

## **Wykład 12**

### **Wiele okien programu**



## Własne okna dialogowe (okna modalne)



## Własne okna dialogowe

---

**Własne okna dialogowe** to zwykłe okna dziedziczące po klasie `Form`. Posiadają one wszystkie własności okien – możemy używać wszystkich kontrolek i tworzyć własny `Layout`. Różnica polega na sposobie ich uruchomienia.

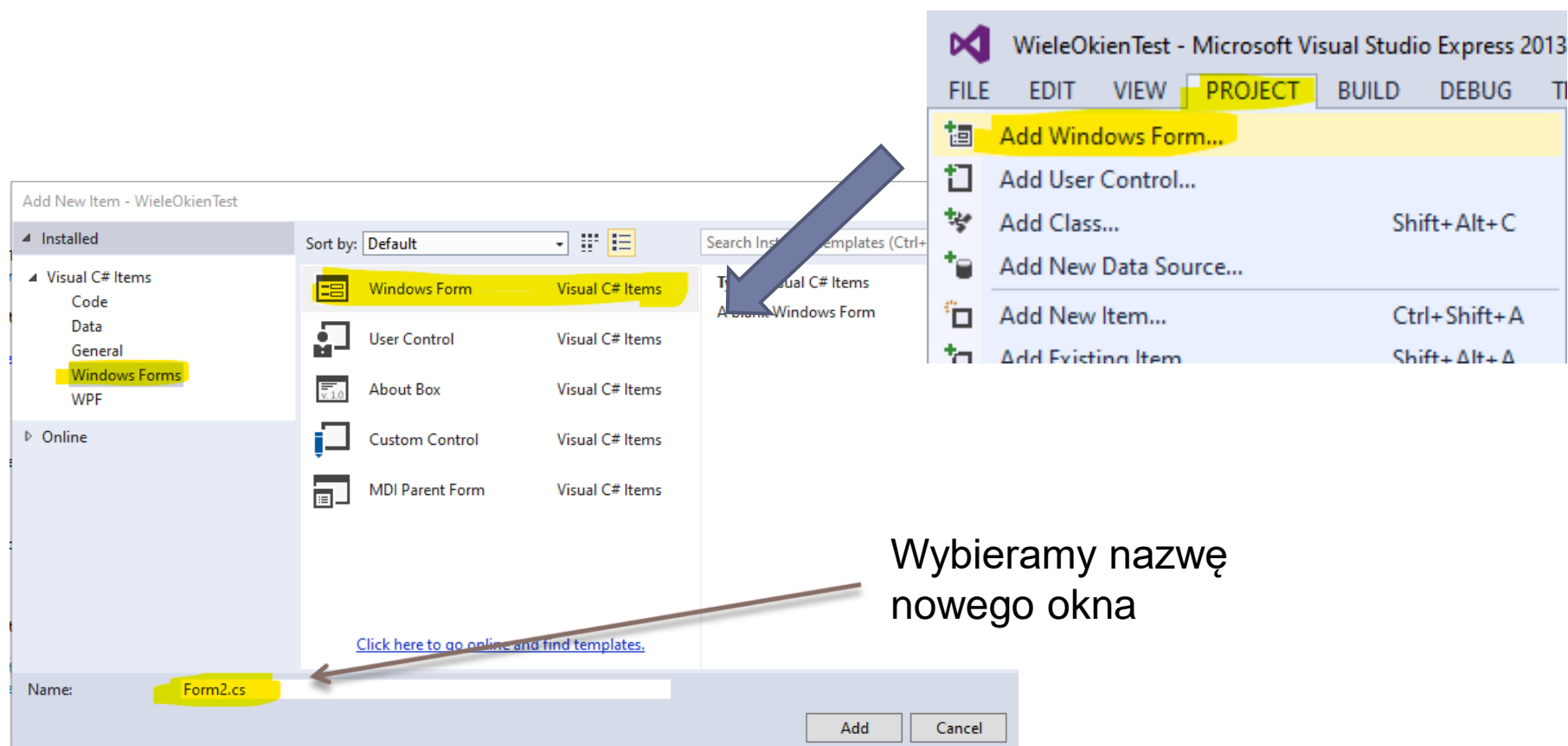
Okno tworzymy w następujących etapach:

1. Przygotowanie okna (stworzenie klasy i przygotowanie layoutu (analogicznie jak przygotowanie okna głównego programu)).
2. Zdefiniowanie odpowiedzi zwracanych przez okno (opcjonalnie).
3. W kodzie okna głównego – utworzenie nowego okna (referencji i obiektu za pomocą „new”) oraz wywołanie dla obiektu metody `.ShowDialog( )`.
4. Przyjęcie odpowiedzi okna (opcjonalnie).

