизайну

Для деяких шрифтів значення міжрядкового інтервалу більше, ніж сума висоти з надрядковим елементом і висоти підрядкового елементу, а для деяких вони рівні.

Створимо 72-пунктний шрифт Times New Roman та привласнимо властивості *PageUnit* значення *GraphicsUnitPoint.* Це означає, що *GetHeight* повертає значення у пунктах незалежно від роздільної здатності графічного пристрою. Ми отримаємо такі значення (округлені до двох знаків після коми):

Метрики шрифту Times New Roman

Властивість або метод	Значення у пунктах
font.SizeInPoints()	72
font.GetHeight()	82.79

Метрика дизайну, названа висотою *«ем»*, відповідає кеглю шрифту. Якщо розділити 72 на 2048 ми отримаємо 0,03515625. Це коефіцієнт масштабування для перетворення координат знаків шрифту у кегль. Помноживши його на міжрядковий інтервал (2355), ви отримаєте 82,79 – величину, що повертається методом *GetHeight.* Сенс у тому, що можна застосувати один і той же коефіцієнт окремо до висоти з надрядковим елементом і до висоти підрядкового елементу метрик дизайну, щоб отримати величини, яких у нас до цього не було. На основі табл. 16 і отриманого коефіцієнту масштабування можна отримати метрику у пунктах для 72-пунктового шрифту. Результати зведені в табл. 17.

Таблиця 17

Метрики дизайну шрифту Times New Roman

Метрика	Значення метрики дизайну	Значення для 72-пунктового шрифту (у пунктах)	Властивість або метод
Висота ем (*Em Height*)	2048	72	*font.SizeInPoints*()
Висота з надрядковим елементом (*Ascent*)	1825	64,16	
Висота підрядкового елементу (*Descent*)	443	15,57	
Висота з надрядковим елементом + висота підрядкового елементу	2268	79,73	
Міжрядковий інтервал	2355	82,79	*font.GetHeight*()

Інформацію, отриману із метрик дизайну, можна використати для позиціонування тексту відносно базової лінії шрифту. На лістингу 37 наводиться програма побудови базової лінії 144-пунктного шрифту Times New Roman у центрі клієнтської області. Це відображено у результаті вирішення програми на рис. 41.

Лістинг 37. (exam37)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;z
using System.Data;
```

```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam37
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
        }

    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        Graphics gr = e.Graphics;
        String strFont = "Hello Wqrld";
        Font font = new Font("Times New Roman", 20,
                             FontStyle.Italic);
        Brush br = new SolidBrush(Color.FromArgb(255,0,0));
        GraphicsState gs = gr.Save();
        gr.PageScale = 0.2f;
        SizeF sizef = gr.MeasureString(strFont, font);
        float fScaleHorz = ClientSize.Width / sizef.Width;
        float fScaleVert = ClientSize.Height / sizef.Height;
        gr.ScaleTransform(fScaleHorz,fScaleVert);
        gr.DrawString(strFont,font,br,0,0);
        font.Dispose();
        gr.Restore(gs);
        float yBaseline = ClientSize.Height *2/ 3;
        Pen pen = new Pen(Color.FromArgb(0,0,255));
        gr.DrawLine(pen,0,yBaseline,ClientSize.Width,yBaseline);
        font = new Font("Times New Roman", 144);
        float cyLineSpace = font.GetHeight(gr);
        int iGellSpace =
             font.FontFamily.GetLineSpacing(font.Style);
        int iGellAscent =
            font.FontFamily.GetCellAscent(font.Style);
        float cyAscent = cyLineSpace * iGellAscent / iGellSpace;
        string bb = font.GetHeight(gr).ToString() + "  " +
        font.FontFamily.GetLineSpacing(font.Style) + "  " +
        font.FontFamily.GetCellAscent(font.Style) + "  " +
        cyAscent.ToString();
        Text = bb;
        gr.DrawString("Baseline",font,br,0,yBaseline-cyAscent);
    }
  }
}
```

Рис. 41. Побудова базової лінії

15.3. Знайомство з глобальним перетворенням

Клас *Graphics* має 4 змінюваних властивостей, істотно впливаючих на вигляд графічних фігур:

PageScale і *PageUnit* визначають одиниці розмірів при малюванні; за умовчанням одиницею розміру при малюванні на екрані є піксель;

Transform – це об'єкт типу *Matrix,* що визначає матричне перетворення для всієї графічної інформації, що виводиться; перетворення транслює (переміщує), масштабує, обрізає та обертає точки координат;

Clip – це вирізана область; будучи встановленою, вона обмежує область виведення для будь-якої з функцій малювання, що викликаються.

Перші дві якості ми розглядали вище. Розглянемо об'єкт Transform як основу світового перетворення (*world transform*) Воно використовує традиційну числову одиничну матрицю 3 × 3. Всі операції, пов'язані з переміщенням, поворотом, масштабуванням змінюють значення чисел у матриці, яка в кінцевому рахунку визначає перетворення подання даних в двовимірний простір. Перетворення можна виконувати двома способами. Перший спосіб виконується строго прямою зміною даних матриці. При цьому окреме перетворення (наприклад, поворот на певну кількість градусів) може бути представлено окремою одиничною матрицею. А результуюче перетворення визначається добутком окремих матриць. Другий спосіб значно простіше і полягає у використанні методів класу Graphics, які забезпечують зміни матриці. Розглянуті способи не виключають їхнє спільне використання.

Методи *Graphics*:

1. Транслювання (переміщення)

void TranslateTransform(float dx, float dy)
void TranslateTransform(float dx, float dy, MatrixOrder mo)

2. Масштабування

void ScaleTransform(float sx, float sy)
void ScaleTransform(float sx, float sy, MatrixOrder mo)

3. Обертання (поворот)

void RotateTransform(float fAngle)
void RotateTransform(float fAngle, MatrixOrder mo)

4. Повертає все у вихідний стан. Формування одиничної матриці.

void ResetTransform()

Перелічення *MatrixOrder* складається всього з двох членів:

MatrixOrder.Append – об'єднання (множення матриць) нового перетворення з існуючим

MatrixOrder.Prepend – об'єднання (множення матриць) існуючого з новим

Розглянемо перетворення матриці при використанні методів перетворення.

Створимо одиничну матрицю:

grfx.Transform = new Matrix(1, 0, 0, 1 ,0 ,0) ;

Розглянемо приклад об'єднаних перетворень. Нехай спочатку був викликаний метод: *grfx.ScaleTransform*(2, 2). Після цього можна отримати результуючу матрицю, викликав:

float[] afElements = grfx.Transform.Elements;

У результаті ви побачите значення 2, 0, 0, 2, 0, 0.

Слід зазначити, що у матриці 3x3 дев'ять елементів, а в запропонованому масиві 6 елементів. Це зробили з метою скорочення масиву, оскільки тільки 6 елементів матриці змінюються.

Однак матриця буде виглядати наступним чином:

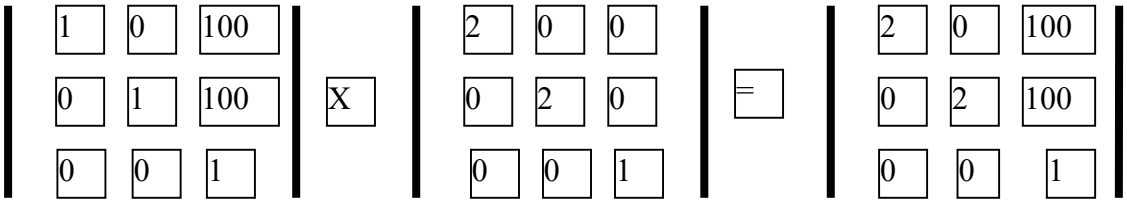
2	0	0
0	2	0
0	0	1

Тепер викличемо: *grfx.TranslateTransform*(100, 100);

Цей метод дасть таку матрицю:

1	0	100
0	1	100
0	0	1

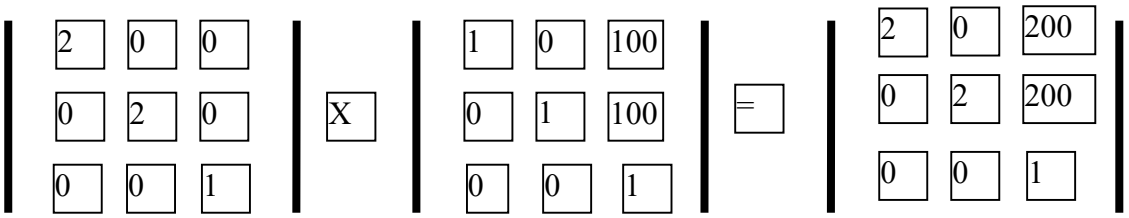
Розглянемо об’єднаний результат виклику двох методів. Матриця, обумовлена другим методом, множиться на поточне значення властивості *Transform* (визначена першим методом), у підсумку значення цієї властивості зміниться:



Тепер викличемо методи у зворотному порядку:

grfx.TranslateTransform(100, 100);
grfx.ScaleTransform(2, 2)

У цьому випадку результуюче перетворення розраховується шляхом множення другої матриці на першу:



Це ж перетворення можна отримати, викликав

grfx.ScaleTransform(2, 2);
grfx.TranslateTransform(100, 100, MatrixOrder.Append);

Розглянемо функцію повороту на наступному прикладі:

grfx.RotateTransform(45) ;

Текст повертається на 45° за годинниковою стрілкою. Послідовні виклики *RotateTransform* мають кумулятивний ефект. У результаті викликів:

grfx.RotateTransform(5);
grfx,RotateTransform(10);
grfx.RotateTransform(-20)

текст повернеться на 5° проти годинникової стрілки.

При виклику *grfx.RotateTransform(a)* із заданим кутом *a* результуюче перетворення вихідних координат *X*, *Y* виглядає так:

$$X_{рез} = X_{исх}cos(a) + Y_{исх}sin(a)$$
$$Y_{рез} = - X_{исх}sin(a) + Y_{исх}cos(a)$$

При повороті за годинниковою стрілкою на 30 градусів

$$X_{рез} = X_{исх}0.86 + Y_{исх}0.5$$
$$Y_{рез} = - X_{исх}0.5 + Y_{исх}0.86$$

Матриця буде виглядати наступним чином.

	$X_{исх}$	$Y_{исх}$	
$X_{рез}$	0.86	0.5	0
$Y_{рез}$	- 0.5	0.86	0
	0	0	1

При розробці програми розроблені 2 приклади. Ці приклади відповідно наведені у 2 формах *Form*1 и *Form*2. Як діалог в *Form*1 підключається *Form*2. У лістингах 38а та 38б наводяться тексти програм реалізації відповідно першого і другого прикладі матричного перетворення координат. На рис. 42 та рис. 43 наводяться результати вирішення відповідно першого і другого прикладів. Тут наводяться приклади переміщення, повороту і масштабування. Опис лістингів недоцільний, оскільки теоретична основа дана у достатньому обсязі.

Лістинг 38а (exam38).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam38
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        public void Matr(Graphics gr,float ots_top)
        {
            float[] af = gr.Transform.Elements;
            float ot_str, ot_stl;
            ot_str = 30;
            ot_stl = 40;
            double[,] ma = new double[3, 3];
            ma[0, 0] = af[0];
            ma[0, 1] = af[1];
            ma[0, 2] = af[4];
            ma[1, 0] = af[2];
            ma[1, 1] = af[3];
            ma[1, 2] = af[5];
            ma[2, 0] = 0;
            ma[2, 1] = 0;
            ma[2, 2] = 1;
           for (int i = 0; i < 3; i++)
           for (int j = 0; j < 3; j++)
             gr.DrawString(string.Format("{0:F2}", ma[i, j]),
             Font, Brushes.Blue, ot_stl*j, ot_str * i+ ots_top);
        }

       private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Text = "Пример 1";
            float ots_top = 50;
            Graphics gr = e.Graphics;
            Matr(gr, ots_top);
            gr.TranslateTransform(200,  0 );
            Matr(gr, ots_top);

            gr.TranslateTransform(-200, 100);
            Matr(gr, ots_top);
            gr.DrawLine(new Pen(Brushes.Blue), 0, 40, 200, 40);
            gr.TranslateTransform(200, 0);
            gr.RotateTransform(30);
            Matr(gr, ots_top);
            gr.DrawLine(new Pen(Brushes.Blue), 0, ots_top,
                        200, ots_top);
        }

        private void пример1ToolStripMenuItem_Click
                     (object sender, EventArgs e)
        {
            Form2 dlg = new Form2();
            dlg.ShowDialog();
         }
    }
}
```

