Лабораторна робота 2. Розробка компонентів Windows Forms.

Мета роботи: ознайомитись з елементами технології створення компонентів Win Forms.

План

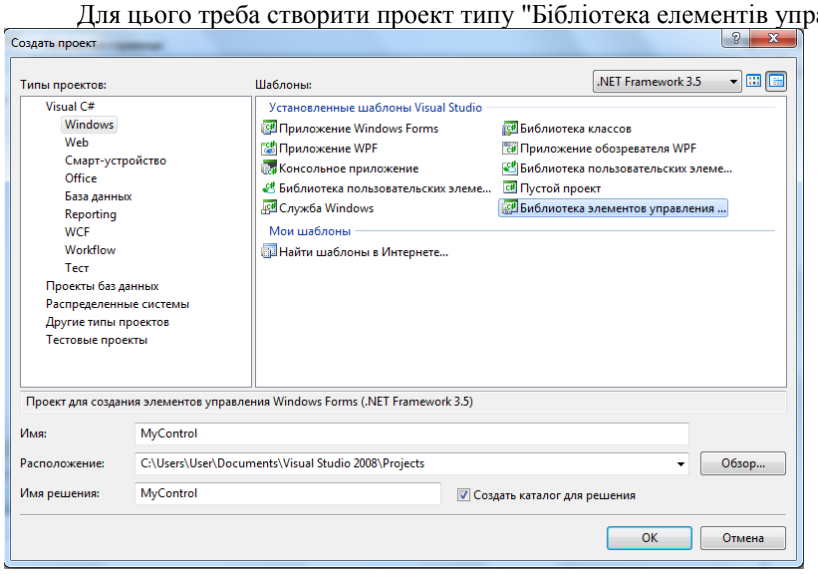
1. Розробити власний компонент.
2. Розмістити компонент на панелі елементів.
3. Створити подію для розробленого компонента.
4. Розширити функціональність компонента в режимі розробки: смарт-теги.
5. Створити програму з використанням розробленого компонента.

Розглянемо приклад розробки компонента – нащадка стандартного компонента Label з двома унікальними властивостями – початковим та кінцевим кольорами `StartColor` та `EndColor`.

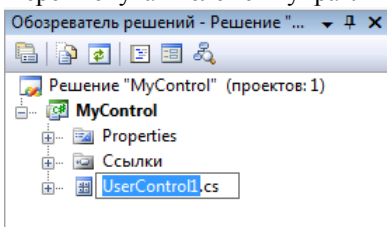
Порядок виконання роботи

Розробимо компонент `GradientLabel`, який є нащадком компонента Label. Колір компонента буде лінійно змінюватись від `StartColor` до `EndColor`.

Для цього треба створити проект типу "Бібліотека елементів управління".



Перейменувати елемент управління – на `GradientLabel`.



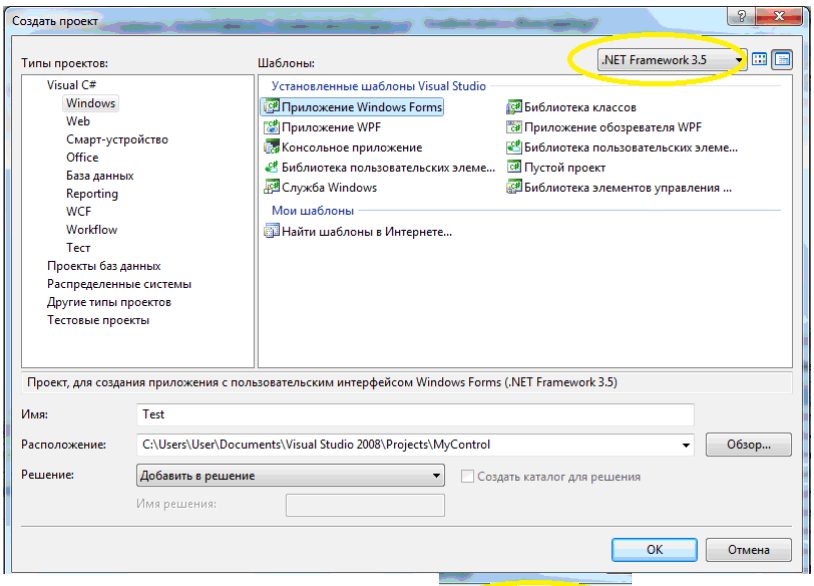
Компонент `GradientLabel` успадкуємо від `Label`.