W chwili wywołania okna dialogowego, okno nadrzędne jest zatrzymywane. Okno nadrzędne kontynuuje pracę dopiero po zamknięciu dialogowego.

## Własne okna dialogowe

**1. Przygotowanie okna (stworzenie klasy i przygotowanie layoutu**  
(analogicznie jak przygotowanie okna głównego programu).

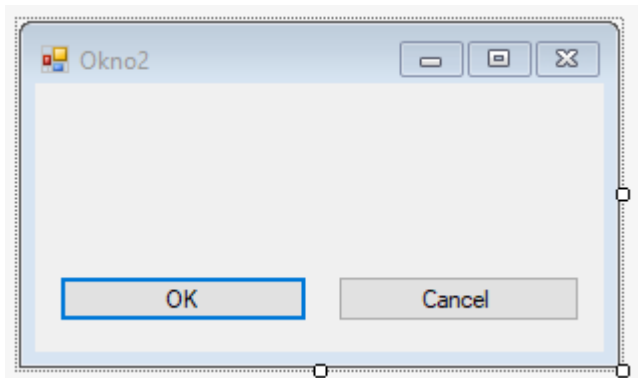


Wybieramy nazwę nowego okna

The image shows two screenshots from Microsoft Visual Studio Express 2013. The top screenshot shows the 'PROJECT' menu with 'Add Windows Form...' highlighted. The bottom screenshot shows the 'Add New Item' dialog box. In the 'Add New Item' dialog, the 'Visual C# Items' category is selected, and 'Windows Form' is highlighted in the list. The 'Name' field at the bottom is set to 'Form2.cs'. A blue arrow points from the 'Add Windows Form...' menu item to the 'Windows Form' item in the dialog. A red arrow points from the text 'Wybieramy nazwę nowego okna' to the 'Name' field.

# Własne okna dialogowe

## 2. Zdefiniowanie odpowiedzi zwracanych przez okno (opcjonalnie).



|                           |        |
|---------------------------|--------|
| <b>Behavior</b>           |        |
| AllowDrop                 | False  |
| AutoEllipsis              | False  |
| ContextMenuStrip          | (none) |
| DialogResult              | OK     |
| Enabled                   | None   |
| TabIndex                  | OK     |
| TabStop                   | Cancel |
| UseCompatibleTextRenderin | Abort  |
| Visible                   | Retry  |
| <b>Data</b>               |        |
| (ApplicationSettings)     | Ignore |
| (DataBindings)            | Yes    |
|                           | No     |

|              |         |
|--------------|---------|
| <b>Misc</b>  |         |
| AcceptButton | button1 |
| CancelButton | button2 |
| KeyPreview   | False   |

We właściwościach okna **wybieramy, przyciski, które będą odpowiadać za akcje „Accept” i „Cancel”** (wyjście z potwierdzeniem, lub bez)

**Dla wybranych Button-ów ustawiamy własność „DialogResult”**

Uwaga – teraz przyciski te automatycznie zamkną okno i przekażą do okna nadrzędnego wybraną odpowiedź w postaci zmiennej typu DialogResult

## Własne okna dialogowe

### 3. W kodzie okna głównego:

- utworzenie nowego okna (referencji i obiektu za pomocą „new”)
- wywołanie dla obiektu metody `.ShowDialog()`.

```
public partial class Form1 : Form
{
    Okno2 okno2;
```

Tworzymy zmienną referencyjną dla nowego okna.

Nazwa klasy zgodnie z nazwą jaką nadaliśmy oknu (slajd 4)

```
    public Form1()
    {
        InitializeComponent();
        okno2 = new Okno2();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        okno2.ShowDialog();
    }
}
```

Tworzymy obiekt - nowe okno. (obiekt powstaje tylko w pamięci, nie jest jeszcze wyświetlany na ekranie)

Wyświetlamy okno jako dialogowe (modalne), tzn., dopóki go nie zamkniemy nie możemy pracować w oknie głównym.

