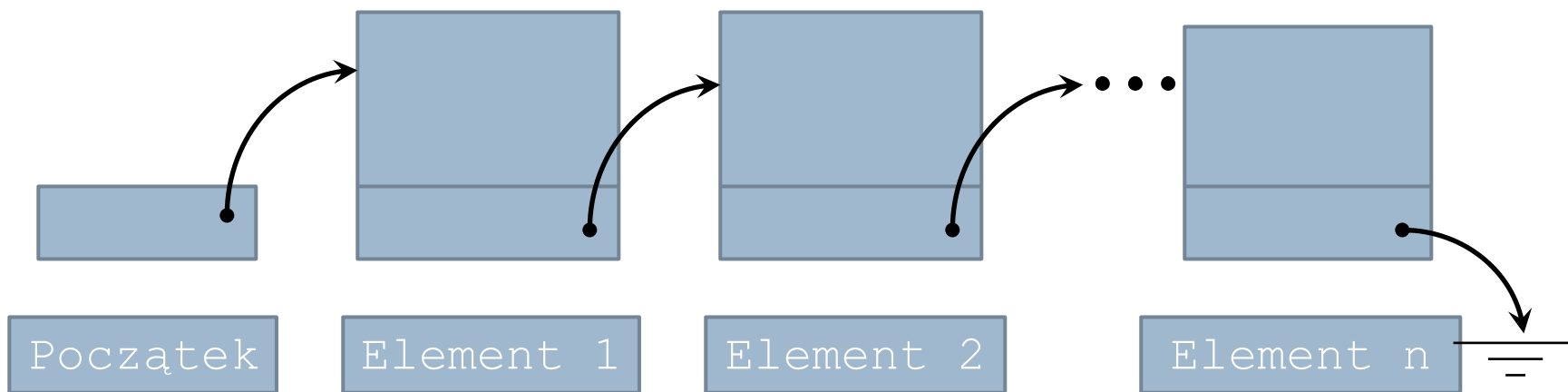


Wykład 4

- **Pliki tekstowe,**
- **Własne okna dialogowe**



Listy - przypomnienie





Kolekcja „Listy”

Lista - należy do grupy typów ogólnych (ang. generic types).

- ✓ W porównaniu z tablicą (Array) ma tą zaletę, że liczba elementów może być zmieniana już po utworzeniu listy.
- ✓ Można dodawać elementy na koniec, na początek i w środek listy.
- ✓ Można też usuwać dowolny element listy.
- ✓ Dostęp do dowolnego elementu listy możliwy jest, tak samo jak w przypadku tablicy.



Kolekcja „Listy”

Tworzenie listy:

```
List<typ> l = new List<typ>(tab. wart. inicjalizujących);
```

```
List<int> l = new List<int>();
```

```
List<String> s = new List<String>();
```

W parametrze konstruktora listy możemy podać tablicę wartości inicjalizujących.

```
List<int> l = new List<int>(new int[] {1,2,3,4,5});
```

```
List<String> s = new List<String>(new String[] {"aa","bb","cc"});
```



Podstawowe operacje na listach (na przykładzie tablicy String):

```
List<String> nazwa = new List<String>();
```

```
nazwa.Add("element");
```

- Dodawanie elementu

```
nazwa.AddRange(new String[] { "aa", "bb" });
```

- Dodanie tablicy elementów (na koniec listy)

```
nazwa.Insert(0, "aa");
```

- wstawianie elementu na wskazaną pozycję -UWAGA- nie zastępujemy tylko wstawiamy

```
nazwa.InsertRange(0, new String[] { "aa", "bb" });
```

- wstawianie listy elementu na wskazaną pozycję

```
nazwa.RemoveAt(0);
```

- usunięcie wskazanego elementu

```
nazwa.Remove("bb");
```

- usunięcie elementu o wskazanej wartości,



Kolekcja „Listy”

Podstawowe operacje na listach (na przykładzie tablicy String):

`nazwa.Clear();`

- wyczyszczenie listy

`nazwa.Sort();`

- sortowanie listy

`nazwa.Reverse();`

- odwrócenie listy

`nazwa.Count();`

- podaje liczbę elementów

`nazwa.ToArray(TablicaDocelowa);`

- eksportuje listę do tablicy.



Strumienie i Pliki





Strumienie i pliki

Strumienie są formą wymiany i transportu danych, obsługiwana przez klasy przestrzeni *System.IO*.