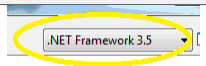
Видалимо властивість `AutoScaleMode`, якої у компонента `Label` немає

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

Компіляція приводить до створення компонента.

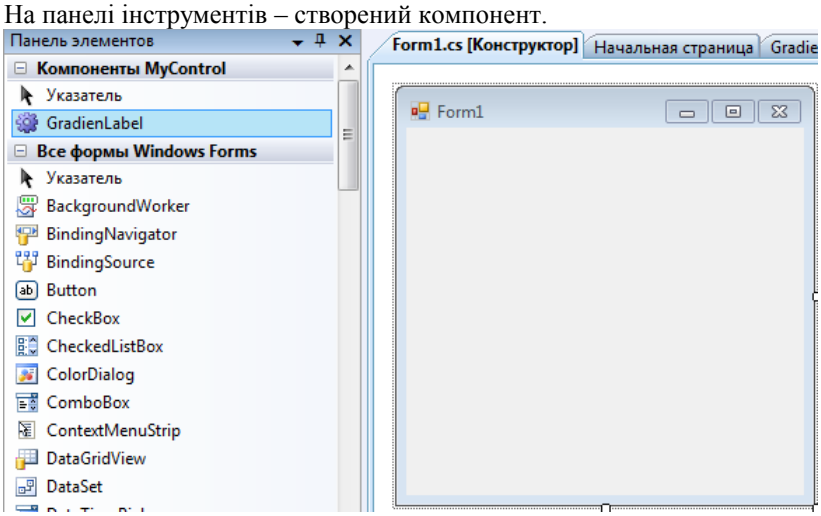
Створимо застосування Windows Forms.



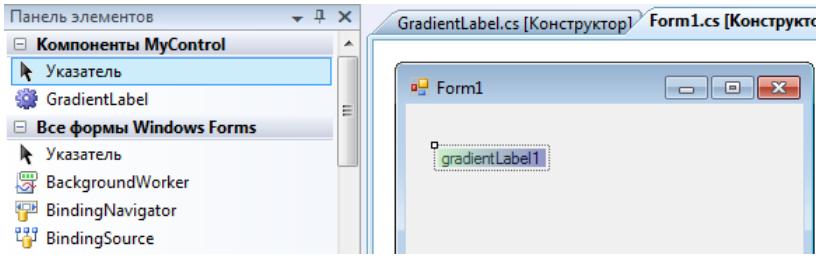


Зверніть увагу на цільову платформу: ,

На панелі інструментів – створений компонент.



Він може бути розміщений на формі



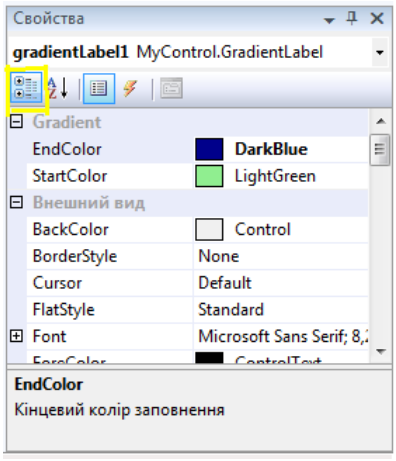
За допомогою атрибутів додамо інформацію про належність до категорії та опис властивостей `StartColor` та `EndColor`:

```
[Category("Gradient"), Description("Початковий колір заповнення")]
```

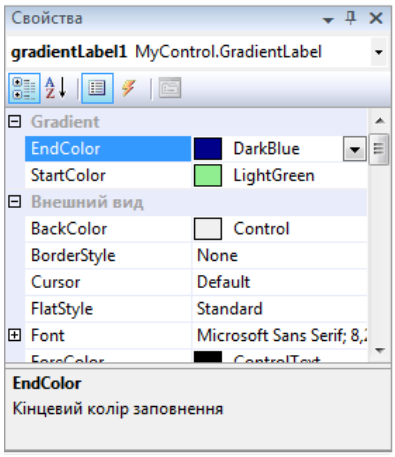
та

```
[Category("Gradient"), Description("Кінцевий колір заповнення ")]
```

відповідно. Тепер властивості зібрані в одну категорію у інспекторі властивостей.