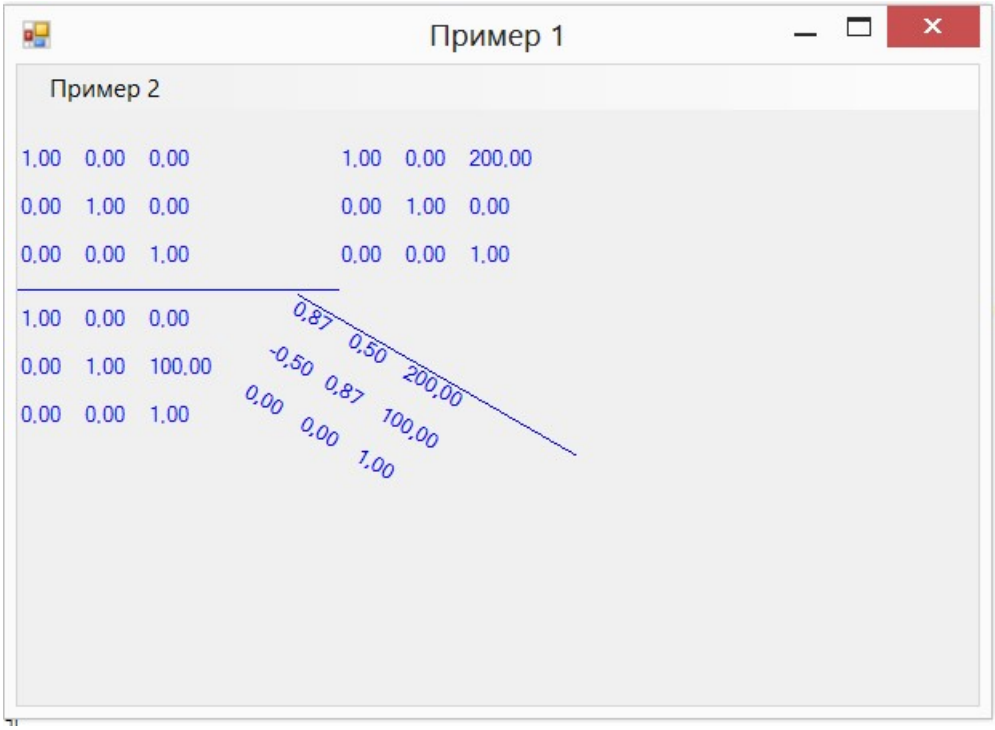


Рис. 42. Результати вирішення прикладу 1

Лістинг 38б. (exam38)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam38
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

       private void Form2_Paint(object sender, PaintEventArgs e)
        {
            Text = "Пример 2";
            Graphics gr = e.Graphics;
            double re = 10.11, im = 2.55;
            gr.DrawString(string.Format(" 1 -
                          ({0,5:0.##}:{1,5:0.##)}", re, im),
                           Font, Brushes.Blue, 0, 0);
            gr.Transform = new Matrix(1, 0, 0, 2, 100, 10);
```

```
            gr.DrawString(string.Format(" 2 -
                        ({0,5:0.##}:{1,5:0.##)}", re, im),
                        Font, Brushes.Blue, 0, 0);
            gr.Transform = new Matrix(1, 0, 0, 1, 0, 0);
            gr.ScaleTransform(3, 3);
            gr.TranslateTransform(20, 10, MatrixOrder.Append);
            gr.RotateTransform(20);
            gr.DrawString(string.Format("3 -
                         ({0,5:0.##}:{1,5:0.##)}", re, im),
                         Font, Brushes.Blue, 0, 0);
            gr.Transform = new Matrix(1, 0, 0, 1, 0, 0);
            // или gr.ResetTransform();
            gr.TranslateTransform(0, 100, MatrixOrder.Append);
            gr.DrawString(string.Format(" 4 -
                        ({0,5:0.##}:{1,5:0.##)}", re, im),
                        Font, Brushes.Blue, 0, 0);
            gr.ResetTransform();
            gr.Transform = new Matrix(4f,1f,1.0f,4f, 50f, 200f);
            gr.DrawString(string.Format(" 5 -
                        ({0,5:0.##}:{1,5:0.##)}", re, im),
                        Font, Brushes.Blue, 0, 0);
        }
    }
}
```

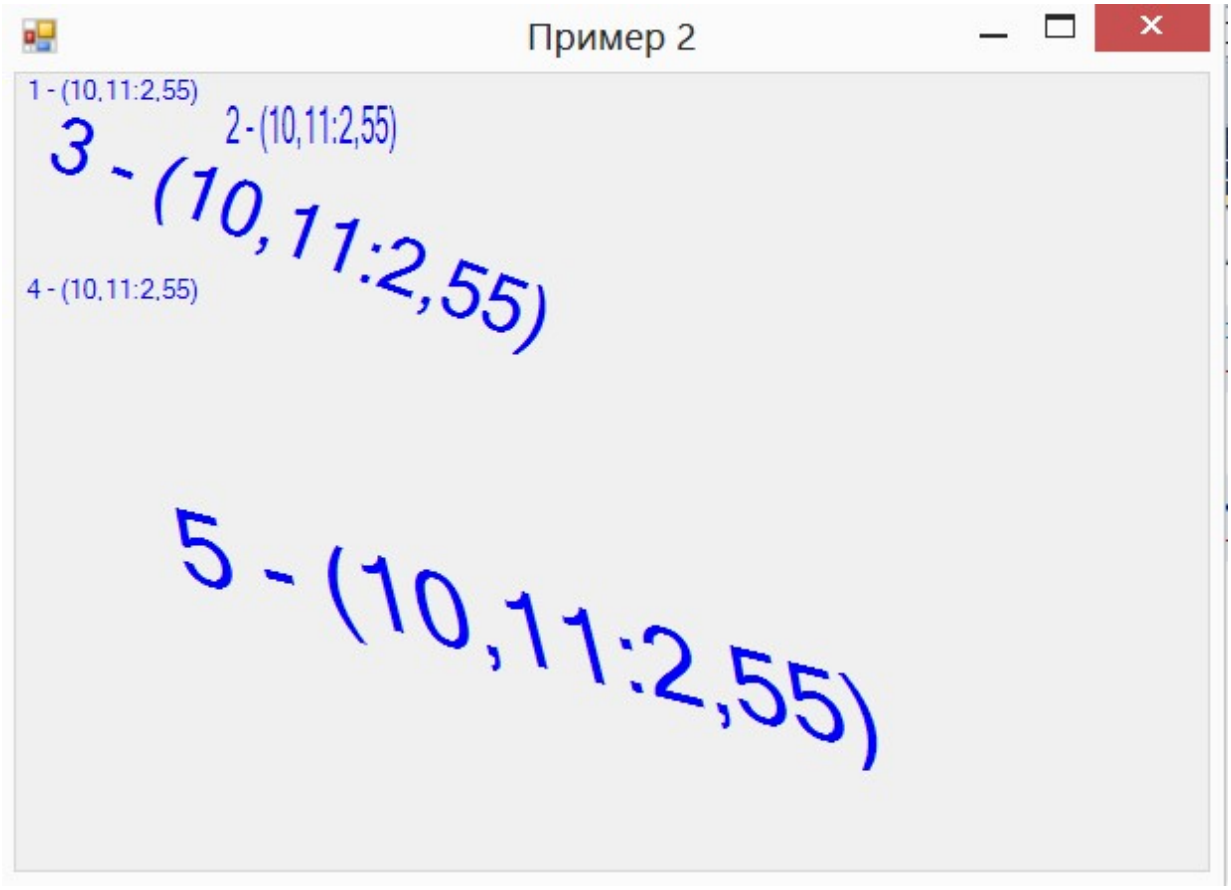


Рис. 43. Результати вирішення прикладу 2

15.4. Продовження роботи з матрицями

У лістингу наводиться програма малювання послідовності цифр по колу навколо центру екрану. Слід звернути увагу на використання функції зберігання матриці:

*GraphicsState gr*1 *= gr.Save();*

Тут запам'ятовується матриця, яку можна відновити у будь-якому місці функції *Form1_Paint*. Відновлюється матриця за допомогою функції *gr.Restore*(*gr*1). Як видно з програми, у циклі кожний раз відновлюється центр екрану, відносно якого здійснюється поворот чисел. Результат виведення чисел наведено на рис. 44.

Лістинг 39. (exam39)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam39
{
    public partial class Form1 : Form
    {
        //360 повинно ділитися на це число.
        const double Degrees = 20;
        Font font;
        public Form1()
        {
            InitializeComponent();
            font = new Font("Arial",15);
            this.ResizeRedraw = true;
        }

      private void Form1_Paint(object sender, PaintEventArgs e)
       {
         Graphics gr = e.Graphics;
         StringFormat strfmt = new StringFormat();
         strfmt.LineAlignment = StringAlignment.Center;
         gr.TranslateTransform(Size.Width / 2, Size.Height / 2);
            for (double i = 0; i < 359; i += Degrees)
            {
                GraphicsState gr1 = gr.Save();
                gr.RotateTransform((float)i);
                gr.TranslateTransform(150, 0);
```

```
                gr.DrawString( i.ToString(), font, Brushes.Blue,
                0, 0, strfmt);
                gr.Restore(gr1);
            }
        }
    }
}
```