# Własne okna dialogowe

## 4. Przyjęcie odpowiedzi okna (opcjonalnie).

```
public partial class Form1 : Form
{
    Okno2 okno2;
    public Form1()
    {
        InitializeComponent();
        okno2 = new Okno2();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        DialogResult odp = okno2.ShowDialog();
        if (odp==DialogResult.OK)
            label2.Text = odp.ToString();
    }
}
```

Metoda `.ShowDialog()` zwraca odpowiedź typu `DialogResult`

Jeżeli odpowiedź okna równa będzie np. „`DialogResult.OK`” podejmujemy akcję (w tym przykładzie wypisujemy odpowiedź w postaci tekstowej)



## Okna niemodalne





## Własne okna (niemodalne)

---

**Własne okna niemodalne** to zwykłe okna dziedziczące po klasie `Form`. Posiadają one wszystkie własności okien – możemy używać wszystkich kontrolek i tworzyć własny `Layout`. W odróżnieniu od okien dialogowych (modalnych) okno nadrzędne pracuje równolegle z nim.

Okno tworzymy w następujących etapach:

1. Przygotowanie okna (stworzenie klasy i przygotowanie layoutu (analogicznie jak przygotowanie okna głównego programu)).
2. Zdefiniowanie odpowiedzi zwracanych przez okno (opcjonalnie).
3. W kodzie okna głównego – utworzenie nowego okna (referencji i obiektu za pomocą „new”) oraz wywołanie dla obiektu metody `.Show()`.
4. ~~Przyjęcie odpowiedzi okna (opcjonalnie).~~

**Różnice pomiędzy omówionym wcześniej oknem dialogowym oknem a niemodalnym widzimy głównie w punkcie 3, czyli w sposobie jego otwierania.**

## Własne okna (niemodalne)

### 3. W kodzie okna głównego:

- utworzenie nowego okna (referencji i obiektu za pomocą „new”)
- wywołanie dla obiektu metody `.Show()`.

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

```
{
```

```
Okno2 okno2;
```

```
public Form1()
```

```
{
```

```
InitializeComponent();
```

```
}
```

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
okno2 = new Okno2();
```

```
okno2.Show();
```

```
}
```

```
}
```

Tworzymy zmienną referencyjną dla nowego okna.

Nazwa klasy zgodnie z nazwą jaką nadaliśmy oknu (slajd 4)

- Tworzymy obiekt - nowe okno
  - Wyświetlamy okno za pomocą metody `.Show()`
- Obie te operacje wykonujemy każdorazowo przy otwarciu okna.**



## Własne okna (niemodalne)

---

### Trochę teorii

**Metoda .Show() nie zwraca odpowiedzi** - w odróżnieniu od wcześniej użytej metody .ShowDialog( )

**Okna dialogowe (modalne)** tworzone są tylko raz (najczęściej w konstruktorze okna nadrzędnego) i istnieją (są przechowywane w pamięci), aż do chwili, gdy istnieje jakakolwiek referencja, która na nie wskazuje.

Otwierając okno dialogowe wielokrotnie w istocie wywołujemy wielokrotnie ten sam obiekt. Pojawia się on na ekranie wraz ze wszystkimi danymi, takimi jak np. tekst wpisany do pól textBox.

**Okna niemodalne** usuwane są z pamięci po ich zamknięciu. Aby otworzyć takie okno po raz drugi należy utworzyć nowy obiekt, inaczej spowodujemy błąd programu.



## Odbieranie danych z okna dialogowego



## Odbieranie danych z okna dialogowego

---

### Trochę teorii

**Ponieważ okna dialogowe (modalne) nie są usuwane z pamięci po ich zamknięciu, odebranie danych z okna jest możliwe po zamknięciu okna i odebraniu jego odpowiedzi (DialogResult).**

**Wywołanie okna dialogowego metodą .ShowDialog()** zatrzymuje pracę okna głównego – przetwarzanie kodu zatrzymuje się na poleceniu .ShowDialog()

**Z tego powodu odczytanie danych z okna podrzędnego przez główne następuje po zamknięciu okna podrzędnego.**

**Wszelkie dane wprowadzone do okna dialogowego w trakcie jego pracy zachowywane są w pamięci, aż do ewentualnego, ponownego wywołania okna i ich modyfikacji.**

## Komunikacja pomiędzy dwoma oknami

W kodzie okna głównego:

```
Okno2 okno2;  
public Form1()  
{  
    InitializeComponent();  
    okno2 = new Okno2();  
}  
private void button1_Click(object sender, EventArgs e)  
{  
    if (okno2.ShowDialog() == DialogResult.OK)  
    {  
        String danaZokna1 = okno2.textBox1.Text;  
        int danaZokna2 = okno2.dana;  
    }  
}
```

W tym momencie program jest zatrzymany i czeka na zamknięcie okna dialogowego

Aby sięgnąć do kontrolek i zmiennych okna dialogowego odwołujemy się do jego nazwy (obiektu nie klasy)

## Komunikacja pomiędzy dwoma oknami

W kodzie okna dialogowego:

```
public partial class Okno2 : Form
{
    public int dana = 100;
    public Okno2()
    {
        InitializeComponent();
    }
}
```

### UWAGA:

Wszystkie zmienne i kontrolki do których sięgać będzie okno nadrzędne muszą być publiczne.

|                |          |
|----------------|----------|
| Design         |          |
| (Name)         | textBox1 |
| GenerateMember | True     |
| Locked         | False    |
| Modifiers      | Public   |
| Focus          |          |

Kontrolki są domyślnie prywatne – można to zmienić w ustawieniach.



## **Dwustronna komunikacja pomiędzy dwoma oknami (niemodalnymi)**





## Komunikacja pomiędzy dwoma oknami

---

### Trochę teorii

**Ponieważ okna niemodalne usuwane są z pamięci** po ich zamknięciu, odebranie danych z okna jest możliwe tylko, dopóki jest ono wyświetlane.

**Wywołanie okna niemodalnego** metodą `.Show()` nie zatrzymuje pracy (przetwarzania kodu) okna głównego.

**Z tego powodu** odczytanie danych z okna podrzędnego przez główne następuje od razu po jego utworzeniu.

**Wszelkie dane wprowadzone do okna** dialogowego w trakcie jego pracy należy przesłać w odwrotną stronę: z okna podrzędnego do głównego.

# Komunikacja pomiędzy dwoma oknami

## PRZEKAZANIE DANYCH Z OKNA GŁÓWNEGO DO PODRZĘDNEGO

```
public partial class Form1 : Form
{
    Okno2 okno2;
    public Form1()
    {
        InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        okno2 = new Okno2();
        okno2.Show();
        String danaZokna1 = okno2.textBox1.Text;
        int danaZokna2 = okno2.dana;
        // UWAGA: dane będą pobrane tylko raz, po uruchomieniu okna2
        // zmiany wprowadzone później będą zignorowane
    }
}
```

W tym momencie jest wywoływane okno podrzędne, ale program główny działa nadal

Aby sięgnąć do kontrolek i zmiennych okna podrzędnego odwołujemy się do jego nazwy (obiektu nie klasy)

# Komunikacja pomiędzy dwoma oknami

## PRZEKAZANIE DANYCH Z OKNA PODRZĘDNEGO DO GŁÓWNEGO

W kodzie okna głównego:

```
public partial class Form1 : Form
{
    Okno2 okno2;
    public Form1()
    {
        InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        okno2 = new Okno2(this);
        okno2.Show();
    }
}
```

Otwierając okno podrzędne musimy przekazać do niego informację o tym, które okno je wywołało.

W tym celu, w parametrze konstruktora obiektu `okno2`, umieszczamy `this`, czyli referencję na obiekt, wewnątrz którego aktualnie jesteśmy

# Komunikacja pomiędzy dwoma oknami

## PRZEKAZANIE DANYCH Z OKNA PODRZĘDNEGO DO GŁÓWNEGO

W kodzie okna podrzędnego:

```
public partial class Okno2 : Form
```

```
{
```

```
    Form1 okno1;
```

```
    public Okno2(Form1 ktoOtworzył)
```

```
{
```

```
        InitializeComponent();
```

```
        okno1 = ktoOtworzył;
```

```
}
```

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    okno1.label1.Text = "tekst przekazany z okna podrzędnego";
```

```
    Close();
```

```
}
```

```
}
```

Tworzymy referencję na obiekt klasy odpowiadającej oknu nadrzędnemu

Jako parametr konstruktora dodajemy inną referencję na okno nadrzędne- tu trafi adres wysłany za pomocą **this** (poprzedni slajd)

Zapamiętujemy przesłany adres w zmiennej globalnej, żeby móc używać go poza konstruktorem.