- ✓ Przy użyciu strumieni można komunikować się z konsolą oraz operować na danych znajdujących się w pamięci komputera, w plikach.
- ✓ Np., strumień może być plikiem, pamięcią operacyjną lub współdzielonym zasobem sieciowym.



Strumienie i pliki

Klasy służące do operowania na plikach i katalogach

Klasa	Opis
Directory	Służy do operowania na katalogach (przenoszenie, kopiowanie).
File	Klasa umożliwia tworzenie, usuwanie oraz przenoszenie plików.
Path	Służy do przetwarzania informacji o ścieżkach (do katalogów i plików)
DirectoryInfo	Podobna do klasy Directory. Stosujemy, jeżeli dokonujemy wielu działań na katalogach, gdyż nie wykonuje testów bezpieczeństwa.
FileInfo	Podobna do klasy File. Stosujemy, jeżeli dokonujemy wielu działań na plikach, gdyż nie wykonuje testów bezpieczeństwa.



Strumienie i pliki

Przykładowe operacje na katalogu

W naszym przykładzie katalog „test”– sprawdzamy, czy katalog istnieje i tworzymy go gdy nie istniał.

```
if (folderBrowserDialog1.ShowDialog()==DialogResult.OK)
{
    if (!Directory.Exists(folderBrowserDialog1.SelectedPath+"test"))
    {
        Directory.CreateDirectory(folderBrowserDialog1.SelectedPath + "test");
    }
}
```



Strumienie i pliki

Tworzenie i usuwanie plików

```
File.CreateText("C:\\plik.txt");
```

```
if (!File.Exists("C:\\plik.txt"))  
{  
    StreamWriter sw = File.CreateText("C:\\plik.txt");  
  
    sw.WriteLine("Witaj świecie");  
    sw.Close();  
}
```

```
File.Delete("C:\\plik.txt");
```

Tworzy nowy plik gotowy do zapisu tekstu z kodowaniem UTF-8.

Aby zapisać tekst do pliku można skorzystać z klasy `StreamWriter`, której obiekt jest zwracany przez metodę `CreateText()`:

Kasowanie pliku

Strumienie i pliki

Kopiowanie i przenoszenie plików

```
string src = "C:\\test.txt";  
string dst = "C:\\kopiatestu.txt";
```

```
if (!File.Exists(dst))  
{  
    File.Copy(src, dst);  
}
```

Kopiowanie pliku pod nową nazwą

```
string src = "C:\\test.txt";  
string dst = "D:\\test.txt";
```

```
if (!File.Exists(dst))  
{  
    File.Move(src, dst);  
}
```

Przenoszenie pliku - w tym przykładzie z dysku c: na dysk d:



Strumienie i pliki

Strumienie

Do odczytywania i zapisywania danych do strumieni używamy odrębnych klas — **StreamReader** oraz **StreamWriter**.

W przypadku danych binarnych są to odpowiednio klasy **BinaryWriter** i **BinaryReader**

Zaczynamy od utworzenia egzemplarza klasy **FileStream**.

Jej konstruktor wymaga podania trzech parametrów:

1. *ścieżki do pliku,*
2. trybu otwarcia pliku,
3. trybu dostępu do pliku.

```
FileStream fs = new FileStream("C:\\test.txt",  
                               FileMode.OpenOrCreate,  
                               FileAccess.ReadWrite);
```



Strumienie i pliki

Aby odczytać zawartość w pliku tekstowym, należy też utworzyć egzemplarz klasy **StreamReader**.

W parametrze jego konstruktora należy przekazać obiekt klasy **FileStream**

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    FileStream fs = new FileStream(openFileDialog1.FileName,
    FileMode.Open, FileAccess.Read);
    try
    {
        StreamReader sr = new StreamReader(openFileDialog1.FileName);
        textBox1.Text = sr.ReadToEnd();
        sr.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

Cała zawartość pliku
odczytać możemy za
pomocą metody **ReadToEnd**

Jednak cały plik zapisany w pojedynczym łańcuchu jest trudny do przetwarzania

Strumienie i pliki

Częściej odczytujemy plik wiersz po wierszu.

```
while (!sr.EndOfStream)
{
    textBox1.Text += sr.ReadLine();
}
```

Odczyt pojedynczej linii

Zawartość pliku można
zapisać w tablicy – jeden
wiersz w każdej komórce.

```
String[] tab = new String[100];
int i = 0;
while (!sr.EndOfStream)
{
    tab[i] = sr.ReadLine();
    i++;
}
```

**Częściej jednak zapisujemy plik do listy – puste pola tablicy mogą
sprawiać kłopoty.**

Strumienie i pliki

Wyświetlenie pliku w kontrolce **textBox**

**TextBox**

Kontrolka **textBox** posiada pole **textBox.Text**, gdzie zapisać możemy pojedynczy łańcuch – to z niego korzystaliśmy dotychczas.