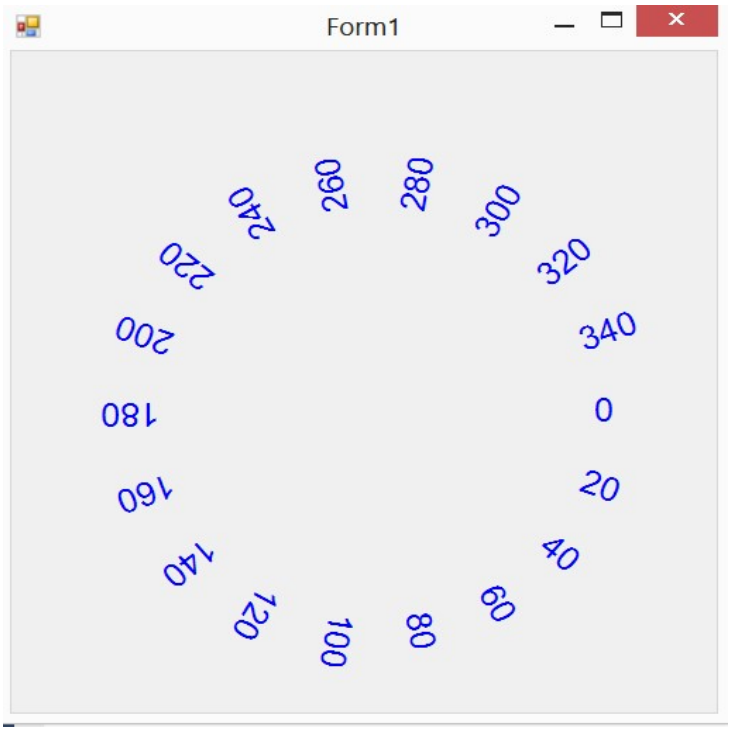


Рис. 44. Виведення чисел по колу

15.5. Імена шрифтів

У лістингу 40 наводиться текст програми виведення імен шрифтів. Частина з них показана на рис. 45.

Лістинг 40. (exam40)

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

namespace exam40
{
    public partial class Form1 : Form
    {
```

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

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics gr = e.Graphics;
            Brush br = new SolidBrush(Color.FromArgb(255,0,0));
            float y = 0, x = 0;
            FontFamily[] aff = FontFamily.Families;
            Font font;
            int n = 0;

               foreach (FontFamily ff in aff)
                {
                    n++;
                    if (ff.IsStyleAvailable(FontStyle.Regular))
                   {
                    font = new Font(ff, 12);
                    gr.DrawString(ff.Name, font, br, x,y);
                    y+= font.GetHeight(gr);
                    font.Dispose();
                   }

                    if (n % 30 == 0) { x += 250; y = 0; }
                }
                      base.OnPaint(e);
    } // Onpaint
  }
}
```

Рис. 45. Фрагмент виведення імен шрифтів

15.5. Згладжування тексту

Windows дозволяє керувати згладжуванням при виведенні тексту на екран. Цю функціональність можна включити, встановив прапорець *Smooth Edges Of Screen Fonts* (Згладжувати нерівності екранних шрифтів) на вкладці *Effects* (Ефекти) діалогового вікна *Display Properties* (Властивості екрану).

Можна також перевизначити користувацьке налаштування, установив властивість *TextRenderingHint* класу *Graphics.* Перелічення *TextRenderingHint* – це перелічення, визначене у просторі імен *System.Drawing.Text*. Значення перелічення наведені у таблиці 18.

Таблиця 18

Перелічення *TextRenderingHint*

Член	Значення	Опис
SystemDefault	0	За умовчанням
SingleBitPerPixelGridFit	1	Без згладжування, з підгонкою під координатну сітку
SingleBitPerPixel	2	Без згладжування, без підгонки під координатну сітку
AntiAlias	3	Із згладжуванням, без підгонки під координатну сітку
AntiAliasGridFit	4	Із згладжуванням, з підгонкою під координатну сітку
ClearTypeGridFit	5	*ClearType* для рідкокристалічних моніторів

У лістингу 41 наводиться текст програми, яка демонструє використання всіх шести значень перелічення. На рис. 46 показаний результат роботи програми.

Лістинг 41 (exam41)

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

namespace exam41
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        protected override void OnPaint(PaintEventArgs e)
        {
          Graphics gr = e.Graphics;
          Brush brush = new SolidBrush(Color.FromArgb(255,0,0));
          Font fon = new Font(Font.Name, 96);
          string str = "3";
          int cxText = (int) gr.MeasureString(str,fon).Width;
          int cyText = (int)gr.MeasureString(str, fon).Height;
            for (int i = 0; i < 6; i++)
          {
             gr.TextRenderingHint = (TextRenderingHint)i;
             gr.DrawString(str, fon, brush, cxText * i, 0);
          }
             for (int i = 0; i < 6; i++)
                 gr.DrawString(str, fon, brush, cxText * i,
                               (int)(0.6*cyText));
                     base.OnPaint(e);
        }
    }
}
```

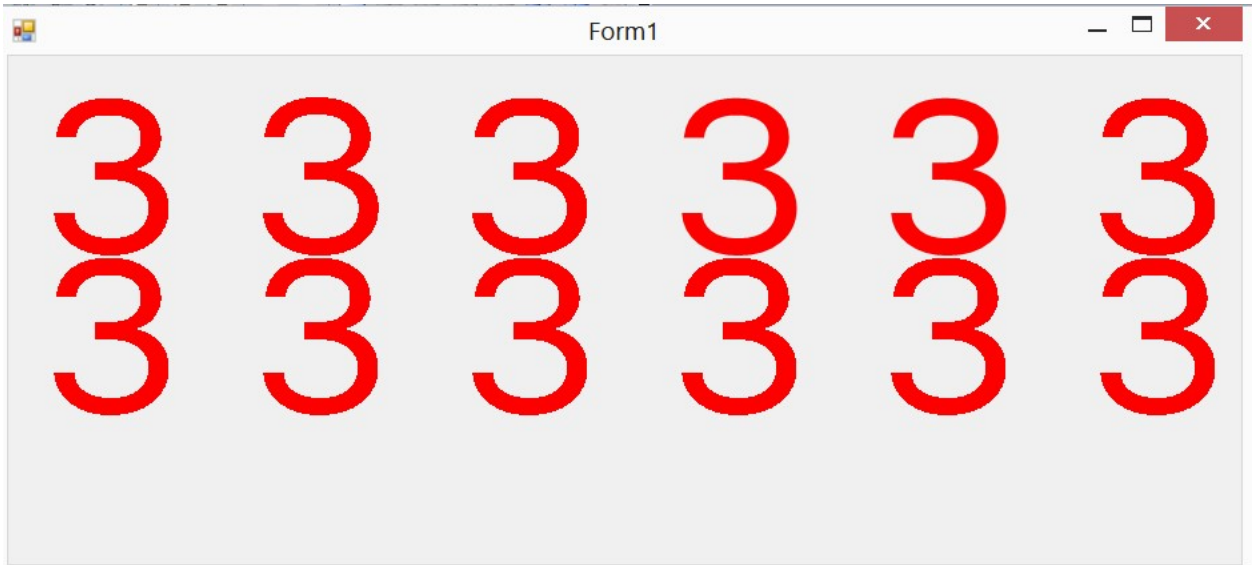


Рис. 46. До згладжування тексту

15.7. Вирівнювання тексту всередині прямокутника

Клас *StrinFormat* інкапсулює відомості про структуру тексту (наприклад, вирівнювання, орієнтація і позиції табуляції), операції з відображенням (такі як вставка багатокрапки і заміна національних цифр) та можливості *OpenType*. Цей клас не спадкується. У роботі використовувались наступні властивості класу *StrinFormat*: *Alignment*, *LineAlignment*, *GenericTypographic*, *FormatFlags*). Для подальшого використання класу і отримання інформації щодо його можливостей наведена у табл. 19.

Таблиця 19

Опис класу StringFormat

Ім’я	Опис
Конструктори	
StringFormat()	Ініціалізує новий об’єкт *StringFormat*.
StringFormat(StringFormat)	Ініціалізує новий об’єкт StringFormat з вказаного існуючого об’єкту *StringFormat*.
StringFormat(StringFormatFlags)	Ініціалізує новий об’єкт *StringFormat* із заданим переліченням *StringFormatFlags*.
StringFormat(StringFormatFlags, Int32)	Ініціалізує новий об’єкт *StringFormat* із заданими переліченням *StringFormatFlags* та мовою.
Властивості	
Alignment	Отримує або задає горизонтальне вирівнювання рядку.
DigitSubstitutionLanguage	Отримує мову, що використовується при заміні арабських цифр місцевими.
DigitSubstitutionMethod	Отримує метод, що використовується для заміни цифр.
FormatFlags	Отримує або задає перелічення *StringFormatFlags*, яке містить відомості щодо форматування.
GenericDefault	Повертає універсальний об’єкт *StringFormat* за умовчанням.
GenericTypographic	Повертає універсальний друкарський об’єкт *StringFormat*.
HotkeyPrefix	Отримує або задає об’єкт *HotkeyPrefix* для цього об’єкту *StringFormat*.
LineAlignment	Отримує або задає вертикальне вирівнювання рядку.
Trimming	Отримує або задає перелічення *StringTrimming* для об’єкту *StringFormat*.
Методи	
Clone	Створює точну копію об’єкту *StringFormat*.
CreateObjRef	Створює об’єкт, який містить всю необхідну інформацію для створення проксі-серверу, що використовується для взаємодії з видаленим об’єктом. (Успадковано від *MarshalByRefObject*.)
Dispose	Звільняє всі ресурси, що використовуються цим об’єктом *StringFormat*.
Equals(Object)	Визначає, чи рівний заданий об’єкт поточному об’єкту. (Успадковано від *Object*.)

Продовження таблиці 19

Ім’я	Опис
GetHashCode	Грає роль хеш-функції для певного типу. (Успадковано від *Object*.)
GetLifetimeService	Витягує об'єкт обслуговування під час існування, який керує політикою часу існування даного екземпляра. (Успадковано від *MarshalByRefObject*.)
GetTabStops	Отримує позиції табуляції для цього об’єкту *StringFormat*.
GetType	Повертає об’єкт *Type* для поточного екземпляру. (Успадковано від *Object*.)
InitializeLifetimeService	Повертає об'єкт обслуговування під час існування для керування політикою часу існування цього екземпляра. (Успадковано від *MarshalByRefObject*.)
SetDigitSubstitution	Вказує мову та метод, які використовуються при заміні арабських цифр місцевими.
SetMeasurableCharacterRanges	Задає масив структур *CharacterRange*, що представляють діапазони знаків, що вимірюються за допомогою виклику методу *MeasureCharacterRanges*.
SetTabStops	Задає позиції табуляції для об’єкту *StringFormat*.
ToString	Перетворює цей об’єкт *StringFormat* у зручний для сприйняття рядок. (Перевизначає *Object::ToString*().)

Достатньо ефективним керуванням виведення тексту всередині прямокутника забезпечується використанням властивості *Trimming*. Ця властивість визначає, як буде закінчуватись останній рядок, використовуючи версію методу *DrawString* з *RectangleF*, коли у прямокутнику не вміщається весь текст. Ця властивість приймає значення одного з членів перелічення *StringTrimming*, які наведені у табл. 20.

Таблиця 20

Перелічення StringTrimming

Член	Значення	Опис
None	0	Якби не було нижньої межі
Character	1	Закінчується знаком.
Word	2	Закінчується цілим словом.
EllipsisCharacter	3	Закінчується знаком з багатокрапкою (...).
EllipsisWord	4	Закінчується словом з багатокрапкою (...).
EllipsisPatb	5	Багатокрапка перед останньою директорією.

Програма на лістингу 42 ілюструє дію цих параметрів. Результат виконання програми показаний на рис. 47.

Лістинг 42 (exam42)

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

namespace exam42
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            this.ResizeRedraw = true;
        }

        protected override void OnPaint(PaintEventArgs e)
        {
          Graphics gr = e.Graphics;
          int cx = ClientSize.Width;
          int cy = ClientSize.Height;
          Pen pen = new Pen(Color.FromArgb(255,0,0));
         //Rectangle rect = new Rectangle(0, 0, cx / 2,cy/ 2);
         Brush br = new SolidBrush(Color.FromArgb(0,0,255));
         float cyText = Font.GetHeight(gr);
         float cyRect = cyText;
         Rectangle rect = new Rectangle(0, 0, cx, (int)cyRect);
          string str =
          "Сегодня хорошая погода " +
          "Поют птицы и мы наслаждаемся их пением " +
          "Мы изучаем C# : работу с выравниванием текста ";
          StringFormat strfmt = new StringFormat();
          strfmt.Trimming = StringTrimming.Character;
          gr.DrawString(str, Font, br, rect, strfmt);
          gr.DrawRectangle(pen, rect);
          rect.Offset(0, (int)(cyRect + cyText));
          strfmt.Trimming = StringTrimming.Word;
          gr.DrawString(str, Font, br, rect, strfmt);
          gr.DrawRectangle(pen, rect);
          rect.Offset(0, (int)(cyRect + cyText));
          strfmt.Trimming = StringTrimming.EllipsisCharacter;
          gr.DrawString(str, Font, br, rect, strfmt);
          gr.DrawRectangle(pen, rect);
          rect.Offset(0, (int)(cyRect + cyText));
```

```
            strfmt.Trimming = StringTrimming.EllipsisWord;
            gr.DrawString(str, Font, br, rect, strfmt);
            gr.DrawRectangle(pen, rect);
            rect.Offset(0, (int)(cyRect + cyText));
            strfmt.Trimming = StringTrimming.EllipsisPath;
            gr.DrawString(str, Font, br, rect, strfmt);
            gr.DrawRectangle(pen, rect);
            rect.Offset(0, (int)(cyRect + cyText));
            strfmt.Trimming = StringTrimming.None;
            gr.DrawString(str, Font, br, rect, strfmt);
            gr.DrawRectangle(pen, rect);
            base.OnPaint(e);
        }
    }
}
```