Інформацію про значення за замовчуванням можна додати за допомогою конструкції `DefaultValue(typeof(Color), "Назва кольору")`. Після компіляції редактор властивостей більше не виділяє значень напівжирним шрифтом, що свідчить про наявність у нього інформації про початкове значення:



Для негайного оновлення компонента при зміні значень властивостей треба визначити метод `OnChangeProperties()`:

```
private void OnChangeProperties()
        {
            Invalidate();
        }
```

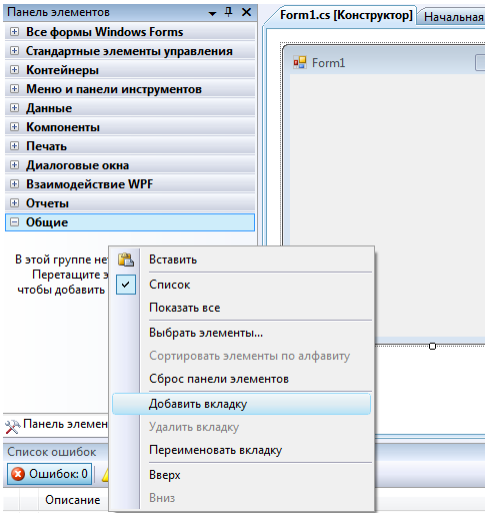
Таким чином маємо код

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

namespace MyControl
{
    [Description("Компоненнт Label з градієнтним заповненням")]
    [DefaultProperty("StartColor")]
    public partial class GradientLabel :Label
    {
        private Color startColor = Color.LightGreen;
        private Color endColor = Color.DarkBlue;

       [Category("Gradient"), Description("Початковий колір заповнення"),
        DefaultValue(typeof(Color), "LightGreen")]
```

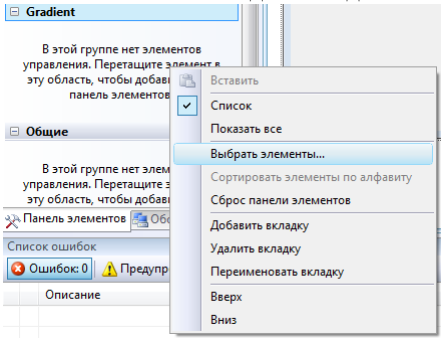
```
        public Color StartColor
        {
            get
            {
                if (startColor == Color.Empty)
                    return Parent.BackColor;
                return startColor;
            }
            set
            {
                startColor = value;
                OnChangeProperties();
            }
        }

       [Category("Gradient"), Description("Кінцевий колір заповнення"),
        DefaultValue(typeof(Color), "DarkBlue")]
        public Color EndColor
        {
            get { return endColor; }
            set { endColor = value; OnChangeProperties(); }
        }

        private void OnChangeProperties()
        {
            Invalidate();

        }

        protected override void OnPaint(PaintEventArgs pe)
        {
            // Викли базового OnPaint
            base.OnPaint(pe);