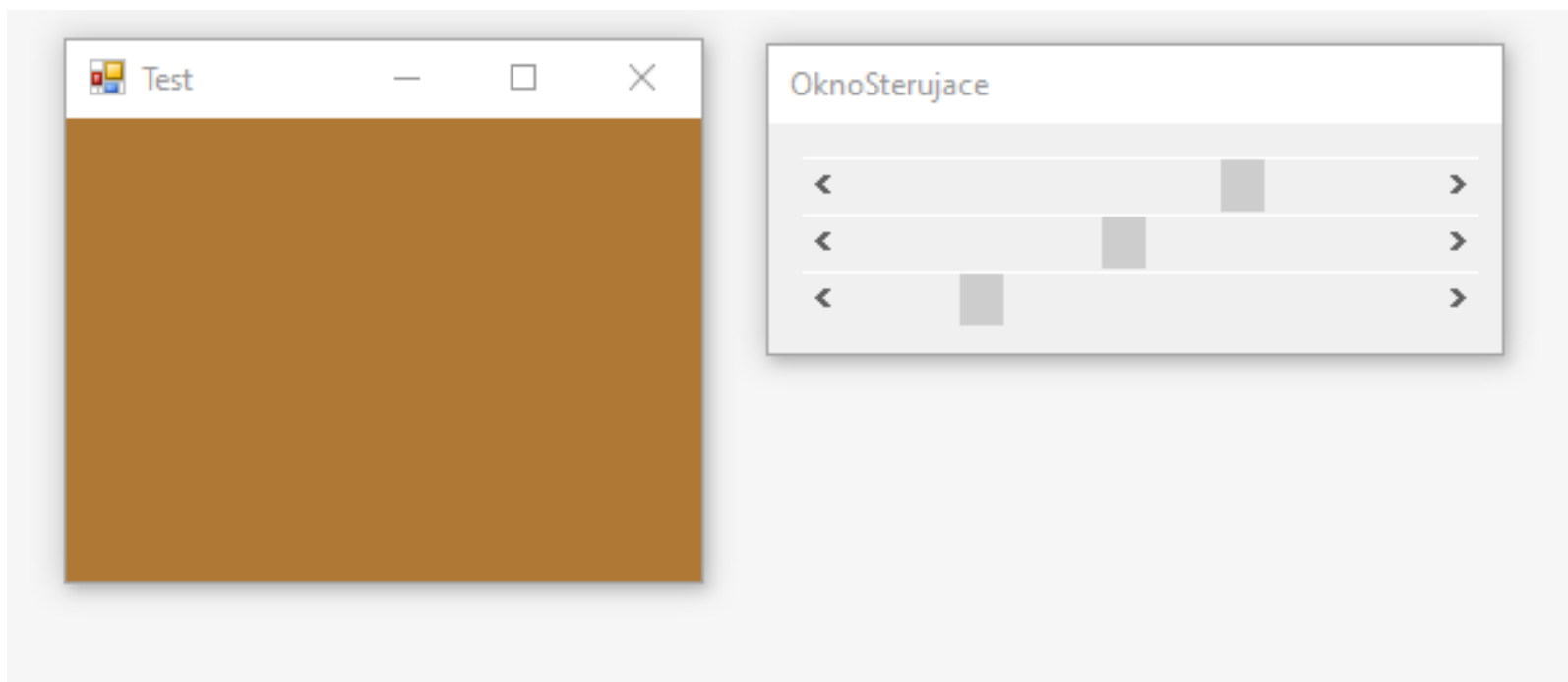
Opcjonalnie:  
zamykamy okno

Teraz możemy sięgnąć do kontrolek i pól okna nadrzędnego – UWAGA: tylko tych publicznych



## Przykład – próbnik kolorów 2

Aplikacja sterowana z osobnego okna



## Przykład 2 – okna niemodalne

1. Tworzymy okno główne - zawiera tylko Panel
2. Tworzymy okno sterujące (patrz slajd 4)

- Ustawienia:

| FormBorderStyle | FixedToolWindow |            |
|-----------------|-----------------|------------|
| +               | Location        | 500; 180   |
| +               | MaximumSize     | 0; 0       |
| +               | MinimumSize     | 0; 0       |
| +               | Padding         | 0; 0; 0; 0 |
| +               | Size            | 300; 128   |
|                 | StartPosition   | Manual     |
| ControlBox      |                 | False      |

3. W konstruktorze okna głównego:
  - Tworzymy obiekt okna sterującego uruchamiamy go metodą `.Show()` z parametrem `this`.
4. W konstruktorze okna sterującego:
  - Odbieramy referencję do okna głównego.
5. Oprogramowujemy kontrolki okna sterującego. Kolor panelu z okna głównego zmieniamy korzystając z referencji przekazanej do konstruktora