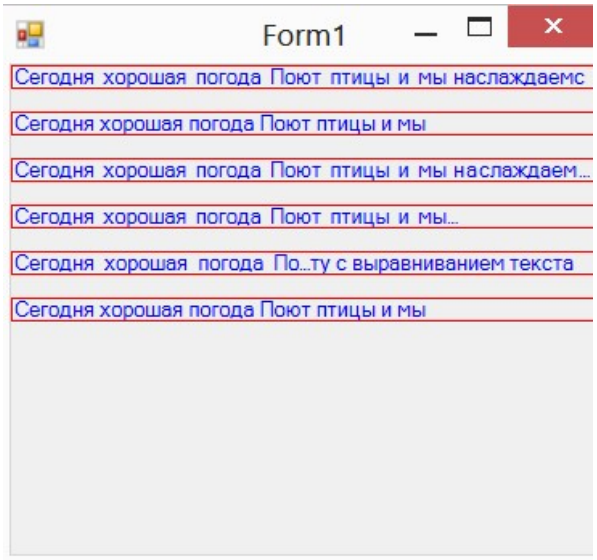


Рис. 47. Вирівнювання тексту всередині прямокутника

15.8. Виведення тексту в колонках

Розміри клієнтської області форми доступні через її властивість *ClientSize* і завжди обчислюються у пікселях. Задавши інше перетворення сторінки, краще виражати розміри клієнтської області над пікселах, а одиницях, що використовуються методів малювання. Існує щонайменше два способи отримання розмірів клієнтської області в метричних одиницях. Перший спосіб – використання властивості *VisibleClipBounds* об'єкту *Graphics.* Вона завжди повертає розміри клієнтської області в одиницях, відповідних поточним значенням властивостей *PageUnii* і *PageScale.* Інший підхід до визначення розмірів області екрану включає використання методу *TramformPoints,* реалізованого у класі *Graphics.*

При наявності масиву структур *PointF,* виражених у одиницях пристрою, можна перерахувати в одиниці сторінки, викликавши метод:

TransformPoints(CoordinateSpace.Page, CoordinateSpace.Device, Point[] aptf);

Ми багаторазово використовували методи *MeasureString* класу *Graphics*, однак не всі методи розглянуті. У нашому випадку, при виведенні тексту в колонках доцільно використання раніше не розглянутого методу. Для узагальнення наведемо усі перевантаження методу *MeasureString*.

SizeF MeasureString(string str, Font font);
SizeF MeasureString(string str, Font font, int iWidth);
SizeF MeasureString(string str, Font font, SizeF sizef);
SizeF MeasureString(string str, Font font, int iWidth,StringFormat strfmt);
SizeF MeasureString(string str, Font font, SizeF sizef,StringFormat strfmt);
SizeF MeasureString(string str, Font font, PointF ptfOrigin,StringFormat strfmt);
SizeF MeasureString(string str, Font font, SizeF sizef,StringFormat strfmt,
out int iCharacters, out int iLines);

Ми вже давно використовуємо першу версію *MeasureString*. Вона повертає ширину і висоту рядка тексту із заданим шрифтом. Властивість *Height* об'єкта, що повертається цим методом *SizeF*, часто дорівнює значенню, отриманому з методу *GetHeight* класу *Font*, а якщо текст включає знаки переведення каретки, *Height* буде кратним *GetHeight*.

Друга версія *MeasureString* включає третій аргумент – ширину тексту. Вона зручна, якщо треба вивести рядок, використовуючи метод *DrawString* з аргументом *RectangleF*, і можливе перенесення рядку. Властивість *Width* об’єкту *SizeF*, що повертається методом *MeasureString*, завжди менше або дорівнює аргументу *iWidth*. Властивість *Height*, поділена на величину *GetHeight*, дає кількість рядків.

Третя версія *MeasureString* має аргумент *SizeF*, що визначає як ширину так і висоту. Якщо його властивість *Width* дорівнює аргументу *iWidth* другої версії *MeasureString* і якщо властивість *Height* достатня для розміщення всіх ліній тексту, що утворюють рядок, ця версія поверне таке ж значення, як і друга. В іншому випадку властивість *Height* об’єкту *SizeF* буде дорівнювати властивості *Height* аргументу *SizeF*, а властивість *Width* об’єкту *SizeF* буде показувати максимальну ширину тексту, який може помістится у прямокутник заданих розмірів.

Четверта, п’ята і шоста версії аналогічні другій і третій за вийнятком того, що вони включають аргумент *StringFormat*. Якщо вам знадобиться аргумент *StringFormat* при виклику *DrawString*, використайте один з цих методів *MeasureString*.

Остання версія *MeasureString* має два аргументи, в яких повертається додаткова інформація. Вони показують кількість знаків і рядків тексту, що відображаються методом *DrawString,* якщо йому передається структура *RectangleF.*

Метод *MeasureString* з цими аргументами дуже зручний, коли для виведення на монітор одного блоку тексту треба безліч викликів *DrawString*. Припустимо, ви хочете використати *DrawString*, щоб роздрукувати на принтері текст, що не вміщується на одній сторінці. *MeasureString* дозволяє ви-

значити розмір тексту, який може вміститися на першій сторінці, і на основі цих даних сформувати другу сторінку. Покажемо, як використовувати цю версію *MeasureString*, у програмі *TextColumns* в кінці глави.

У лістингу 43 наведена програма, яка форматує текст в колонки. Виведення тексту виконується у пунктах. Кожна колонка вимагає виклику метода *DrawString*.

