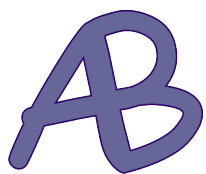
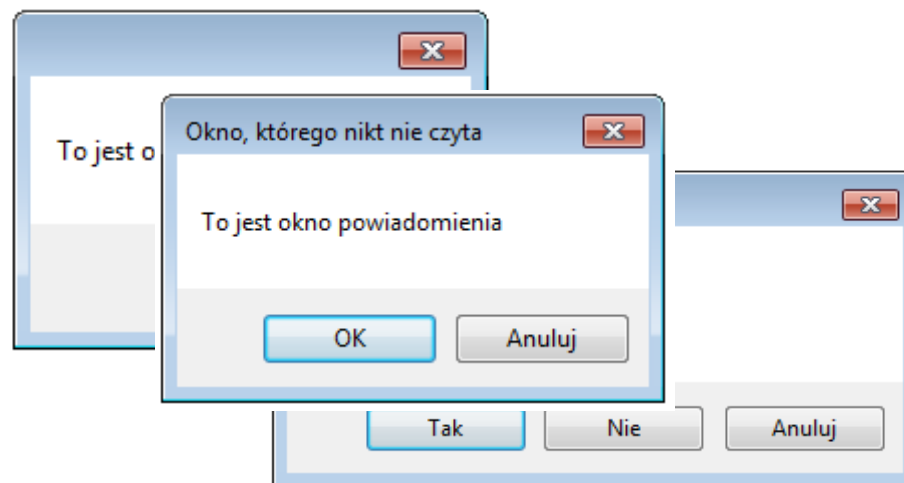


Wykład 7 - 8 Okna dialogowe



Okna dialogowe

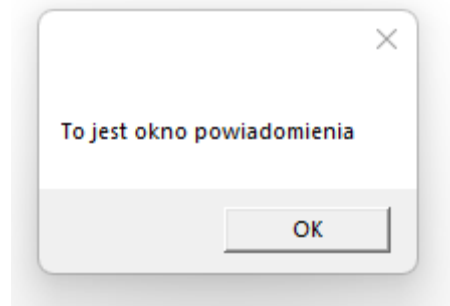


Okna dialogowe - komunikaty

Okno komunikatu to okno dialogowe, które może służyć do wyświetlania informacji tekstowych. Pozwala ono także użytkownikom na podejmowanie decyzji za pomocą przycisków.

```
MessageBox.Show("To jest okno powiadomienia");
```

W „standardzie” dostajemy tekst i przycisk „OK.”



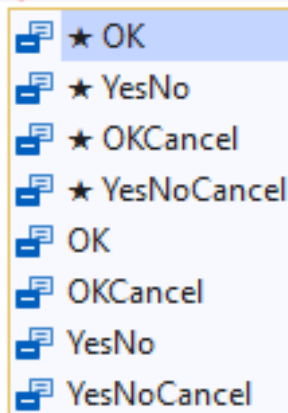
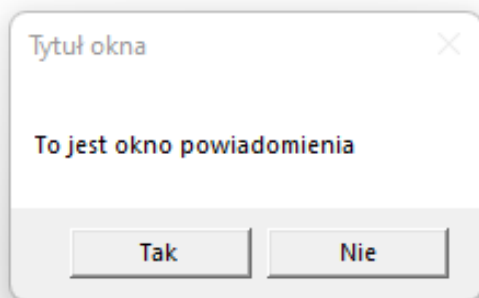
Okno dialogowe możemy uzupełniać o różne elementy.

Okna dialogowe - komunikaty

Okno dialogowe możemy uzupełniać o różne elementy.

```
MessageBox.Show("To jest okno powiadomienia",  
                "Tytuł okna",  
                MessageBoxButton.);
```

Nagłówek



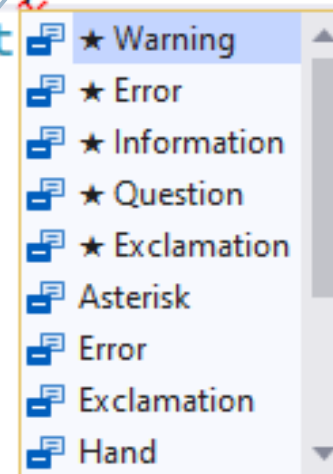
Rodzaje przycisków

Okna dialogowe - komunikaty

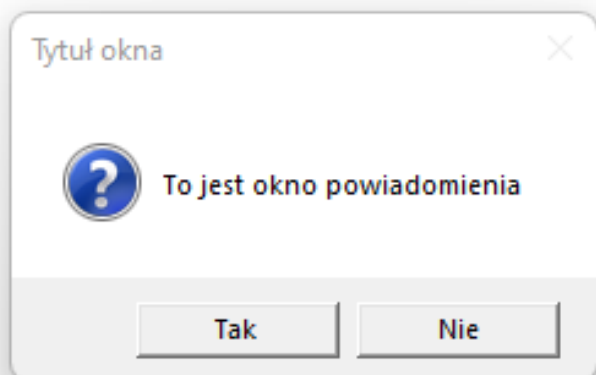
Okno dialogowe możemy uzupełniać o różne elementy.