            // Заповнення
            Color c1 = Color.FromArgb(100, startColor);
            Color c2 = Color.FromArgb(100, endColor);
            Brush b = new System.Drawing.Drawing2D.
                            LinearGradientBrush(ClientRectangle, c1, c2, 10);
            pe.Graphics.FillRectangle(b, ClientRectangle);
            b.Dispose();
        }


        public GradientLabel()
        {
            InitializeComponent();
        }

    }
   }
```

Після компіляції створений компонент готовий до використання.

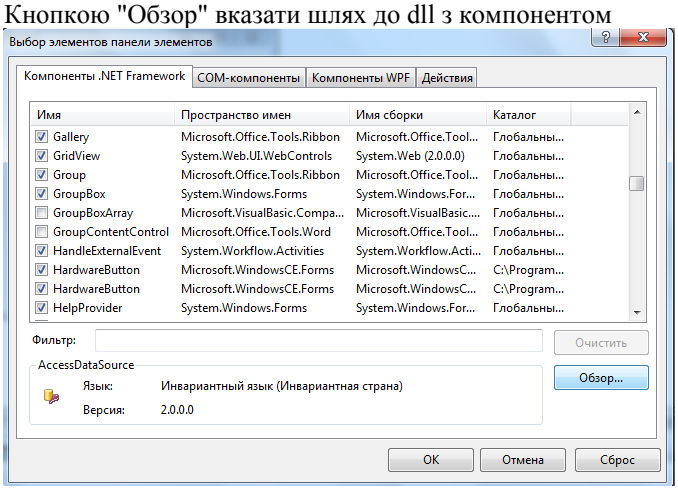
Для розміщення компонента на панелі елементів створимо новий проект Windows Forms. Додамо вкладку



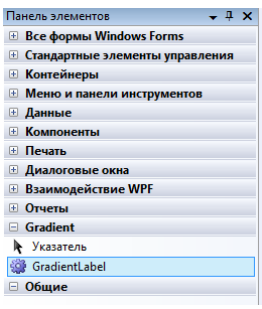
Визначимо елементи для вкладки



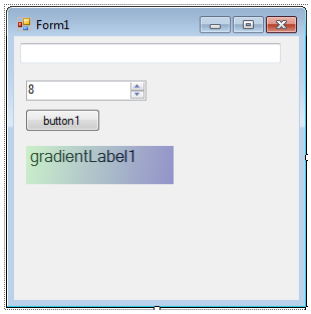
Кнопкою "Обзор" вказати шлях до dll з компонентом



та додати ("Ок"), після чого компонент з'явиться на панелі елементів Visual Studio



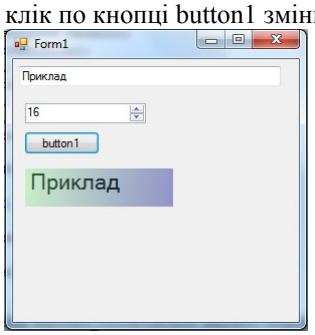
Далі можемо використовувати його, як і інші доступні компоненти.
Наприклад на формі з елементами



та обробником подій

```
private void button1_Click(object sender, EventArgs e)
        {
         gradientLabel1.Font = new System.Drawing.Font("Microsoft Sans
Serif", (float)numericUpDown1.Value);
            gradientLabel1.Text = textBox1.Text;
         }
```

клік по кнопці button1 змінює текст та параметри шрифту мітки gradientLabel1

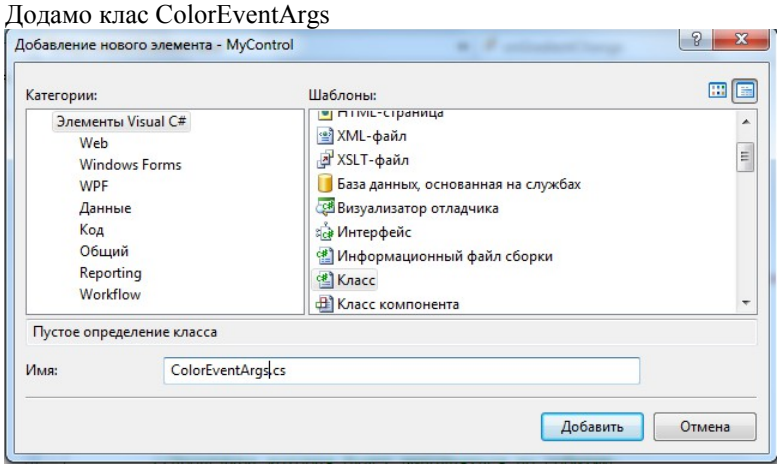


Розширимо функціональність компонента `GradientLabel`. Для цього розробимо подію, яка відбуватиметься при встановленні значення властивості `StartColor`. Крім того додамо використаємо механізм смарт-тегів для доступу до властивостей та методів в режимі розробки

3. Створення події.

Доповнимо функціональність компонента GradientLabel) створивши подію onStartColorIs, яка буде відбуватись при встановленні значення початкового кольору заповнення StartColor.

Додамо клас ColorEventArgs



з таким змістом:

```
class ColorEventArgs :EventArgs
    {
        private readonly Color clr;
        // Метод доступу
        public Color Clr { get { return clr; } }
        // Конструктор
```

```
        public ColorEventArgs(Color color){ clr=color;}
    }
```

Визначаємо подію onStartColorIs:

```
private event EventHandler<ColorEventArgs> onStartColorIs;
```

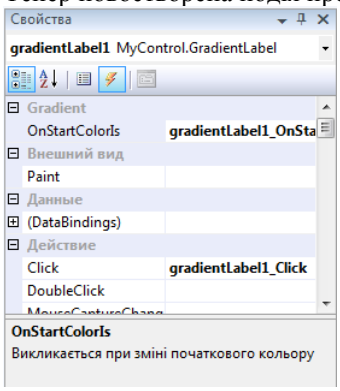
Додамо опис події для редактора властивостей та аксесори add та remove

```
[Category("Gradient"), Description("Викликається при зміні початкового
кольору")]
public event EventHandler<ColorEventArgs> OnStartColorIs
        {
            add{ onStartColorIs+=value;}
            remove{onStartColorIs-=value;}
        }
```

(зверніть увагу на описи **on**StartColorIs та **On**StartColorIs).
Подія генерується при встановленні значення startColor, у випадку, якщо вона не видалена

```
public Color StartColor
        {
            get
            {
                if (startColor == Color.Empty)
                    return Parent.BackColor;
                return startColor;
            }
            set
            {
                startColor = value;
                if (onStartColorIs != null) // Якщо подія зареєстрована
                {
                    // Згенерувати подію onStartColorIs
                    onStartColorIs(this, new ColorEventArgs(startColor));
                }
                OnChangeProperties();
            }
```

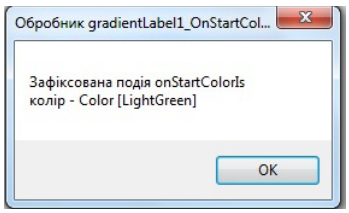
Тепер новостворена подія представлена у редакторі властивостей на вкладці Events



і їй можуть бути призначені обробники, наприклад

```
private void gradientLabel1_OnStartColorIs(object sender,
                                                    MyControl.ColorEventArgs e)
        {
            MessageBox.Show("Зафіксована подія onStartColorIs\n" + "колір - "
+ e.Clr.ToString(), "Обробник gradientLabel1_OnStartColorIs");
        }
```

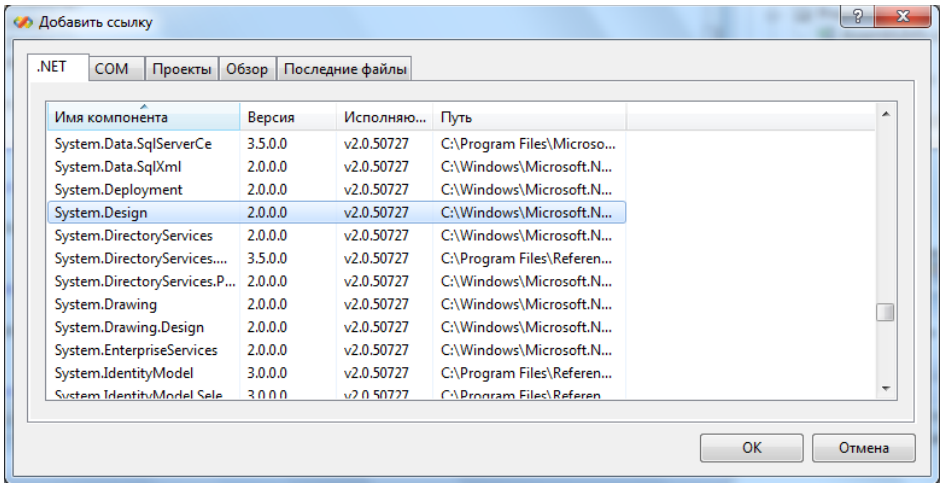
При кожній зміні початкового кольору обробник надає текстове представлення кольору:



4. Розширення функціональності режиму розробки

Для надання доступу до властивостей та методів часу розробки з використанням механізму смарт-тегів необхідно:

- додати посилання на System.Design



- створити клас `GradientLabelDesigner`, нащадок класу `ControlDesigner`:

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

namespace MyControl
{
    class GradientLabelDesigner : ControlDesigner
    {
        private DesignerActionListCollection actionLists;

        // Можна видалити успадковану властивість
        protected override void PreFilterProperties(IDictionary properties)
        {
            base.PreFilterProperties(properties);
            properties.Remove("BackColor");
        }

        public override DesignerActionListCollection ActionLists
        {
            get
            {
                // Якщо не побудований actionList
                if (actionLists == null)
                {
                    // Побудувати ActionList
                    actionLists = new DesignerActionListCollection();
```

```
                    // Додати смарт-тег
                    actionLists.Add(new
GradientLabelActionList(this.Component));
                }

                return actionLists;
            }
        }
    }
}
```