Лістинг 43(exam43)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Text;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace exam43
{
    public partial class Form1 : Form
    {
        int kod;

        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
            Font = new Font(Font.Name, 11);
            kod = 1;
        }


  string str =
  "Сегодня хорошая погода\tСегодня хорошая погода Сегодня хорошая по-
года " +
  "Поют птицы и мы\tнаслаждаемся их пением Поют птицы и мы на-
слаждаемся их пением " +
  "Мы изучаем C# :\tработу с выравниванием текста Поют птицы и
мы наслаждаемся их пением " +
  "Сегодня хорошая\tпогода Сегодня хорошая погода Сегодня хоро-
шая погода " +
  "Поют птицы и мы\tнаслаждаемся их пением Поют птицы и мы на-
слаждаемся их пением " +
  "Мы изучаем C# :\tработу с выравниванием текста Поют птицы и
мы наслаждаемся их пением " +
  "Сегодня хорошая\tпогода Сегодня хорошая погода Сегодня хоро-
шая погода " +
  "Поют птицы и мы\tнаслаждаемся их пением Поют птицы и мы на-
слаждаемся их пением " +
```

```
  "Мы изучаем C# :\tработу с выравниванием текста Поют птицы и
мы наслаждаемся их пением";

 protected override void  OnPaint(PaintEventArgs e)
{
 Graphics gr = e.Graphics;
 Brush br = new SolidBrush(Color.FromArgb(255,0,0));
 int cx,cy,Chars, Lines;
 StringFormat strfmt = new StringFormat();
 // Единицы измерения - пункты.
 cx = ClientSize.Width;
 cy = ClientSize.Height;
 PointF[] aptf = { new PointF(cx, cy) };
 // ------------------------------------------------------------
   string tran = "";
    tran += "    Пикселя x = " + cx + "   y = " + cy + "\n";
    gr.TransformPoints(CoordinateSpace.Device,
                       CoordinateSpace.Page, aptf);
    tran += "    Пикселя x = " + aptf[0].X + "   y = " +
                       aptf[0].Y + "\n";

    gr.PageUnit = GraphicsUnit.Point;

    gr.TransformPoints(CoordinateSpace.Page,
                       CoordinateSpace.Device, aptf);
    tran += "    Point   x = " + aptf[0].X + "   y = " +
                      aptf[0].Y + "\n";

if (kod == 1) MessageBox.Show(tran);
    kod++;
 }
// ------------------------------------------------------------
----------
 float fcx = aptf[0].X;
 float fcy = aptf[0].Y;

 strfmt.Trimming = StringTrimming.Word; //разделение по словам
 // нижняя граница прямоугольника не будет пересекаться
 strfmt.FormatFlags |= StringFormatFlags.NoClip;
 // определение табуляции
 strfmt.SetTabStops(0, new float[]{10});

// Отображение текста.
RectangleF rectf;
String str = this.str;
    for(int x = 0;x<fcx && str.Length>0; x+=156)
     {
      rectf = new RectangleF(x, 0, 144, fcy -
                             Font.GetHeight(gr));
      fcy += Font.GetHeight(gr);
      gr.DrawString(str,Font,br,rectf,strfmt);
      gr.MeasureString(str,Font,rectf.Size,strfmt,
                       out Chars,out Lines);
```

```
        str = str.Substring(Chars);
      }
        base.OnPaint(e);
    }
  }
}
```

У тексті програми використовуються властивості *StringFormat*. Тут *StringTrimmingWord* – щоб у кінці колонок не обрізалися слова, *StringFormatFlagsNoClip* – щоб рядки не обрізалися внизу прямокутника. Встановлена позиція табуляції, рівна 18 пт. Абзац формується у тому випадку, коли у тексті використовується "\t". Цикл оператору for повторюється для кожної колонки. Цикл продовжується до тих пір, поки не досягається край клієнтської області або не виводиться весь текст. Всередині циклу висота прямокутника обчислюється як высота клієнтської області за вирахуванням одного рядка тексту. Метод *DrawString* використовує цей прямокутник. Метод *MeasureString* визначає довжину виведеного тексту методом *DrawString*. Метод *SubString* класу *String* готує рядок для наступної ітерації циклу.

На рис. 48 показано виведення тексту у вигляді колонок – результат виконання програми.

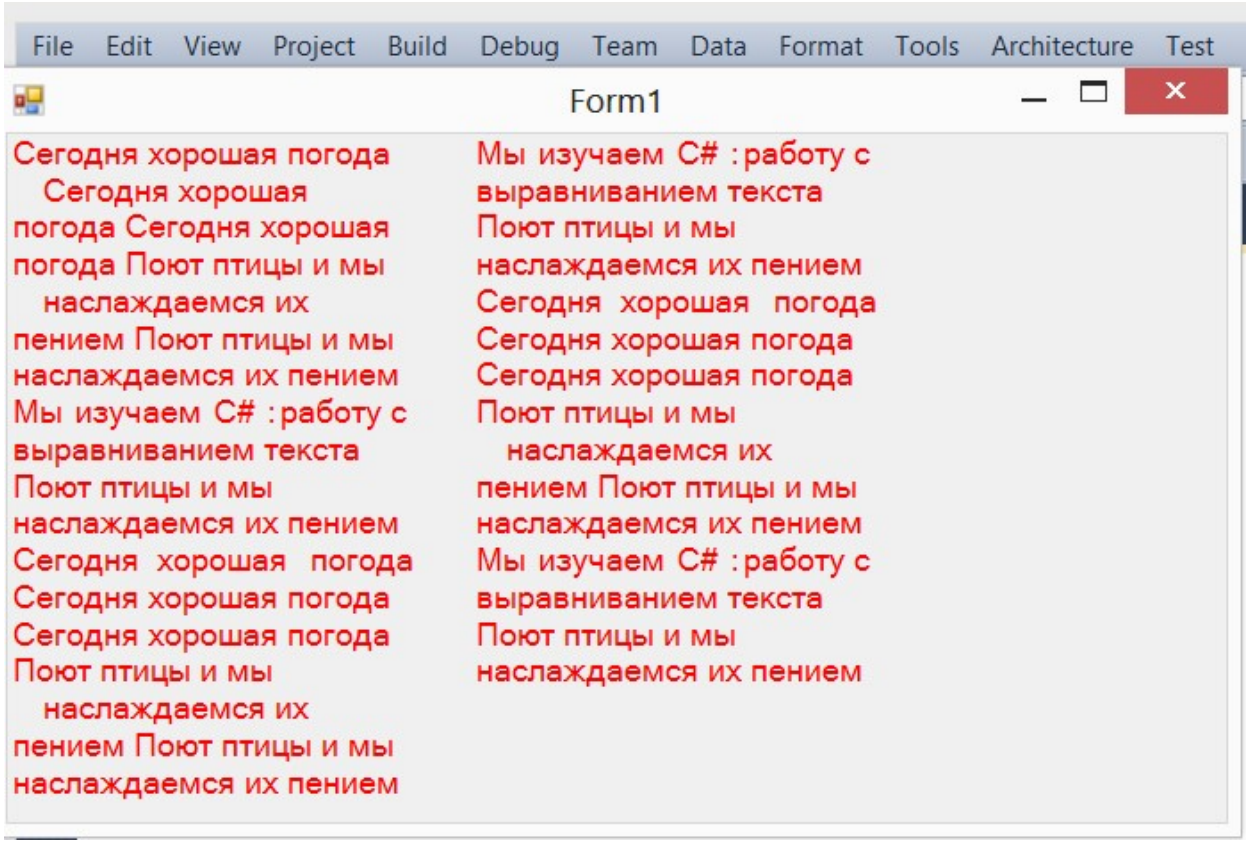


Рис. 48. Виведення тексту у вигляді колонок

16. Робота з таймером

Таймер – це пристрій, який періодично повідомляє програмі про завершення заданого інтервалу часу. Хоча таймер і не настільки важливий пристрій введення, як клавіатура або миша, він все ж таки може бути дуже корисний і знаходить застосування в багатьох додатках *Windows Forms*. Найбільш очевидним додатком для таймера є програма – годинник. Однак є й інші, можливо, не такі очевидні області застосування таймера. Якщо програма повинна виконати великий обсяг роботи, її можна розділити на кілька частин і виконувати кожну частину після отримання події таймера. Таймер підходить для оновлення інформації, що постійно змінюється, наприклад, при виведенні даних про вільні ресурси або проценті виконання будь-якої задачі. Таймер дозволяє періодично зберігати результати роботи програми на диску. Таймер може бути використаний для припинення роботи демо-версій програм.

Об'єкт *Timer* створюється конструктром

Timer timer = new Timer();

Клас містить одну подію *Tick*. Десь у вашому класі слід поставити обробник події для таймера:

```
void TimerOnTick (object obj , EventArgs ea)
{
...
}
```

Відомо, що обробник подій може мати будь-яке ім'я. Далі слід зіставити обробник події створеному раніше об'єкту *Timer.*

timer.Tick += new EventHandler (TimerOntic);

Клас *Timer* включає в себе всього дві властивості:

Interval – інтервал у мілісекундах;
Enabled – для включення таймеру встановлюється в *true*.

Керувати роботою таймеру дозволяють методи *Start i Stopt*.

Замість *Enabled* можна використати виклик методу *Start* – еквівалентний встановленню властивості *Enabled* значення *true,* а методу *Stop* – встановленню властивості *Enabled* значення *false.*

У лістингу 44 наводиться програма використання двох таймерів. Один з них (*timer*) створений без використання елементу управління. Другий (*timer*1) створений з використанням елементу управління (див. рис. 49). По відгукам першого таймеру виводяться прямокутники, координати яких формуються випадковим чином в межах клієнтського вікна. При цьому прямокутники зафарбовуються різними кольорами. Другий таймер забезпечує виведення послідовно наростаючих чисел у *Caption* клієнтського вікна. Результат виконання програми ілюструється на рис. 50.

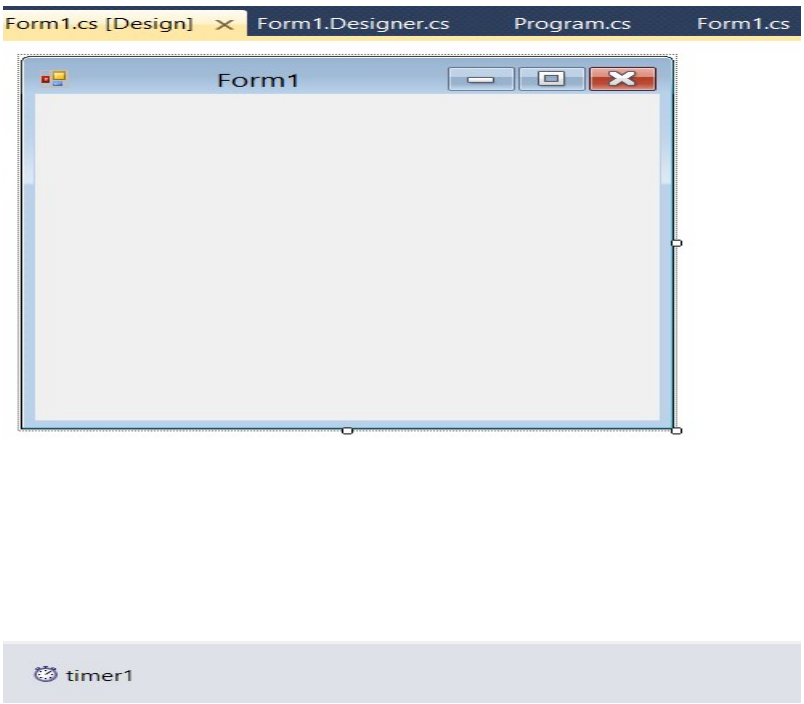


Рис. 49. Використання елементу управління *Timer*

Лістинг 44(exam44)

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

namespace exam44
{
    public partial class Form1 : Form
    {
        int i;
        public Form1()
        {

            InitializeComponent();
            Text = "Random Rectangle";
            Timer timer = new Timer();
            timer.Interval = 1000;
            timer.Tick += new EventHandler(TimerOntic);
            timer.Start();
            timer1.Interval = 20;
            timer1.Start();
     }
```

```
        private void timer1_Tick_1(object sender, EventArgs e)
         {
             Text = (i++).ToString();
             if (i == 100) i = 0;
         }

  void  TimerOntic (object sender, EventArgs e)
  {
   Random rand = new Random();
   int x1 = rand.Next(ClientSize.Width);
   int x2 = rand.Next(ClientSize.Width);
   int y1 = rand.Next(ClientSize.Height);
   int y2 = rand.Next(ClientSize.Height);
   Color color =
   Color.FromArgb(rand.Next(256),rand.Next(256),rand.Next(256));
   Graphics gr = CreateGraphics();
        gr.FillRectangle(new SolidBrush(color),
                         Math.Min(x1,x2),Math.Min(y1,y2),
                         Math.Abs(x1-x2),Math.Abs(y1-y2) );
        gr.Dispose();
       }
    }
}
```