Defaltowy przycisk
(zostanie wybrany, gdy
użytkownik naciśnie
Enter)

```
MessageBox.Show("To jest okno powiadomienia"  
    "Tytuł okna",  
    MessageBoxButton.YesNo,  
    MessageBoxImage.  
    MessageBoxResult
```



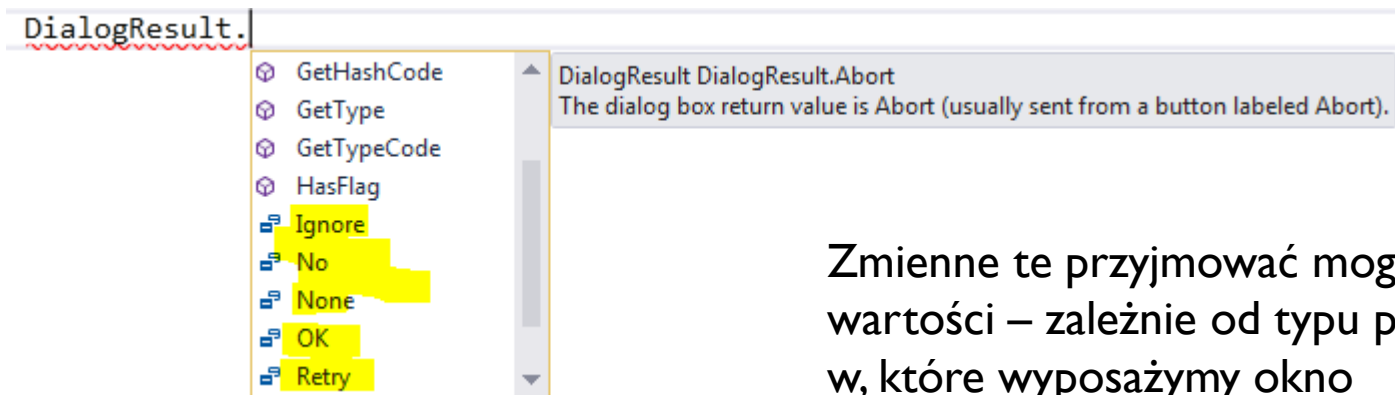
Wybór ikony



Okna dialogowe - komunikaty

Odpowiedź zwracaną przez okno (który przycisk naciśnięto) zapisać możemy w zmiennej typu `DialogResult`.

```
DialogResult odpowiedz = MessageBox.Show("To jest okno z pytaniem", "Pytanie",  
    MessageBoxButton.YesNoCancel,  
    MessageBoxIcon.Question);
```

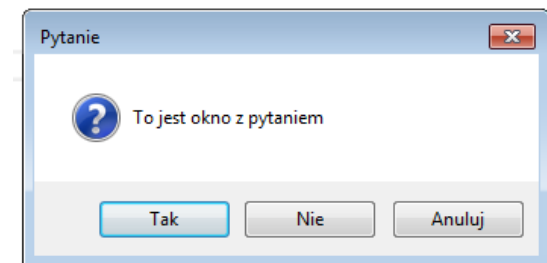


Zmienne te przyjmować mogą różne wartości – zależnie od typu przycisków w, które wyposażymy okno

Okna dialogowe - komunikaty

Okno o trzech możliwych odpowiedziach:

```
DialogResult odpowiedz = MessageBox.Show("To jest okno z pytaniem", "Pytanie",  
    MessageBoxButton.YesNoCancel,  
    MessageBoxIcon.Question);  
  
switch (odpowiedz)  
{  
    case DialogResult.Yes: //akcja, jeżeli naciśnięto "TAK"  
        break;  
    case DialogResult.No: //akcja, jeżeli naciśnięto "NIE"  
        break;  
    case DialogResult.Cancel: //akcja, jeżeli naciśnięto "ANULUJ"  
        break;  
}
```

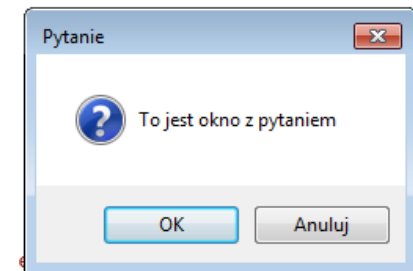


Okna dialogowe - komunikaty

Okno o dwóch odpowiedziach:

```
DialogResult odpowiedz2 = MessageBox.Show("To jest okno z pytaniem", "Pytanie",  
    MessageBoxButton.OKCancel,  
    MessageBoxIcon.Question);
```

```
if (odpowiedz2==DialogResult.OK)  
{  
    //Akcja, jeżeli naciśnięto "OK"  
}  
else  
{  
    //Akcja, jeżeli naciśnięto "ANULUJ"  
}
```