- створити клас `GradientLabelActionList` для організації доступу до властивостей та дій над компонентом:

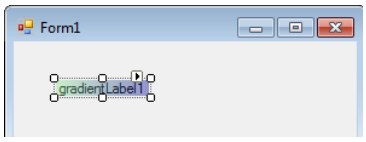
```
using System;
using System.Collections.Generic;
using System.Text;
using System.ComponentModel.Design;
using System.ComponentModel;
using System.Drawing;

namespace MyControl
{
    class GradientLabelActionList : DesignerActionList
    {
        GradientLabel glabel;
        DesignerActionUIService designerActionUIService;

        // Конструктор
        public GradientLabelActionList(IComponent component): base(component)
        {
            // Зберегти посилання на компонент, який редагується
            glabel = component as GradientLabel;
            // Зберегти посилання на сервіс ActionList
            designerActionUIService =
GetService(typeof(DesignerActionUIService)) as DesignerActionUIService;
        }

        // В діалог тега вынести дві властивості та одну команду
        public Color StartColor
        {
            get { return this.glabel.StartColor; }
            set
            {
                GetPropertyByName("StartColor").SetValue(glabel, value);
                // Обновити діалог тега
                designerActionUIService.Refresh(glabel);
            }
        }

        public Color EndColor
        {
            get { return this.glabel.EndColor; }
            set
            {
                GetPropertyByName("EndColor").SetValue(glabel, value);
                // Обновити діалог тега
                designerActionUIService.Refresh(glabel);
            }
        }

        public void InvertColors()
        {
            // Поміняти значення StartColor та EndColor
```

```
            Color tmp = glabel.StartColor;
            StartColor = glabel.EndColor;
            EndColor = tmp;
        }

        public override DesignerActionItemCollection GetSortedActionItems()
        {
            DesignerActionItemCollection items = new
DesignerActionItemCollection();

            // Додати дві группи
            items.Add(new DesignerActionHeaderItem("Властивості",
"Properties"));

            // Додати властивості в категорію Properties
            items.Add(new DesignerActionPropertyItem("StartColor",
                            "Початковий колір", "Properties",
                            "Колір початку градієнтного заповнення"));
            items.Add(new DesignerActionPropertyItem("EndColor",
                            "Кінцевий колір", "Properties",
                            "Колір завершенння градієнтного заповнення"));

            // Додати метод InvertColors, тільки якщо кольори різняться
            if (StartColor != EndColor)
            {
                items.Add(new DesignerActionHeaderItem("Методи", "Methods"));

                // Додати метод в категорію Methods
                items.Add(new DesignerActionMethodItem(this, "InvertColors",
                                "Інверсія кольорів градієнта", "Methods",
                                "Змінити початковий і кінцевий кольори
градієнтного заповнення"));
            }

            // Додати статичний текст
            items.Add(new DesignerActionHeaderItem("Інформація", "Info"));
            string info = string.Format("Розмір {0}x{1}", glabel.Width,
glabel.Height);
            items.Add(new DesignerActionTextItem(info, "Info"));

            return items;
        }

        // Повертає дескриптор властивостей за ім'ям
        private PropertyDescriptor GetPropertyByName(String propName)
        {
            PropertyDescriptor prop = TypeDescriptor.GetProperties(glabel)
[propName];
            if (prop == null)
            {
                throw new ArgumentException("Властивысть не існує",
propName);
            }
            return prop;
        }
    }
}
```

