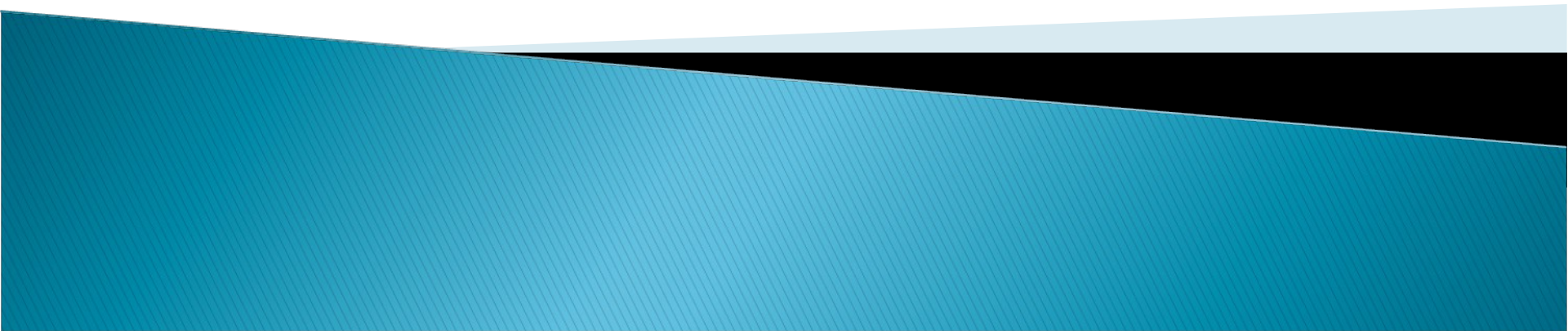
# Windows Forms

Przygotowała: Ewelina Bendlin



# Spis treści

# Co to jest Windows Forms?

**Windows Forms** (często nazywana w skrócie *WinForms*) jest biblioteką wizualną, umożliwiającą zaprojektowanie graficznego interfejsu użytkownika (GUI, czyli *Graphical User Interface*). WinForms jest częścią środowiska .NET Framework. Windows Forms jest zbiorem elementów służących do tworzenia aplikacji okienkowych.

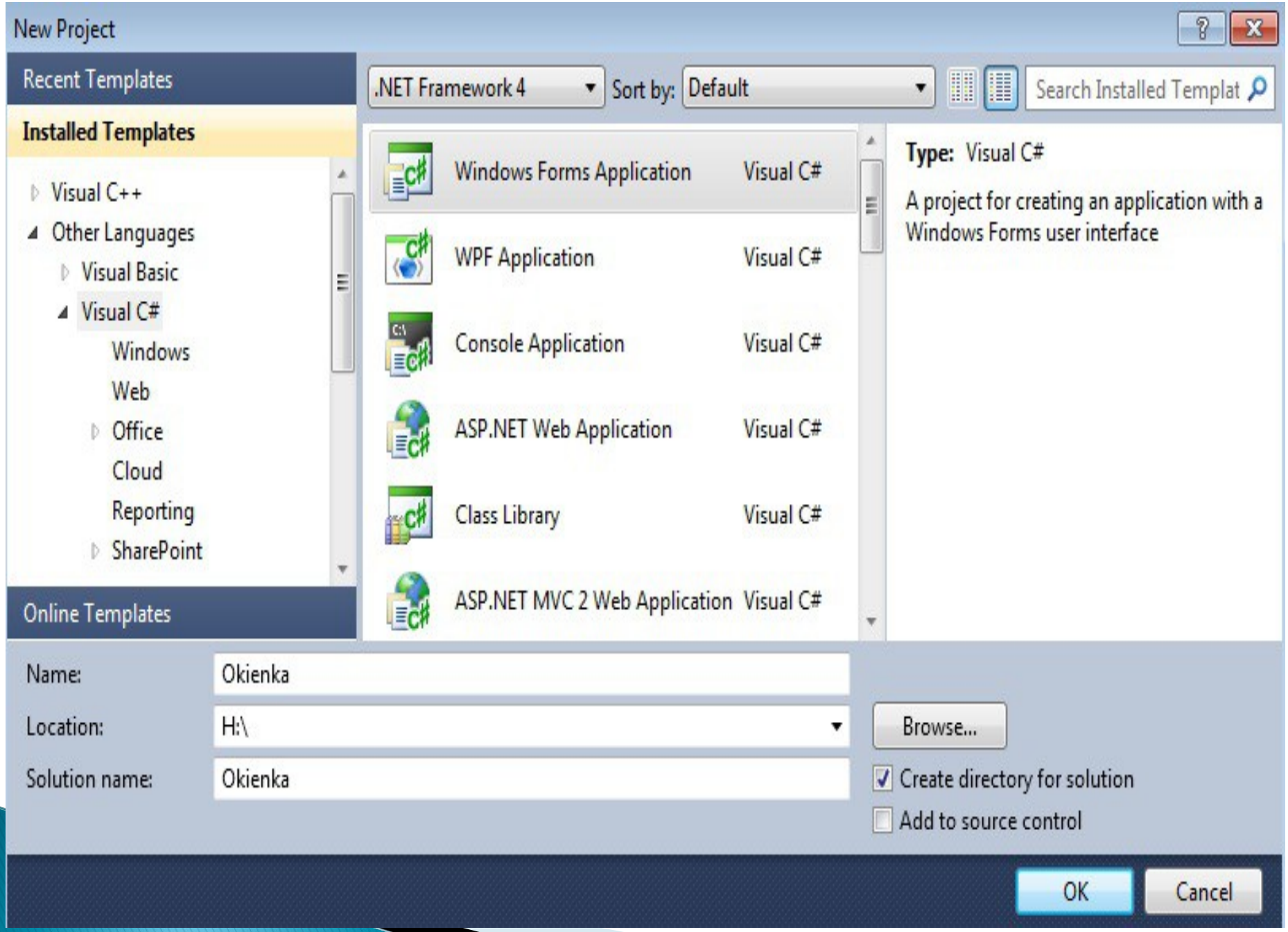
Taką aplikację nazywamy *sterowaną zdarzeniami*. Takowa aplikacja przez większość czasu oczekuje na czynności użytkownika programu (np. wciśnięcie przycisku czy wypełnienie pola formularza).