Zapisywanie odpowiedzi okna w pośredniczącej zmiennej nie jest konieczne.

```
if (MessageBox.Show("To jest okno z pytaniem", "Pytanie",  
    MessageBoxButton.OKCancel, MessageBoxIcon.Question) == DialogResult.OK)  
{  
    //Akcja, jeżeli naciśnięto "OK"  
}  
else  
{  
    //Akcja, jeżeli naciśnięto "ANULUJ"  
}
```

Zapis
skrócony:

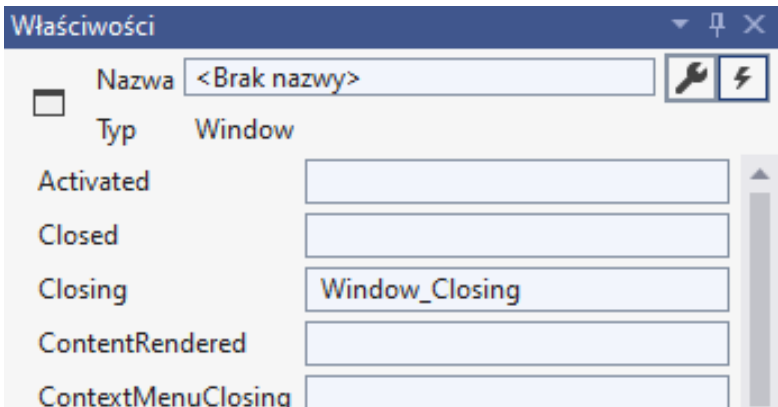


Okna dialogowe - komunikaty

Przykład - potwierdzenie zamknięcia programu

```
private void button01_Click(object sender, RoutedEventArgs e)
{
    Close();
}
```

Do zamknięcia okna (w przypadku okna głównego równoznaczne z zamknięciem programu) służy polecenie `Close()`;

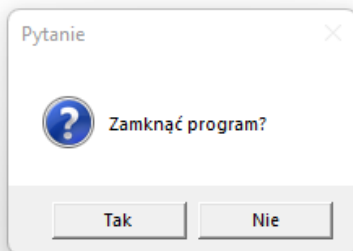


Aby przed zamknięciem okna wykonana została jakaś czynność (np. wyświetlenie okna dialogowego) oprogramować należy zdarzenie **Closing**

Okna dialogowe - komunikaty

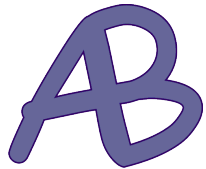
Przykład - potwierdzenie zamknięcia programu

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (MessageBox.Show("Zamknąć program?", "Pytanie",
        MessageBoxButton.YesNo, MessageBoxImage.Question,
        MessageBoxResult.No) == MessageBoxResult.No)
        e.Cancel = true;
}
```

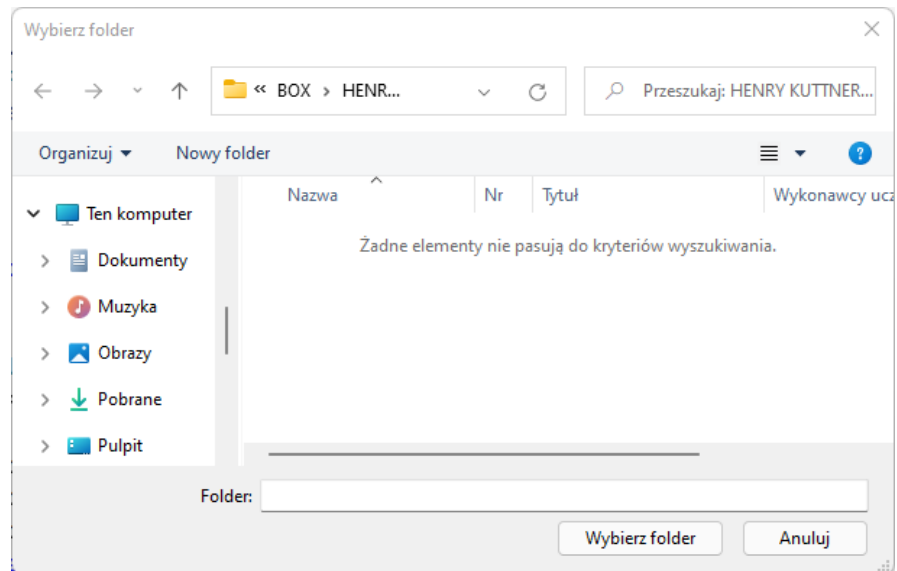


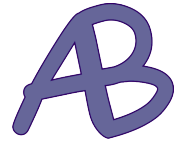
„e” to zdarzenie przesłane w parametrze.

Ustawienie jego pola „Cancel” na „true” kasuje polecenie zamknięcia okna.



Okna dialogowe systemu





Okna dialogowe – Okno otwierania pliku

Na początek ważna informacja: samo okno otwierania pliku żadnego pliku nie otworzy !

Okno (klasa OpenFileDialog) służy tylko do znalezienia nazwy i ścieżki do pliku.

Krok1: Utworzenie obiektu klasy „OpenFileDialog”