Jeżeli ustawimy własność kontrolki **multiline** na **true** możemy korzystać też ze struktury **textBox.Lines**, która jest tablicą zmiennych **String** – jedno pole jedna linijka.

Stąd, jeżeli mamy tablicę łańcuchów możemy ją łatwo wyświetlić w polu **textBox**.

```
String[] tab = new String[] { "aaa", "bbb", "ccc" };  
textBox1.Lines = tab;
```


Strumienie i pliki

Aby zapisać wartość w pliku tekstowym, należy utworzyć egzemplarz klasy **StreamWriter**.

W parametrze jego konstruktora należy przekazać obiekt klasy **FileStream**

```
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
    FileStream fs = new FileStream(saveFileDialog1.FileName,
        FileMode.OpenOrCreate, FileAccess.ReadWrite);
    try
    {
        StreamWriter sw = new StreamWriter(fs);
        sw.WriteLine("Hello World!");
        sw.WriteLine("Bye!");
        sw.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

Do zapisu tekstu użyć
można metody **WriteLine()**

```
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
    FileStream fs = new FileStream(saveFileDialog1.FileName,
        FileMode.OpenOrCreate, FileAccess.ReadWrite);
    try
    {
        StreamWriter sw = new StreamWriter(fs);
        int ile=0;
        String[] tab = new String[100];
        ile = textBox1.Lines.Count();
        tab = textBox1.Lines;
        for (int i = 0; i < textBox1.Lines.Count(); i++)
            sw.WriteLine(tab[i]);
        sw.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

AB

Przykład:

zapis zawartości
pola textBox do
pliku



Przykład:

Wykonamy prosty notatnik obsługujący pliki tekstowe. Notatnik posiadał będzie następujące funkcje:

- odczyt z pliku,
- zapis do pliku,
- zamian koloru fontu i tła
- zmiana rozmiaru fontu,
- obsługa schowka

Notatnik

Notatnik zbudujemy w oparciu o zwykły komponent textBox.

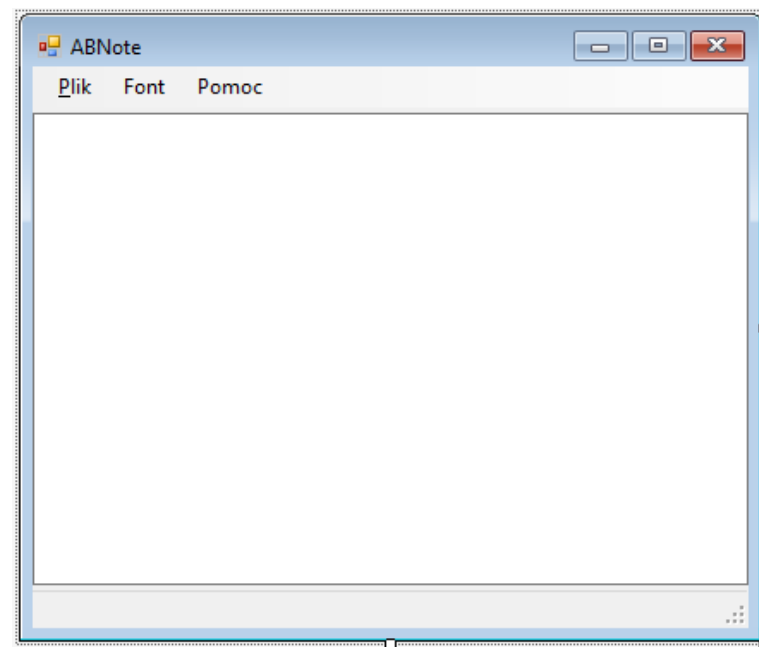
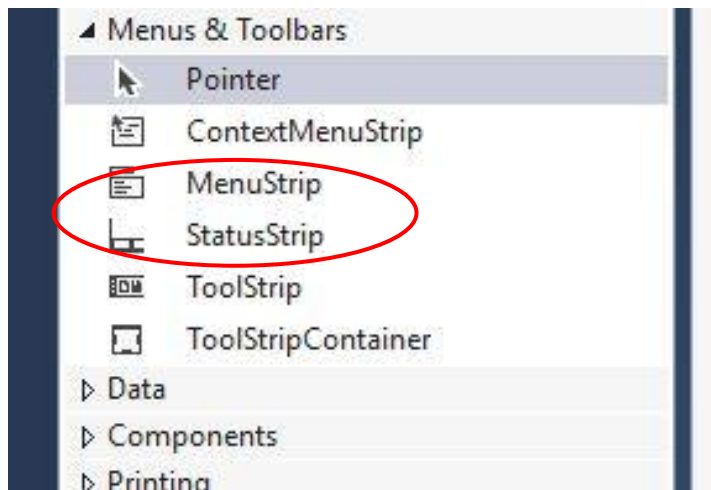
Po ustawieniu pole `.Multiline` na true, może on wyświetlać teksty wielolinijkowe.