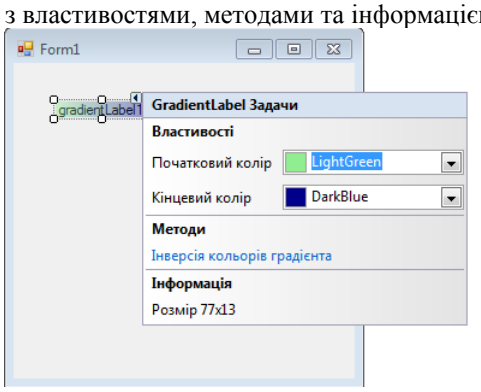
Для зв'язування дизайнера з компонентом додамо атрибут

```
[DesignerAttribute(typeof(GradientLabelDesigner))]
[DefaultProperty("StartColor")]
[DefaultEventAttribute("onStartColorIs")]
public partial class GradientLabel :Label
```

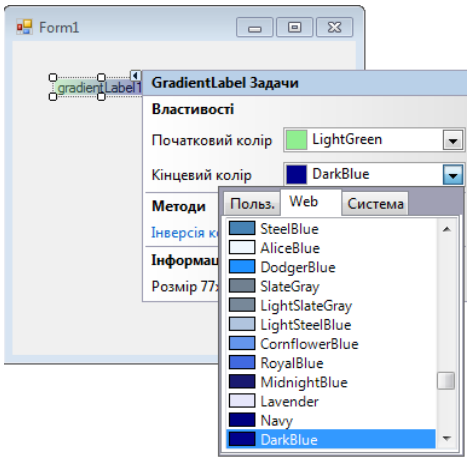
Тепер у режимі розробки маємо смарт-тег

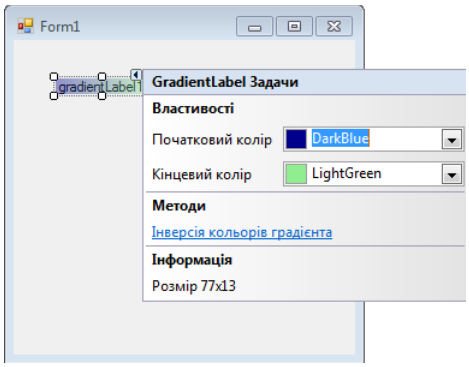


з властивостями, методами та інформацією



Метод та кольори тепер доступні для виконання/редагування у режимі розробки.





Завдання для самостійної роботи

1. Створити візуальний компонент з кількома властивостями та подіями, відмінними від наведених у прикладі. Використати механізм смарт-тегів для доступу до властивостей та однієї операції у режимі розробки. Успадкувати від компонента у відповідності до номера варіанта:

1. Button
2. Checkbox
3. ComboBox
4. GroupBox
5. Label

6. LinkLabel
7. ListBox
8. ListView
9. NumericUpDown
10. Panel
11. PictureBox
12. ProgressBar
13. RadioButton
14. RichTextBox
15. TextBox
16. TrackBar
17. TreeView

2. Додати створений компонент до панелі елементів Visual Studio.
3. Створити Windows Forms застосунок з використанням розробленого компонента та його унікальних властивостей.

Література

1. Агуров П.В. C#. Разработка компонентов *в* MS Visual Studio *2005/*2008 – СПб.: БХВ-Петербург, 2008