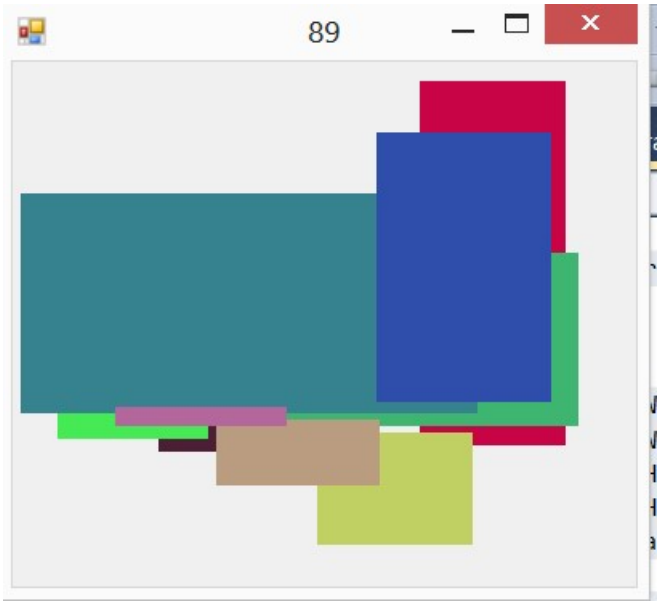


Рис. 50. До використання елементу управління *Timer*

17. Структура DateTime

DateTime призначена для зберігання і обробки інформації дати та часу. Створити об’єкт типу *DateTime* можна, використовуючи сім конструкторів, три з яких наступні:

Конструктори *DateTime*:

DateTime(int year, int month, int day)
DateTime(int year, int month, int day, int hour, int minute, int second)
DateTime{int year, int month, int day, int hour, int minute, int second, int msec)

Значення року може лежати в діапазоні від 1 до 9999, місяця – від 1 до 12, дня – від 1 до числа днів у відповідному місяці цього року. Значення кількості годин лежить у межах від 0 до 23, хвилин і секунд – від 0 до 59, мілісекунд від 0 до 999. Якщо будь-який з аргументів виходить за дозволені межі, конструктор генерує виняток.

Крім того, конструктор *DateTime* генерує виняток у тому випадку, якщо значення року, місяця та дня несумісні. Наприклад, значення місяця, що дорівнює 2, і дня – 29, допустимо тільки для високосного року. Конструктори *DateTime* застосовують правила високосного року для Григоріанського календаря (введеного Папою Григорієм XIII в 1582 р. і розійшовся по всьому світу в наступні століття). У Григоріанському календарі рік є високосним, якщо число, що означає цей рік, ділиться на 4, але не ділиться на 100 або якщо воно ділиться на 400. 1900 не є високосним; 2000 – є.

У табл. 21 описані деякі властивості структури *DateTime*. Всі вони визначені з модифікатором *public*.

Таблиця 21

Деякі властивості *DateTime*

Член	Опис
Date	Повертає дату
Day	Повертає день
Month	Повертає місяць
Year	Повертає рік
DayOfWeek	Повертає день тижня
DayOfYear	Повертає номер дня у році
Hour	Повертає години
Minute	Повертає хвилини
Second	Повертає секунди
Millisecond	Повертає мілісекунди
Ticks	Повертає тіки
Now	Повертає поточні дату та час
Today	Повертає поточну дату
Ticks	Повертає кількість тіків
ToLongDateString	Повертає дату з назвою місяця
ToShortDateString	Повертає дату з номером місяця
ToLongTimeString	Повертає час із секундами
ToShortTimeString	Повертає час без секунд

Для перетворення дати/часу в рядок у типу *DateTime* є 4 перевизначення методу *ToString*:

ToString(string format, IFormatProvider provider);
ToString(IFormatProvider provider);
ToString(string format);
ToString(),

де *format* – рядок форматування дати/часу. Рядок *format* може складатися з 1 символу, тоді він вважається стандартним форматом дати часу, або з кількох, тоді він вважається рядком настроюваного формату дати і часу.

Існують наступні стандартні формати дати та часу:

d – короткий шаблон дати.
D – повний шаблон дати.
f – повний шаблон дати та часу (короткий шаблон часу).
F – повний шаблон дати та часу (повний шаблон часу).
g – загальний шаблон дати та часу (короткий шаблон часу).
G – загальний шаблон дати та часу (повний шаблон часу).
M, *m* – шаблон днів місяця.
O, *o* – шаблон зворотного перетворення дати та часу.
s – сортований шаблон часу і дати.
t – короткий шаблон часу.
T – повний шаблон часу.
u – універсальний сортований шаблон часу і дати.
U – універсальний повний шаблон дати та часу.
Y, *y* – шаблон місяця року.

Для того щоб отримати нестандартний рядок часу, наприклад у такому вигляді 07:27:15, параметр *format* повинен бути рядком настроюваного формату дати та часу, і мати значення "*hh*:*mm*:*ss*". У параметрі *format* допустимі наступні специфікатори:

d – день місяця від 1 до 31;
dd – день місяця від 01 до 31;
ddd – день тижня, скорочено (Пн, Вт і т.д.);
dddd – день тижня, повністю (понеділок і т.д.);
h – години від 1 до 12;
hh – години від 01 до 12;
H – години від 0 до 23;
HH – години від 00 до 23;
m – хвилини від 0 до 59;
mm – хвилини від 00 до 59;
M – місяць від 1 до 12;
MM – місяць від 01 до 12;
MMM – місяць, скорочено;
MMMM – місяць, повністю;
s – секунди від 0 до 59;
ss – секунди від 00 до 59;
y – рік від 0 до 99;
yy – рік від 00 до 99;
yyy – рік від 000 до 999;
yyyy – рік від 0000 до 9999;

yyyyy – рік від 00000 до 99999;
t – перший символ від AM/PM;
tt – AM/PM;
z – зміщення у годинах від UTC (всесвітнього координатного часу, приблизно збігається з часом за Грінвічем);
zz – зміщення у годинах від UTC, з нулями на початку для значень однієї цифри;
zzz – зміщення у годинах та хвилинах від UTC, наприклад 6/15/2009 1:45:30 PM -07:00 -> -07:00;
g або *gg* – епоха або ера;
f, *ff*, *fff*, *ffff*, *fffff*, *ffffff*, *fffffff* – долі секунди;
F, *FF*, *FFF*, *FFFF*, *FFFFF*, *FFFFFF*, *FFFFFFF* – долі секунди з нулями на початку;
: – роздільник компонентів часу;
/ – роздільник компонентів дати;
"*string*" або '*string*' – будь-який рядок, копіюється в результат як є;
% – відокремлює символ специфікатора від символу, що стоїть перед ним. Наприклад "*dd*" для 8 березня поверне "08", а "d%d" поверне 88;
\ – *escape*-символ, скасовує дію специфікатора.

Наведений матеріал дає можливість легко зрозуміти програму виведення дати та часу з максимальними розмірами шрифту щодо клієнтської частини форми. Програму наведено на лістингу 45. Результат рішення наводиться на рис. 51.

Лістинг 45(exam45)

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

namespace exam45
{
    public partial class Form1 : Form
    {
        public Form1()
        {
           InitializeComponent();

           Text = "Simple Clock";
           BackColor = SystemColors.Window;
```

```
            ForeColor = SystemColors.WindowText;
            timer1.Interval = 1000;
            timer1.Start();
            ResizeRedraw = true;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
             Invalidate();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics gr = e.Graphics;
            string strTime = DateTime.Now.ToString("T");
            SizeF sizef = gr.MeasureString(strTime, Font);
            float fScale = Math.Min(ClientSize.Width/sizef.Width,
            ClientSize.Height / sizef.Height);
            Font font = new Font(Font.FontFamily, fScale *
                                 Font.SizeInPoints);
             sizef = gr.MeasureString(strTime, font);
             gr.DrawString(strTime, font, new
              SolidBrush(ForeColor),
             (ClientSize.Width - sizef.Width ) / 2,
             (ClientSize. Height - sizef.Height) / 2);
             base.OnPaint(e);
        }
    }
}
```

Рис. 51. Приклад використання структури *DateTime*

18. Завантаження і виведення зображення

18.1. Виведення зображення з файлу та Інтернету

Якщо треба лише завантажувати та виводити растрові зображення, клас *Image* містить все необхідне для цього. Клас *Image* розташований у просторі імен *System.Drawing*. Даний клас є абстрактним класом, який надає функціональні можливості для похідних класів *Bitmap* і *Metafile*.

Формати файлів, підтримувані класом *Image,* вказуються у статичних властивостях класу *ImageFormat*, визначеного у просторі імен *System.Drawing.Imaging*: bmp, MemoryBmp, Icon, Gif, Jpeg, Png, Tiff, Exif, Wmf і Emf.

Клас *Image* має 4 статичні методи, які повертають об'єкти типу *Image* і необхідні для завантаження бітової карти або метафайлу з файлу або потоку:

1. *public static Image FromFile(string filename);*
//*string filename* – шлях до файлу, який завантажується

2. *public static Image FromFile(string filename, bool useEmbeddedColorManagement);*
// *useEmbeddedColorManagement* – параметр, який вказує чи потрібно використовувати інформацію щодо налаштувань кольору

3. *public static Image FromStream(Stream stream);*
// *Stream stream* – поток, що завантажується

4. *public static Image FromStream(Stream stream, bool useEmbeddedColorManagement);*

Статичний *метод Image FromStream* корисний при отриманні потоку від джерела, що відрізняється від файлової системи. Наприклад, метод *FromStream* дозволяє завантажити зображення з Інтернету.

Клас *Image* містить властивості, серед яких є властивості, що вказують розмір зображення у пікселях.