`textBox`, oprócz pole `.Text` w którym zapisać można pojedynczą zmienną string, posiada strukturę `.Lines`, która jest tablicą string-ów (jedno pole jedna linia wyświetlona w textBox-ie).

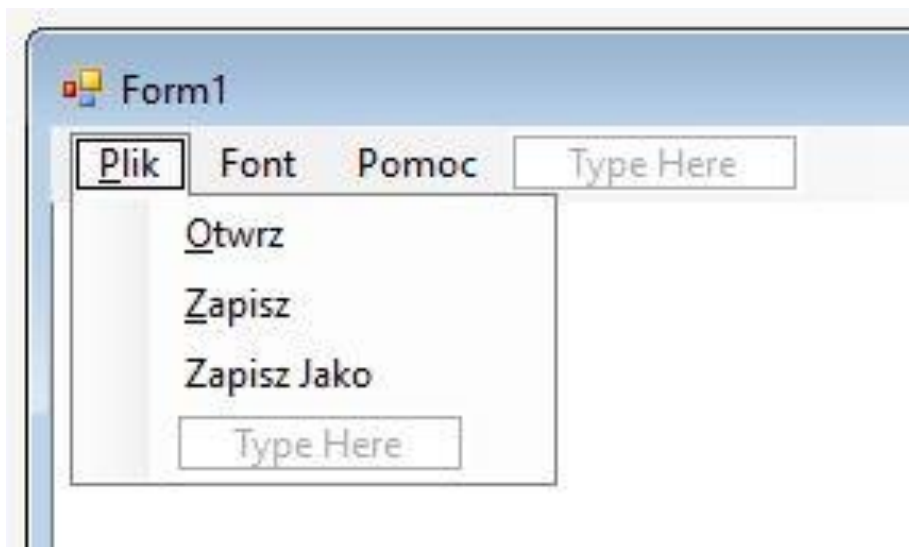
Notatnik - layout

Oprócz textBox wstawiamy komponenty menuStrip i StatusStrip



Notatnik - layout

Kolejnym krokiem jest wypełnienie pozycji menu

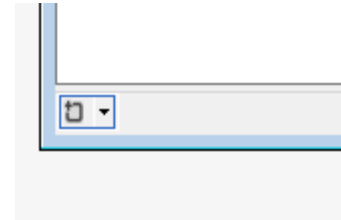


Poprzedzając nazwę pozycji menu znakiem & (np. &Plik) wybieramy aktywny klawisz.

Można też dodać skróty klawiaturowe w oknie właściwości pozycja [ShortcutKeys](#).

Notatnik - layout

Przygotowujemy pasek statusu



Kontrolka statusStrip posiada strukturę .Items, do której dodać możemy kilka typów obiektów. W naszym przypadku dodajemy Label, który wyświetli nazwę pliku.

Tekst wstawimy w sposób następujący:

```
statusStrip1.Items[0].Text = "tekst na pasku statusu";
```

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

[0] oznacza pierwszy element na liści Items



Notatnik - layout

Teraz dodamy i oprogramujemy obiekt klasy `openFileMenu` – aby wczytać nazwę pliku.