```
OpenFileDialog openFileDialog = new OpenFileDialog();
```

Krok2: Dodanie tytułu okna

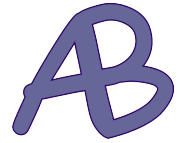
```
openFileDialog.Title = "Otwieranie pliku";
```

Krok3: Zdefiniowanie filtru typów plików (rozszerzeń)

```
openFileDialog.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
```

Krok4: Wywołanie metody „ShowDialog()” okna dialogowego

```
openFileDialog.ShowDialog()
```

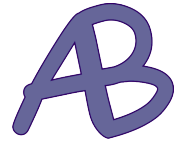


Okna dialogowe – Okno otwierania pliku

Metoda ShowDialog() okna dialogowego zwraca wartość prawda lub fałsz zależnie od tego jak zostało zamknięte okno (przyciskiem „Otwórz” lub „Anuluj”).

```
OpenFileDialog openFileDialog = new OpenFileDialog();
openFileDialog.Title = "Otwieranie pliku";
openFileDialog.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
openFileDialog.Multiselect = true;
if (openFileDialog.ShowDialog() == true)
{
    text_01.Text = openFileDialog.FileName;
}
```

Po zamknięciu okna przyciskiem „Otwórz” w polu obiektu „openFileDialog.FileName” znajduje się ścieżka i nazwa wybranego pliku.

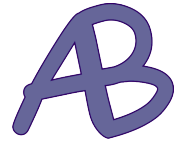


Okna dialogowe – Folder startowy okien

Folder startowy dla okien odczytu i zapisu do pliku możemy wybrać za pomocą parametru: `.InitialDirectory`