```
public Size Size {get;}
public int Height {get;}
public int Width {get;}
```

Об'єкт *Image* можна вивести на екран або принтер за допомогою методу *DrawImage* класу *Graphics*.

```
public void DrawImage(
  Image image,
  Point point
);

public void DrawImage(
  Image image,
```

```
//масив з 3 структур Point, що визначають паралелограм.
  Point[] destPoints
);

public void DrawImage(
Image image,
PointF point );
public void DrawImage(
Image image,
// Rectangle rect – визначення розташування і розміру зображення
Rectangle rect) ;
```

У наступному прикладі, код призначений для роботи з *Windows Forms*. Обробник для події *Paint*. Об'єкт *Graphics* передається у подію та використовується для малювання зображення у формі.

Код виконує наступні дії:

- Створює малюнок із файлу з ім'ям *SampImag.jpg*. Цей файл повинен знаходитись у тій же папці, що і виконуємий файл додатку.
- Створює точку, в якій буде розміщуватися верхній лівий кут зображення.

```
protected override void OnPaint(PaintEventArgs e)
{
      Image newImage = Image.FromFile( "SampImag.jpg" );
      Point ulCorner = Point(100,100);
      e->Graphics->DrawImage( newImage, ulCorner );
}
```

У лістингу 46 наводиться приклад завантаження зображення із файлу та Інтернету. Зокрема, наводиться завантаження зображення з Інтернету при використанні методу *FromStream*. Завантаження з файлу при використанні методу *FromFile* закоментовано.

Лістинг 46 (exam46)

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

namespace exam46
```

```
{
    public partial class Form1 : Form
    {
        Image image;
        public Form1()
        {
            InitializeComponent();
            string strURL =
   "http://www.kryvyirih.dp.ua/ua/osximage/name/251109545166015_n_8o";

             string strFile = "1.jpg";
            WebRequest webreq = WebRequest.Create(strURL);
            WebResponse webres = webreq.GetResponse();
            Stream stream = webres.GetResponseStream();
            image = Image.FromStream(stream);
            //image = Image.FromFile(strFile);
            ResizeRedraw = true;
            stream.Close();
          }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            Graphics gr = e.Graphics;
            float cx = image.Width;
            float cy = image.Height;
            Rectangle rect = ClientRectangle;
            gr.DrawImage(image, (rect.Width - cx) / 2,
              (rect.Height - cy) / 2,image.Width, image.Height);
            gr.DrawLine(new Pen(Color.FromArgb(255, 0, 0)), 0,
              rect.Height / 2, rect.Width, rect.Height / 2);
        }
    }
}
```

Виведення зображення наведено на рис. 52.

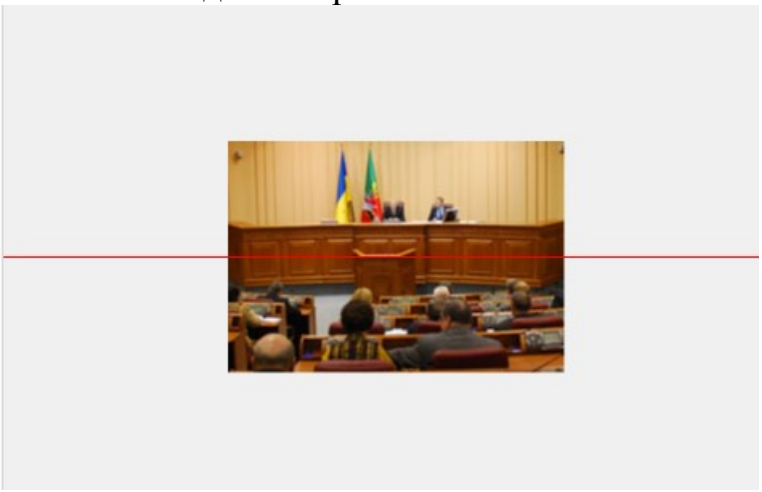


Рис. 52. Виведення зображення

Оператори, що використовують класи *WebRequest* і *WebResponse*, демонструють стандартний підхід до завантаження файлів із *Web*. В цій програмі метод *GetResponseStream* класу *WebResponse* отримує доступний для читання потік JPEG-файла. На даному етапі потік можна просто передати методу *FromStream*(*stream*). Для оцінки центрування зображення проведена горизонтальна лінія.

18.2. Керування розташуванням зображення відносно клієнтського вікна

Версії DrawImage, орієнтовані на прямокутник, можуть не тільки масштабувати зображення, але й дещо інше. Якщо вказати від'ємну ширину, картинка буде повернута по вертикальній вісі, і ви отримаєте її зеркальну копію. При від'ємній висоті метод повертає її по горизонтальній вісі і відображає догори ногами. У будь-якому випадку верхній лівий кут вихідного, неперевернутого зображення позиціонується відповідно координатам *Point* і *PointF* прямокутника, що вказані у *DrawImage*. Програма на лістингу 47 виводить чотири зображення (див. рис. 53). Крім пешого зображення, інші мають від'ємну ширину або висоту. У всіх чотирьох випадках другий і третій аргументи методу *Draw Image* вказують центр клієнтської області.

Лістинг 47 (exam47)

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

namespace exam47
{

    public partial class Form1 : Form
    {
        Image image;
        public Form1()
        {
            InitializeComponent();
            this.Size = new Size(1000, 300);
            Text = this.Size.ToString();
            string strURL =
  "http://www.kryvyirih.dp.ua/ua/osximage/name/251109545166015_n_8o";

            string strFile = "1.jpg";
```

```
            //WebRequest webreq = WebRequest.Create(strURL);
            //WebResponse webres = webreq.GetResponse();
            //Stream stream = webres.GetResponseStream();
            //image = Image.FromStream(stream);

            image = Image.FromFile(strFile);
            ResizeRedraw = true;
            //stream.Close();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            Graphics gr = e.Graphics;
            int cx_im = image. Width;
            int cy_im = image.Height;
            int wth = 10;
            gr.DrawImage(image,10,10,cx_im, cy_im);
            wth+=2*cx_im + 10;
            gr.DrawImage(image, wth, 10, -cx_im, cy_im);
            wth += 10;

            gr.DrawImage(image, wth, cy_im+10, cx_im, -cy_im);
            wth += 2*cx_im + 10;

            gr.DrawImage(image, wth, cy_im + 10, -cx_im, -cy_im);
        }
    }
}
```

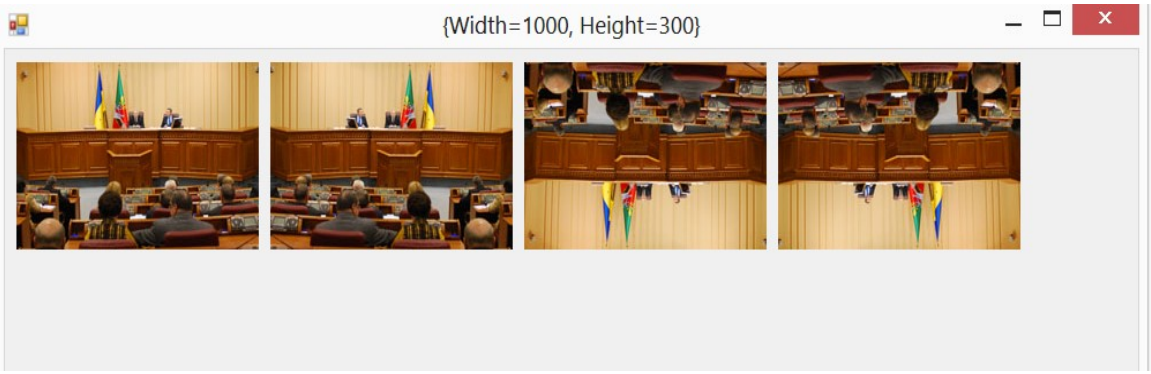


Рис. 53. До керування розташуванням зображення

18.3. Зміна розміру зображення

Клас *Image* включає декілька додаткових методів, надаючих обмежені можливості зберігання і маніпуляції зображеннями.

Методи *Save* класу *Image*(вибірково):

void Save(strin g strFilename, ImageFormat if)
void Save(Stream stream , imageFormat if)

Метод *Save* не працює з об'єктами *Image*, завантаженими з метафайлу або що зберігаються у бітовій карті в пам’яті. Крім того, не можна зберегти зображення у форматі метафайлу або бітової картки в пам’яті. Наступні два методи надають можливості зміни розміру, а також обертання та повороту зображення відповідно.

Методи класу *Image*(вибірково):

Image GetThumbnailImage(int cx, int cy,
Image.GetThumbnaillmageAbort gtia, IntPtr pData);
void RotateFlip(RotateFliptype rft).

Метод *GetTbumbnaillmage* створює піктограми – зменшені зображення, що дозволяють додаткам представляти їх вміст користувачеві, заощаджуючи час і місце. Тим не менш, метод *GetTbumbnaillmage* фактично є універсальною функцією зміни розміру зображення. Зображення можна як збільшити, так і зменшити. Останні два аргументи вказують на функцію зворотного виклику, але їм можна задати значення *null* і 0, що не позначиться на роботі методу. Програма в лістингу 48 створює піктограму, що розміщується в квадраті зі стороною 64 пікселі. Метод *OnPaint()* заповнює (див. рис. 54) всю клієнтську область піктограмами.

Лістинг 48 (exam48)

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

namespace exam48
{
public partial class Form1 : Form
{
    const int isq = 64;
    Image image1;


  public Form1()
 {
     InitializeComponent();
      ResizeRedraw = true;
     Image image2 = Image.FromFile("1.jpg");
     int cx, cy;

     if (image2.Width > image2.Height)
     {
         cx = isq;
```

```
            cy = cx* image2.Height / image2. Width;

        }
        else
        {
            cy = isq;
            cx = cy *image2. Width/image2.Height;
        }
         image1 = image2.GetThumbnailImage(cx,cy,null,(IntPtr)0);
}

protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    int cxx = ClientSize.Width;
    int cyy = ClientSize.Height;
    for (int y = 0; y < cyy; y += isq)
    for (int x = 0; x < cxx; x += isq)
    e.Graphics.DrawImage(image1,
    x + (isq - image1.Width) / 2,
    y + (isq - image1.Height) / 2,
    image1.Width, image1.Height);
  }
 }
}
```