Obsługę okna `openFileDialog` umieszczamy w zdarzeniu kliknięcia na pozycję menu (wystarczy kliknąć dwukrotnie na pozycję „Plik->Otwórz”

```
private void otwrzToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog()==DialogResult.OK)
    {
        statusStrip1.Items[0].Text = openFileDialog1.FileName;
        textBox1.Lines = CzytajPlik(openFileDialog1.FileName);
    }
}
```

Metodę `CzytajPlik()` musimy jeszcze napisać. Powinna ona zwrócić tablicę łańcuchów, którą wstawiamy do pola `Lines` w `textBox1`



Notatnik - layout

```
public static string[] CzytajPlik(string nazwaPliku)
{
    List<string> tekst = new List<string>();
    try
    {
        using (StreamReader sr = new StreamReader(nazwaPliku))
        {
            string wiersz;
            while (!sr.EndOfStream)
            {
                wiersz = sr.ReadLine();
                tekst.Add(wiersz);
            }
        }
        return tekst.ToArray();
    }
    catch (Exception e)
    {
        MessageBox.Show("Błąd odczytu pliku " + nazwaPliku + " (" + e.Message + ")");
        return null;
    }
}
```

Odczyt linii z pliku i
zapis dodawanie ich
do listy.
Patrz poprzedni wykład



Zapis do pliku

Dodamy i oprogramujemy obiekt klasy SaveFileMenu – aby wybrać nazwę pliku.

Obsługę okna saveDialogFile umieszczamy w zdarzeniu kliknięcia na pozycję menu (wystarczy kliknąć dwukrotnie na pozycję „Plik->Zapisz jako”

```
private void zapiszJakoToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        statusStrip1.Items[0].Text = saveFileDialog1.FileName;
        ZapiszDoPliku(saveFileDialog1.FileName, textBox1.Lines);
    }
}
```

Metodę ZapiszDopliku()() musimy jeszcze napisać. Nie zwraca ona żadnej wartości, ale przyjmuje dwa parametry – nazwę pliku i tablicę łańcuchów do zapisania w pliku. Tablica bierzemy ze struktury lines pola textBox.

Zapis do pliku

```
public static void ZapiszDoPliku(string nazwaPliku, string[] tekst)
{
    using (StreamWriter sw = new StreamWriter(nazwaPliku))
    {
        foreach (string wiersz in tekst)
            sw.WriteLine(wiersz);
    }
}
```

Można też tak:

```
public static void ZapiszDoPliku(string nazwaPliku, string[] tekst)
{
    using (StreamWriter sw = new StreamWriter(nazwaPliku))
    {
        for (int i = 0; i < tekst.Length; i++)
            sw.WriteLine(tekst[i]);
    }
}
```



Obsługa schowka

Obsługa schowka systemowego dla komponentu TextBox jest prosta.

Posiada on gotowe metody wymiany danych ze schowkiem.

```
textBox1.Copy( );  
textBox1.Cut( );  
textBox1.Paste( );  
textBox1.SelectAll( );  
textBox1.Undo( );
```

Mamy do dyspozycji także pole „SelectedText” zawierające tekst zaznaczony.

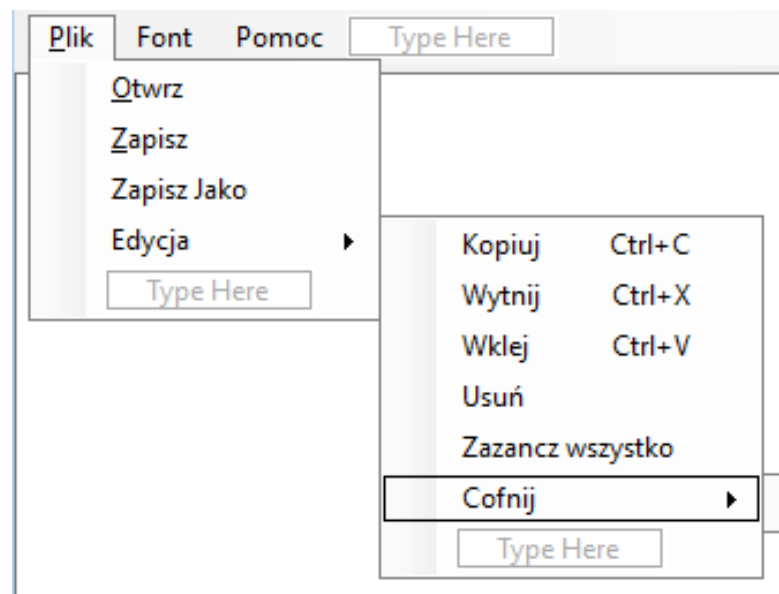
Możemy na przykład wykasować zaznaczenie:

```
textBox1.SelectedText = "";
```

Obsługa schowka

Do menu dodajemy submenu Edycja, np. tak:

Następnie oprogramujemy zdarzenia kliknięcia na menu (zdarzenie **Click**).