# Generowanie aplikacji Windows Forms

Aby wygenerować szablon projektu aplikacji należy:

1.Utworzyć nowy projekt wybierając opcję **Project** z menu **File->New**.
2.Wybrać **Visual C#**.
3.Wybrać wzorzec **Windows Forms Application**.
4.W obszarze **Name** wpisać nazwę projektu.
5.W obszarze **Location** wskazać ścieżkę katalogu głównego, w którym będzie umieszczony katalog projektu.
6.Nacisnąć przycisk **OK**.

Po wygenerowaniu szablonu aplikacji okienkowej zostanie utworzony projekt zawierający kod pozwalający na wyświetlenie okna głównego aplikacji.

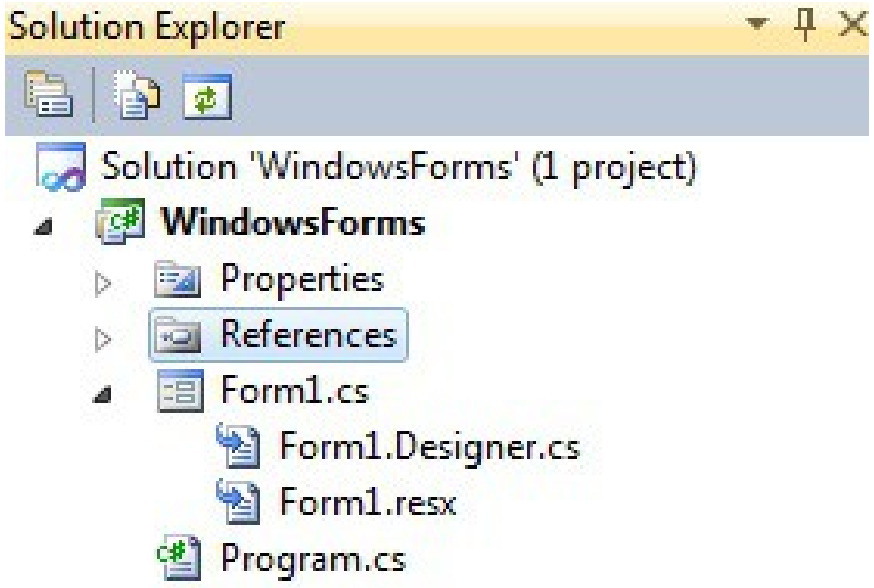

New Project
Recent Templates
Installed Templates
Visual C++
Other Languages
Visual Basic
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Online Templates
.NET Framework 4
Sort by:
Default
Search Installed Templat
Windows Forms Application
Visual C#
WPF Application
Visual C#
Console Application
Visual C#
ASP.NET Web Application
Visual C#
Class Library
Visual C#
ASP.NET MVC 2 Web Application
Visual C#
Type: Visual C#
A project for creating an application with a Windows Forms user interface
Name:
Okienka
Location:
H:\
Browse...
Solution name:
Okienka
Create directory for solution
Add to source control
OK
Cancel