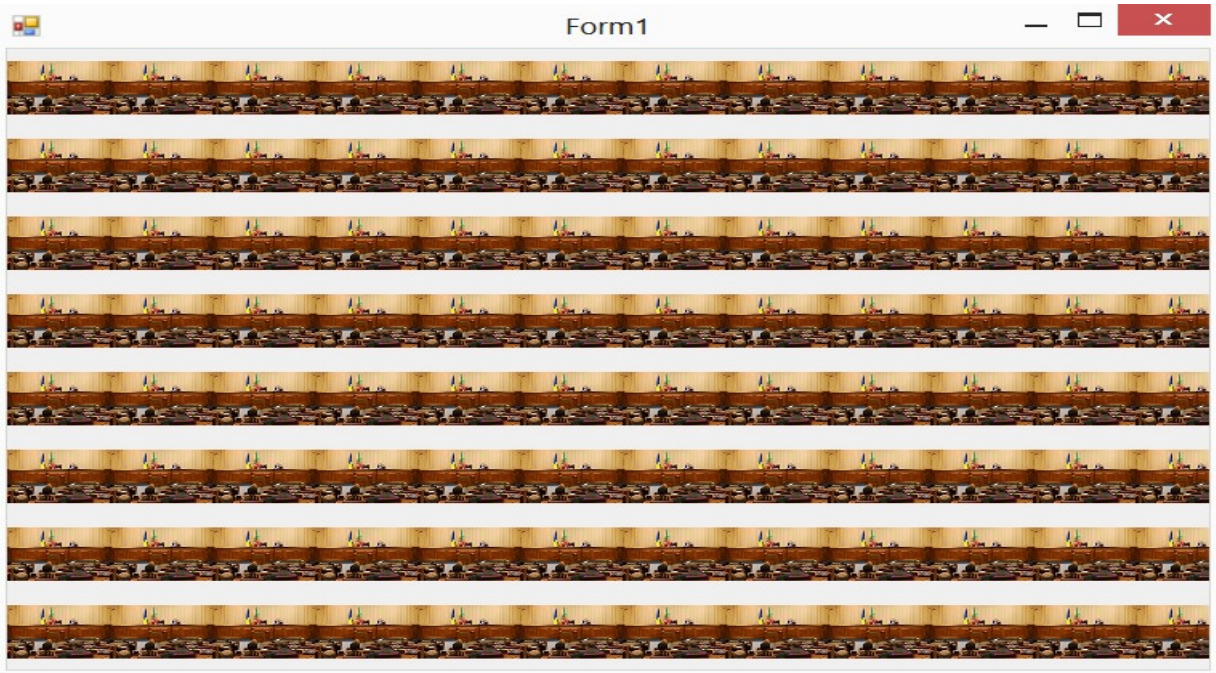


Рис. 54. До зміни розміру зображення

18.4. Класс Bitmap

Ми обговорювали роботу з об'єктами типу *Image*. Простір імен *System.Drawing* також включає клас *Bitmap*, успадкований від *Image*. Всі властивості *Image* застосовні і до *Bitmap*. Будь-які операції, допустимі у відношенні *Image*, допустимі щодо *Bitmap*. Клас *Bitmap* дозволяє працювати прямо з бітами карти. У класу *Image* немає конструкторів, а в *Bitmap* їх 12. Наступні конструктори завантажують об'єкт *Bitmap* з файлу, потоку чи ресурсу.

Bitmap(string strFilename);
Bitmap(string strFilename, bool bUselmageColorManagement);
Bitmap(Stream stream);
Bitmap(Stream stream, bool bUselmageColorManagement);
Bitmap(Type type, string strResource).

Перші чотири конструктори дублюють статичні методи *FromFile* і *FromStream* класу *Image*. П'ятий конструктор завантажує об'єкт *Bitmap* із ресурсів, зазвичай вбудованих до *.exe*-файлів додатку. Далі наведено набір конструкторів, які створюють нові об'єкти *Bitmap* на основі існуючих об'єктів *Image*.

Bitmap(Image image)
Bitmap(Image image, Size size)
Bitmap(Image image, int cx, int cy)

Хоча перший аргумент цих конструкторів визначено як *Image*, він може бути й іншим об'єктом *Bitmap*. Перший конструктор аналогічний методу *Clone* класу *Image* – він створює ідентичний вихідному новий об'єкт *Bitmap*. Другий і третій схожі на метод *GetThumbnailImage* – вони змінюють розмір зображення. У всіх випадках нова бітова карта успадковує формат пікселів вихідної карти, а роздільна здатність нової картки встановлюється рівною роздільній здатності дисплея. Останні чотири конструктори не мають аналогів у класі *Image*. Вони створюють нові об'єкти *Bitmap* з порожніми зображеннями:

Bitmap(int cx, int cy)
Bitmap(int cx, int cy, PixelFormat pf)
Bitmap(int cx, int cy, Graphics grfx)
Bitmap(int cx, int cy, int cxRowBytes, PixelFormat pf, IntPtr pBytes)

Перші три ініціалізують пікселі значенням 0, яке у різних форматах бітових карток має різний сенс. У бітових *RGB*-картах 0 відповідає чорному кольору. В *ARGB*-картах 0 означає прозорість. Четвертий може приймати покажчик на масив байтів, ініціалізує растрове зображення.

Перший конструктор створює об'єкт *Bitmap* заданого розміру з форматом пікселів *PixelFormatormat32bppArgb*. Горизонтальна і вертикальна роздільна здатність задаються рівними роздільній здатності дисплея.

Другий конструктор дозволяє вказати елемент переліченння *PixelFormat*, якщо не задовольняє формат *PixelFormatformat32bppArgb*.

Третій конструктор дозволяє вказати об'єкт *Graphics*. Незалежно від того, пов'язаний зазначений об'єкт з дисплеєм або принтером, і незалежно від того, чи кольоровий принтер. Цей конструктор завжди створює об'єкт *Bitmap* із форматом пікселів *PixelFormatformat32bppArgb*.

Якщо створюються бітові карти, дозволи яких не відповідають ні екранному дозволу, ні дозволу принтера, можна змінити дозвіл завантаженої або створеної бітової картки за допомогою методу *SetResolution* класу *Bitmap*:

void SetResolution(float xDpi, float yDpi);

Для малювання на поверхні бітової карти необхідно створити об'єкт Graphics для даної карти і малювати на ній, як на звичайному графічному пристрої. У програмі на лістингу 49 створюється бітова карта і на ній виводиться текст заввишки 72 пункти. Потім бітова карта виводиться в клієнтській області та (не обов'язково) на сторінці принтера. Результат наведено на рис. 55.

Лістинг 49(exam49)

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

namespace exam49
{
    public partial class Form1 : Form
    {
        const float fResol = 300;
        Bitmap bitmap;

        public Form1()
        {
            InitializeComponent();
            ResizeRedraw = true;
            Size = new Size(600, 300);
            Text = "Hello, World!";
            bitmap = new Bitmap(3,3);
                 //bitmap.SetResolution(fResol,fResol);
            Graphics gr = Graphics.FromImage(bitmap);
            Font font = new Font("Times New Roman", 72);
            Size size = gr.MeasureString(Text, font).ToSize();
            bitmap = new Bitmap(bitmap,size);
                 //bitmap.SetResolution(fResol,fResol);
```

```
            gr.Dispose();
            gr = Graphics.FromImage(bitmap);
            gr.Clear(Color.White);
            gr.DrawString(Text, font, Brushes.Black, 0, 0);
            gr.Dispose();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint(e);
            e.Graphics.DrawImage(bitmap, 100, 0);
        }
    }
}
```

Рис. 55. Створення бітової карти

18.5. Прикріплення іконки до форми

Найчастіше корисно зберігати невеликі двійкові файли (зокрема, бітові карти, значки та нестандартні курсори) прямо в *exe*-файлі додатку. Таким чином вони не зможуть загубитися. Файли, що зберігаються у виконуваному файлі, називаються ресурсами. Найпростіший спосіб прикріплення іконки як ресурсу до форми виконується встановленням властивості *icon*. У цій властивості можна задати іконку для форми. Тут також є кнопка, яка викликає стандартне вікно для відкриття файлу. Завантажена значок потрапить в *resx*-файл форми. Крім того, у редакторі ресурсів можна створити файл **.ico*. Крім *resx*-файлу, можна підключити іконку в конструкторі форми. Це показано у трьох варіантах на лістингу 50.

Лістинг 50 (exam50)

```
using System;
using System.Collections.Generic;
```

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

namespace exam50
{
    public partial class Form1: Form
    {
        public Form1()
        {
           InitializeComponent();
//     Варіант 1
//створимо об’єкт іконки та зв’яжемо його з формою
//у конструктор необхідно передати рядок з іменем файлу іконки
             // Icon icon = new Icon("icon_name.ico");
             // this.Icon = icon;
//     Варіант 2
//     Icon = exam52.Properties.Resources.Icon1;

//     Варіант 3

   // Icon = (Icon)exam52.Properties.Resources.ResourceManager.
   // GetObject("Icon1");

     }
   }
}
```

СПИСОК ЛІТЕРАТУРИ

1. Зеленський О.С., Лисенко В.С. Розробка програмного забезпечення на мові C#.
2. Частина 1. Навчальний посібник. - Кривий Ріг: Криворізький економічний інститут Державного вищого навчального закладу "Криворізький національний університет", 2012.- 160 с.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1./Пер. с англ. - М.: Издательско-торговый дом «Русская Редакция», 2002. – 576 с.