Obsługa schowka

```
private void kopiujToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Copy();
}
private void wytnijToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Cut();
}
private void wklejToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Paste();
}
private void usuńToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.SelectedText = "";
}
private void zazanczWszystkoToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.SelectAll();
}
private void cofnijToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Undo();
}
```



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.

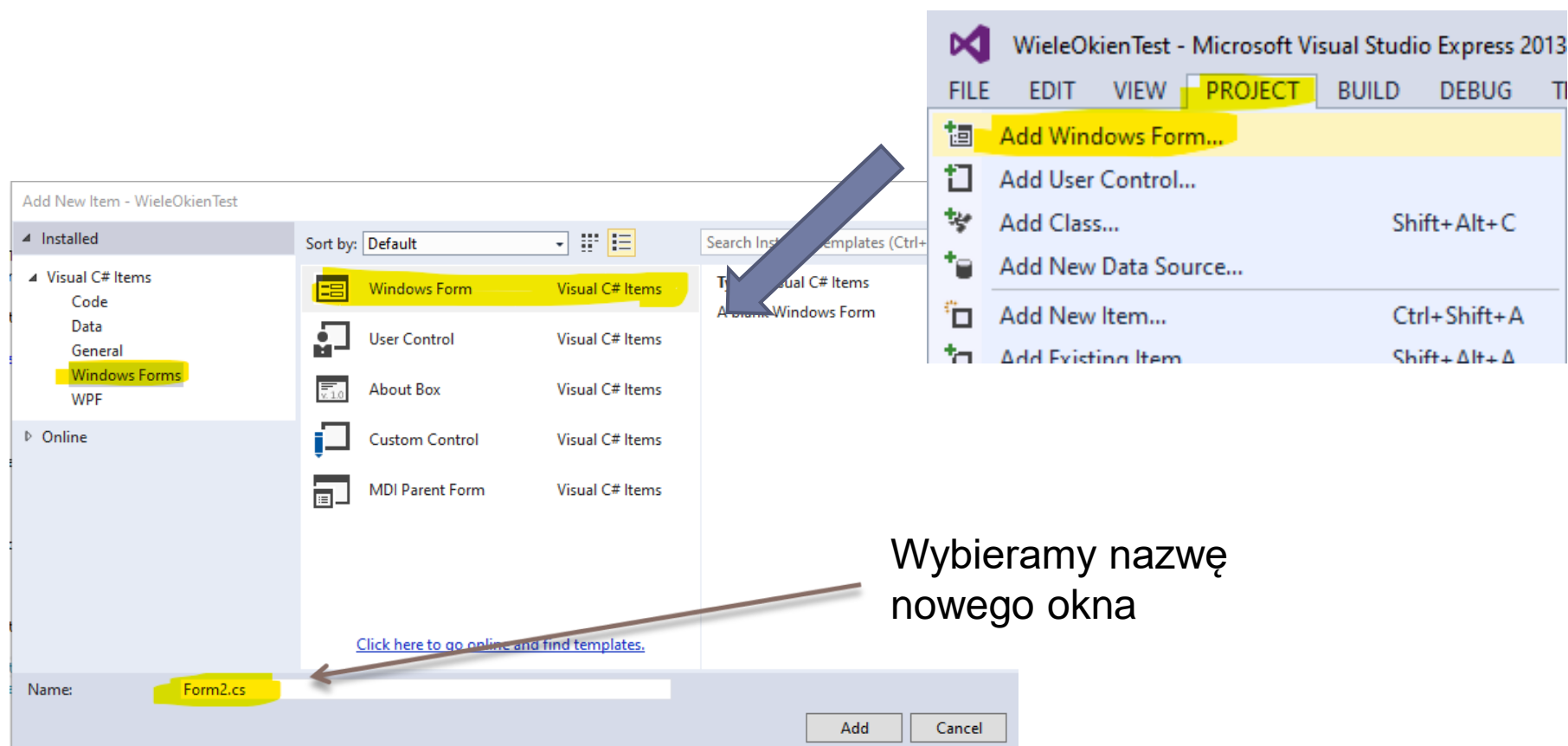
Okno tworzymy w następujących etapach:

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).