```
openFileDialog.InitialDirectory =  
Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
```

Ponieważ w różnych instalacjach Windows foldery specjalne (Dokumenty, Obrazy, Pulpit itp.) mają różne adresy, posługujemy się zmienną środowiskową `Environment.SpecialFolder`



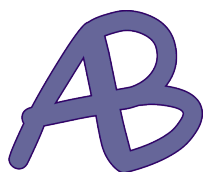
Okna dialogowe – Okno zapisu do pliku

Okno zapisu do pliku nie jest podobne do okna odczytu, stąd różnicą że nie wybieramy istniejącego pliku tylko katalog i podajemy nazwę pliku do utworzenia.

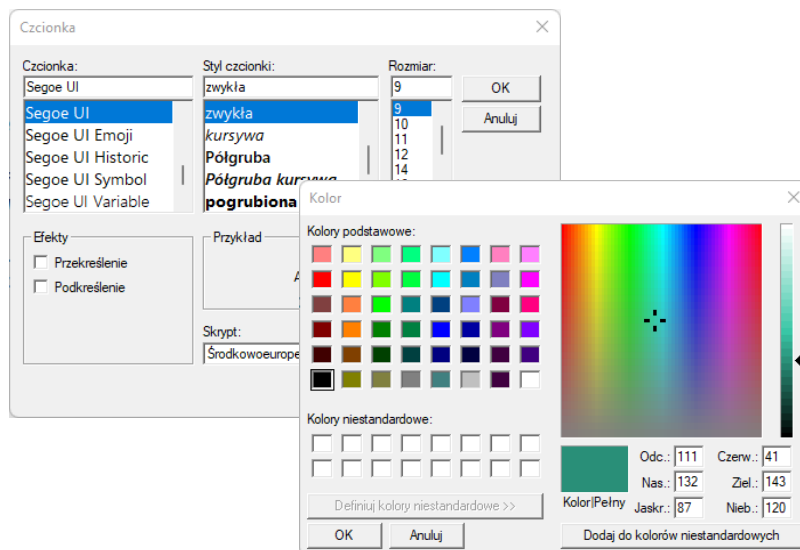
```
SaveFileDialog saveFileDialog = new SaveFileDialog();
saveFileDialog.Filter = "Text files(*.txt)|*.txt|All files(*.*)|*.*";
saveFileDialog.DefaultExt = "txt";
saveFileDialog.Title = "Okno zapisywania do pliku";
saveFileDialog.InitialDirectory =
    Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
if (saveFileDialog.ShowDialog() == true)
{
    text_02.Text = saveFileDialog.FileName;
}
```

Po zamknięciu okna przyciskiem „Zapisz” w polu obiektu „openFileDialog.FileName” znajduje się wybrana ścieżka i podana nazwa pliku.

Rozszerzenie pliku definiowane jest przez wybrany filtr. Oraz (w drugiej kolejności) przez parametr DefaultExt.



Okna dialogowe z Windows Forms



Okna dialogowe – Okna dialogowa Windows Forms

Windows Forms zawierało więcej okien dialogowych. Co prawda sprawiają one pewne problemy, jednak w niektórych sytuacjach można z nich skorzystać również w aplikacjach WPF.

Windows Forms ?

☒ Włącz Windows Forms dla tego projektu.

W ustawieniach projektu należy włączyć obsługę Windows Forms

Do okien dialogowych WF odwołujemy się poprzez przestrzeń nazw `System.Windows.Forms`.

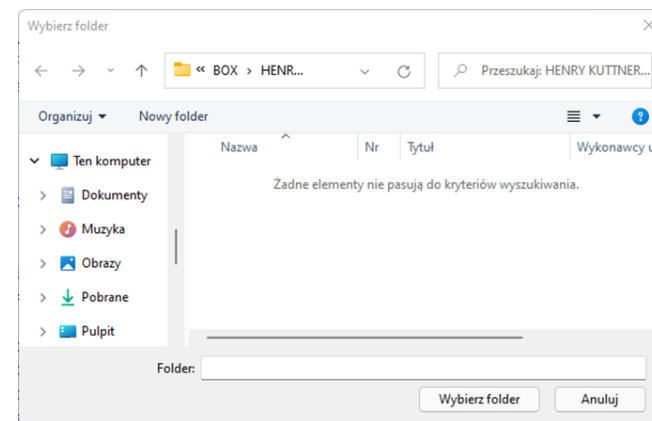
Pewną różnicą jest także to, że okna te nie zwracają wartości `prawda` lub `fałsz` lecz zmienną typu: `System.Windows.Forms.DialogResult.OK`

Okna dialogowe – FolderBrowserDialog

Okno

System.Windows.Forms.FolderBrowserDialog

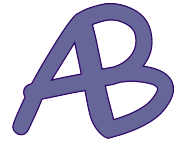
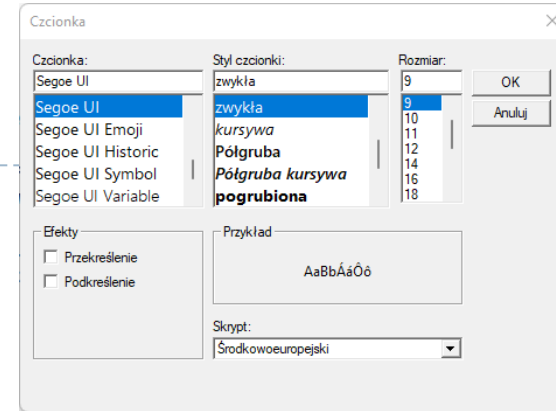
służy do wyboru katalogu



```
System.Windows.Forms.FolderBrowserDialog folderBrowserDialog =  
    new System.Windows.Forms.FolderBrowserDialog();  
if (folderBrowserDialog.ShowDialog()==System.Windows.Forms.DialogResult.OK)  
{  
    text_04.Text = folderBrowserDialog.SelectedPath;  
}
```

Okna dialogowe – FontDialog

Okno `System.Windows.Forms.FontDialog` służy do określenia parametrów używanego fontu.