## Przykład 2 – okna niemodalne

```
namespace WykładPrubnikKolorow_wieleOkien
{
    public partial class Form1 : Form
    {
        OknoSterujace oknoSterujace;
        //Przygotowujemy zmienną referencyjną klasy okna sterującego
        public Form1()
        {
            InitializeComponent();
            oknoSterujace = new OknoSterujace(this);
            //tworzymy obiekt okna sterującego, w parametrze przekazujemy
            //referencję do okna w którym jesteśmy (this)
            oknoSterujace.Show();
            //Uruchamiamy okno sterujące
        }
    }
}
```

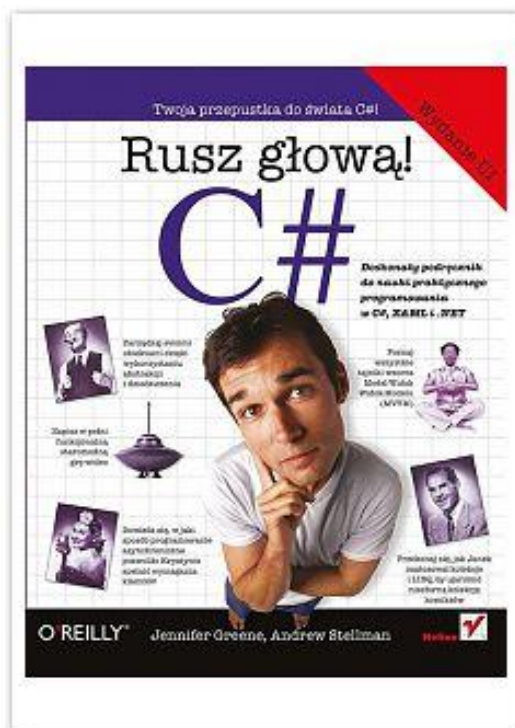
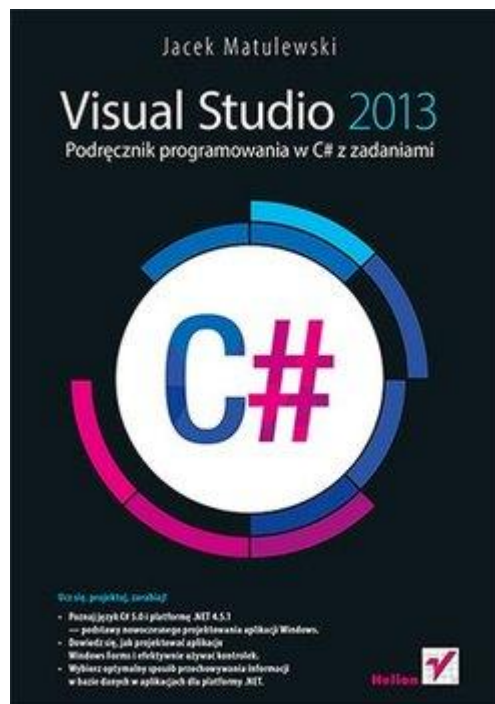
## Przykład 2 – okna niemodalne

```

namespace WykladPrubnikKolorow_wieleOkien
{
    public partial class OknoSterujace : Form
    {
        Form1 oknoGlowne; //zmienna referencyjna do klasy okna gł.
        public OknoSterujace(Form1 ktoUtworzył) //w konstruktorze odbieramy referencje do okna głównego
        {
            InitializeComponent();
            oknoGlowne = ktoUtworzył; //zapamiętujemy referencję przekazaną do konstruktora
        }
        private void ustawKolor() //własna metoda ustawijająca kolor
        {
            Color kolor = Color.FromArgb(255,
                hScrollBar1.Value,
                hScrollBar2.Value,
                hScrollBar3.Value);
            oknoGlowne.panel1.BackColor = kolor; //sięgamy do panelu "panel1" w oknie głównym i zmieniamy jego tło
        }
        private void hScrollBar1_Scroll(object sender, ScrollEventArgs e){
            ustawKolor(); //w metodzie onScroll suwaków wywołujemy metodę ustawKolor()
        }
        private void hScrollBar2_Scroll(object sender, ScrollEventArgs e){
            ustawKolor();
        }
        private void hScrollBar3_Scroll(object sender, ScrollEventArgs e){
            ustawKolor();
        }
    }
}

```





Użyte w tej prezentacji tabelki pochodzą z książki: Visual Studio 2013. Podręcznik programowania w C# z zadaniami Autor: Matulewski Jacek, Helion