WieleOkienTest - Microsoft Visual Studio Express 2013

FILE EDIT VIEW **PROJECT** BUILD DEBUG T

- + Add Windows Form...
- + Add User Control...
- + Add Class... Shift+Alt+C
- + Add New Data Source...
- + Add New Item... Ctrl+Shift+A
- + Add Existing Item Shift+Alt+A

Add New Item - WieleOkienTest

Sort by: Default

Visual C# Items

- Windows Form Visual C# Items
- User Control Visual C# Items
- About Box Visual C# Items
- Custom Control Visual C# Items
- MDI Parent Form Visual C# Items

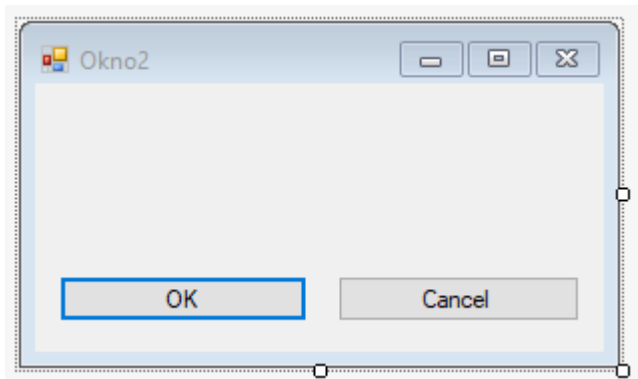
Name: Form2.cs

Add Cancel

Wybieramy nazwę nowego okna

Własne okna dialogowe

2. Zdefiniowanie odpowiedzi zwracanych przez okno (opcjonalnie).



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

Misc	
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.

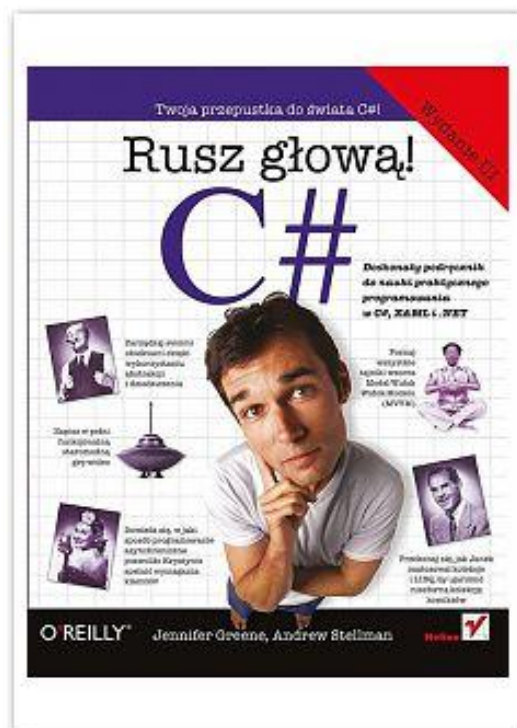
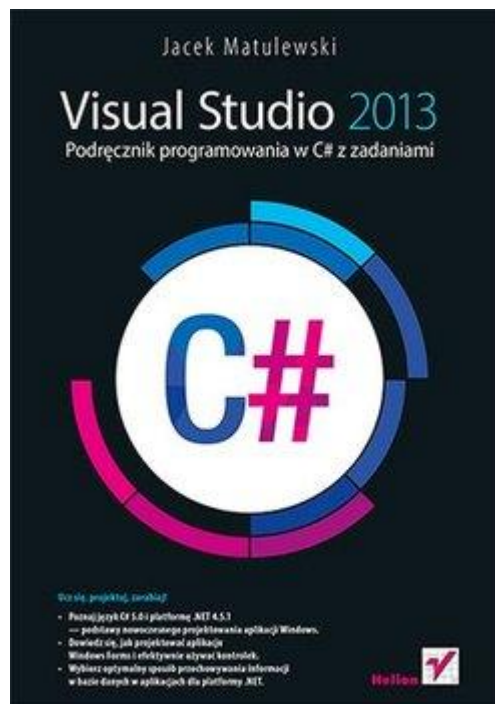
Własne okna dialogowe

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)



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