```
System.Windows.Forms.FontDialog fontDialog = new System.Windows.Forms.FontDialog();
if (fontDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
{
    text_05.Text = fontDialog.Font.FontFamily.Name + " - " + text_05.Text;
    text_05.FontFamily = new FontFamily(fontDialog.Font.FontFamily.Name);
    text_05.FontSize = fontDialog.Font.Size;
}
```

Wykorzystanie tego okna utrudniony jest przez brak kompatybilności pomiędzy Windows Forms, a WPF.

Okno dialogowe zwraca obiekt klasy `Font`, jednak w WPF nie możemy bezpośrednio ustawić parametru `.Font` dla kontrolki.

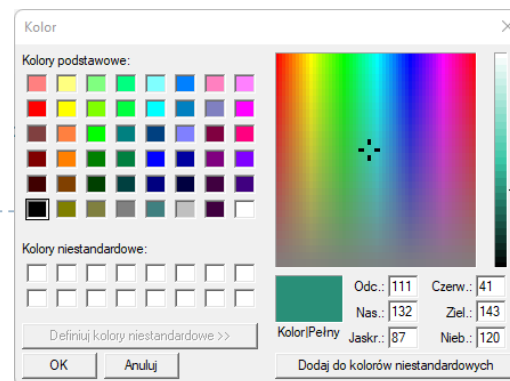
Ponieważ zmienna `font` jest obiektem możemy odczytać z niej najważniejsze parametry (w tym przykładzie `FontFamily` i `FontSize`, czyli rodzaj i rozmiar czcionki).

Inne właściwości takie jak na przykład `Style` nie są kompatybilne i nie można ich bezpośrednio przenieść na ustawienia kontrolki WPF.

Okna dialogowe – ColorDialog



Okno `System.Windows.Forms.ColorDialog` służy do wybierania koloru.



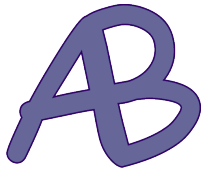
Tu także pojawia się brak kompatybilności pomiędzy Windows Forms, a WPF. Co prawda okno dialogowe zwraca zmienną typu `Color`, jednak przy próbie jej zastosowania pojawia się błąd:

CS0029: Nie można niejawnie przekonwertować typu „`System.Drawing.Color`” na „`System.Windows.Media.Color`”.

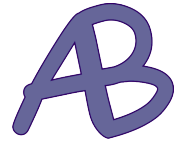
Problem wynika stąd, że pomimo tej samej nazwy klasa `Color` stosowana w WPF nie jest tą samą klasą, co klasa `Color` stosowana w WF (znajdują się w różnych przestrzeniach nazw).

Najprostszym rozwiązaniem problemu jest wyodrębnienie składowych RGB z obiektu kolor zwróconego przez okno dialogowe i utworzenia na ich podstawie obiektu klasy `Color` zgodnego z WPF.

```
System.Windows.Forms.ColorDialog colorDialog = new System.Windows.Forms.ColorDialog();
if (colorDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
{
    byte R = colorDialog.Color.R;
    byte G = colorDialog.Color.G;
    byte B = colorDialog.Color.B;
    SolidColorBrush brush = new SolidColorBrush(Color.FromRgb(R, G, B));
    text_06.Background = brush;
}
```



Własne okna dialogowe (okna modalne)



Własne okna dialogowe

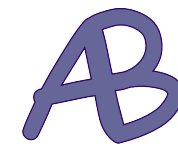
Własne okna dialogowe to zwykłe okna dziedziczące po klasie Window. Posiadają one wszystkie właściwości okien – możemy używać wszystkich kontrolki i tworzyć własny Layout. Różnica polega na sposobie ich uruchomienia.

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

Własne okna dialogowe

2. Zdefiniowanie odpowiedzi zwracanych przez okno (opcjonalnie).

Przyciskom przypisać należy role.

```
<Button IsDefault="True" Click="btn_OK_Click">Ok</Button>
```

```
<Button IsCancel="True">Cancel</Button>
```

Dodatkowo w zdarzeniu kliknięcia na przycisk potwierdzający (Ok) umieszczamy polecenie zwrócenia wartości `true` do okna nadrzędnego

```
private void btn_OK_Click(object sender, RoutedEventArgs e)
{
    this.DialogResult = true;
}
```

Nie musimy tego robić dla przycisku Cancel

Własne okna dialogowe

3. W kodzie okna głównego:

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

```
Okno01 noweOkno = new Okno01();
noweOkno.Title = "Nowe okno";
if (noweOkno.ShowDialog() == true)
    text_03.Text = "Wybrano OK";
else
    text_03.Text = "Wybrano Cancel";
```

Tworzymy zmienną referencyjną dla nowego okna.
Nazwa klasy zgodnie z nazwą jaką nadaliśmy oknu

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.

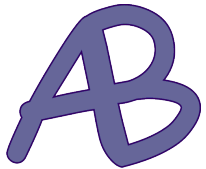
Własne okna dialogowe

4. Przyjęcie odpowiedzi okna (opcjonalnie).

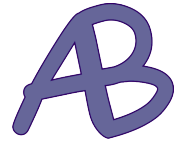
```
Okno01 noweOkno = new Okno01();  
noweOkno.Title = "Nowe okno";  
if (noweOkno.ShowDialog() == true)  
    text_03.Text = "Wybrano OK";  
else  
    text_03.Text = "Wybrano Cancel";
```

Metoda `.ShowDialog()` zwraca odpowiedź typu `true` lub `false` w zależności od tego, którym przyciskiem zostało zamknięte.

Na tej podstawie wybrać można odpowiednią akcję (w tym przykładzie wypisujemy odpowiedź w postaci tekstowej)



Odbieranie danych z okna dialogowego



Odbieranie danych z okna dialogowego

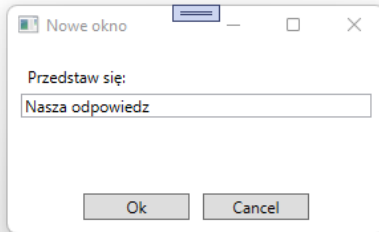
Trochę teorii

Ponieważ okna dialogowe (modalne) nie są usuwane z pamięci dopóki istnieje referencja prowadząca do okna, odebranie danych z okna jest możliwe po zamknięciu okna i odebraniu jego odpowiedzi.

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.

Komunikacja pomiędzy dwoma oknami



Założmy, że okno dialogowe posiada pole **TextBox** o nazwie **"odpowiedz"** do którego wpisać możemy dane

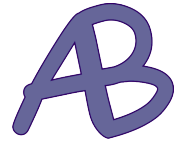
```
<TextBox Name="odpowiedz"></TextBox>
```

W kodzie okna głównego:

```
Okno01 noweOkno = new Okno01();  
noweOkno.Title = "Nowe okno";  
if (noweOkno.ShowDialog() == true)  
    text_03.Text = noweOkno.odpowiedz.Text;  
else  
    text_03.Text = "Anulowano opercję";
```

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 (nazwy obiektu nie klasy)



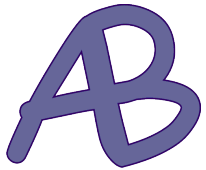
Komunikacja pomiędzy dwoma oknami

UWAGA:

Wszystkie pola (zmienne) okna dialogowego do których sięgać będzie okno nadrzędne muszą być publiczne.

W kodzie okna dialogowego:

```
public int zmienna = 10;
```



Okna niemodalne

Własne okna (niemodalne)

Własne okna niemodalne to zwykłe okna dziedziczące po klasie Window. Posiadają one wszystkie właściwości okien – możemy używać wszystkich kontrolki 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 (modalnym), 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()`.

```
Okno02 noweOkno = new Okno02();
```

```
noweOkno.Title = "Sterowanie";
```

```
noweOkno.suwak.Value = 20;
```

```
noweOkno.Show();
```

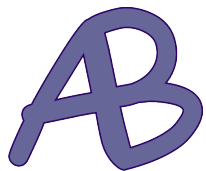
Tworzymy zmienną referencyjną dla nowego okna. Nazwa klasy zgodnie z nazwą jaką nadaliśmy przy dodawaniu nowego okna.

Następnie tworzymy obiekt - nowe okno

Przekazywanie danych do okna lub sterowanie jego kontrolkami z poziomu okna nadrzędnego, najczęściej odbywa się po utworzeniu okna ale jeszcze przed jego wyświetleniem (ale nie jest to jedyny sposób –patrz kolejne slajdy)

Wyświetlamy okno za pomocą metody `.Show()`

Okno niemodalne nie zwraca wartości.



Dwustronna komunikacja pomiędzy dwoma oknami (niemodalnymi)



Komunikacja pomiędzy dwoma oknami

Trochę teorii

Ponieważ:

- okna niemodalne nie zwracają wartości i usuwane są z pamięci po ich zamknięciu,
- wywołanie okna niemodalnego metodą `.Show()` nie zatrzymuje pracy (przetwarzania kodu) okna głównego,

odebranie danych z okna modalnego jest możliwe gdy jest ono wyświetlane.

Wszelkie dane wprowadzone do okna dialogowego w trakcie jego pracy należy przestać z okna podrzędnego do głównego.

Komunikacja pomiędzy dwoma oknami

PRZEKAZANIE DANYCH Z OKNA GŁÓWNEGO DO PODRZĘDNEGO

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

```
Okno02 noweOkno = new Okno02(this);..  
noweOkno.Title = "Sterowanie";  
noweOkno.Show();
```

```
noweOkno.suwak.Value = 20;
```

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:

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

W tym celu, w parametrze konstruktora obiektu `Okno02`, umieszczamy `this`, czyli referencję na obiekt, wewnątrz którego aktualnie jesteśmy

```
private void Button_Click_3(object sender, RoutedEventArgs e)
{
    Okno02 noweOkno = new Okno02(this);

    noweOkno.Show();
    noweOkno.Title = "Sterowanie";
    noweOkno.suwak.Value = 20;
}
```



Komunikacja pomiędzy dwoma oknami

PRZEKAZANIE DANYCH Z OKNA PODRZĘDNEGO DO GŁÓWNEGO

W kodzie okna podrzędnego:

Tworzymy referencję na obiekt klasy odpowiadającej oknu nadrzędnemu

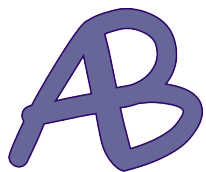
Jako parametr konstruktora dodajemy inną referencję na okno nadrzędne- tu trafi adres wysłany za pomocą **this** (poprzedni slajd)

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

```
public partial class Okno02 : Window
{
    MainWindow mainWindow;
    Odwołania: 2
    public Okno02(MainWindow adrewZwrotny)
    {
        InitializeComponent();
        mainWindow = adrewZwrotny;
    }

    1 odwołanie
    private void suwak_ValueChanged(object sender, RoutedEventArgs e)
    {
        mainWindow.text_04.Text = suwak.Value.ToString();
    }
}
```

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



Okna niemodalne –

- **Zamykanie okna przy zakończeniu programu**
- **zapobieganie otwarciu wielu instancji okna**

Zapobieganie tworzeniu wielu instancji okna

W przypadku okien otwieranych za pomocą metody `.show()` możliwe jest utworzenie wielu instancji okna, czyli otworzenie wielu kopii tego samego okna dialogowego jednocześnie.

Zwykle sytuacja ta jest błędem. Aby jej zapobiec należy dodać mechanizm kontrolujący czy okno jest już otwarte.

W oknie nadrzędnym:

- Referencje do okna dialogowego wyciągamy z wewnątrz metody i tworzymy jako pole klasy. Nadajemy jej wartość `null`.
- Okno otwieramy tylko wtedy, gdy wartość referencji wynosi `null`. W innym wypadku wiemy, że istnieje już otwarte okno dialogowe.

```
public Okno02? noweOkno = null;
1 odwołanie
private void Button_Click_7(object sender, RoutedEventArgs e)
{
    if (noweOkno == null)
    {
        noweOkno = new Okno02(this);
        noweOkno.Show();
        noweOkno.Title = "Sterowanie";
        noweOkno.suwak.Value = 20;
    }
}
```


Zapobieganie tworzeniu wielu instancji okna

W oknie dialogowym:

- W momencie zamykania okna dialogowego musi ono sięgnąć do okna nadrzędnego i zmienić referencja z powrotem na null .

1 odwołanie

```
private void Window_Closed(object sender, EventArgs e)
{
    mainWindow.noweOkno = null;
}
```

Aby było to możliwe referencja w oknie nadrzędnym musi być polem publicznym (patrz poprzedni slajd).

Zamykanie okna przy zakończeniu programu

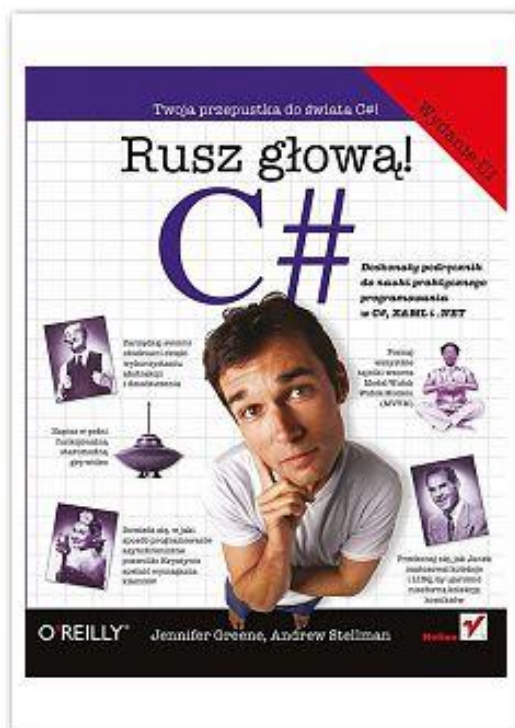
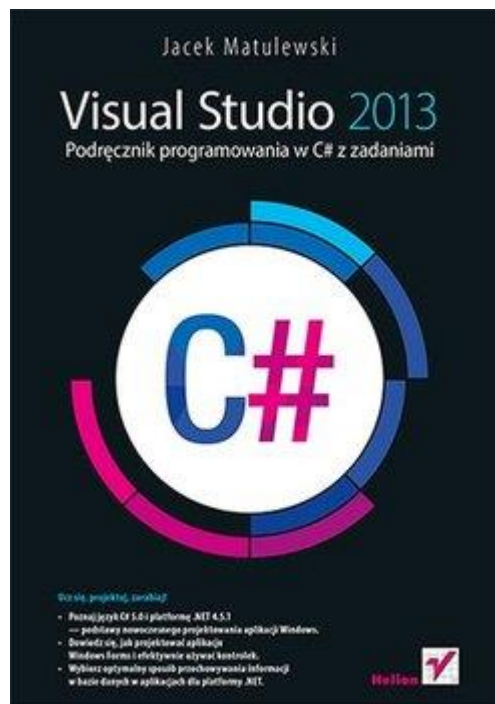
Okna niemodalne otwarte za pomocą metody `.show()` nie są automatycznie zamykane przy zamknięciu okna nadrzędnego.

Aby za zamknąć okno niemodalne automatycznie należy wydać polecenie jego zamknięcia w metodzie **Closing** okna nadrzędnego.

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (noweOkno != null)
    {
        noweOkno.Close();
    }
}
```

Uwaga - próba zamknięcia nieistniejącego okna doprowadzi do błędu. Dlatego mechanizm ten stosujemy wyłącznie w połączeniu z opisanym wyżej mechanizmem zapobiegania otwieraniu duplikatów okna.

Literatura:



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