# Co zostało utworzone?

Form1.cs

Program.cs

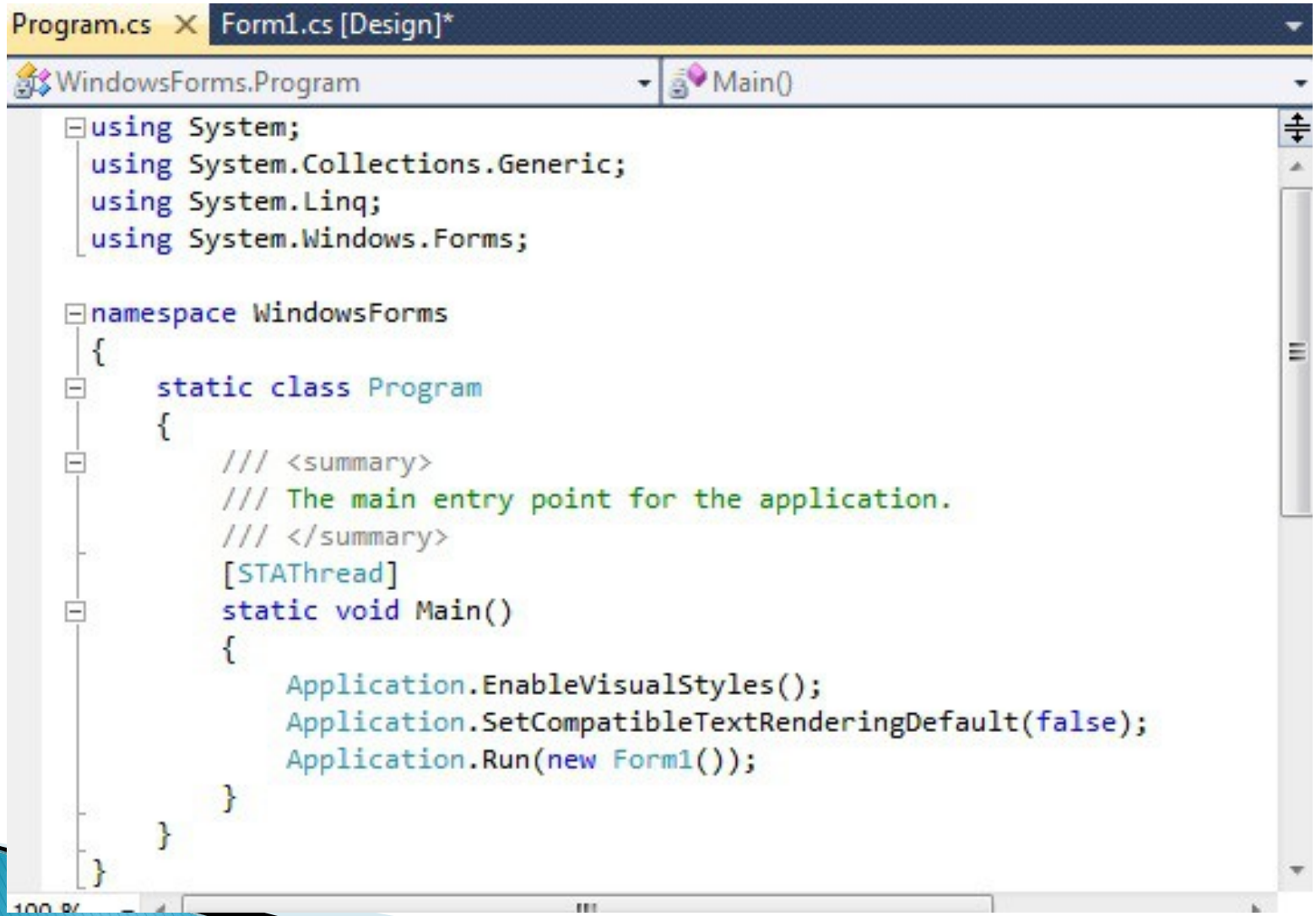
Form1.Designer.cs



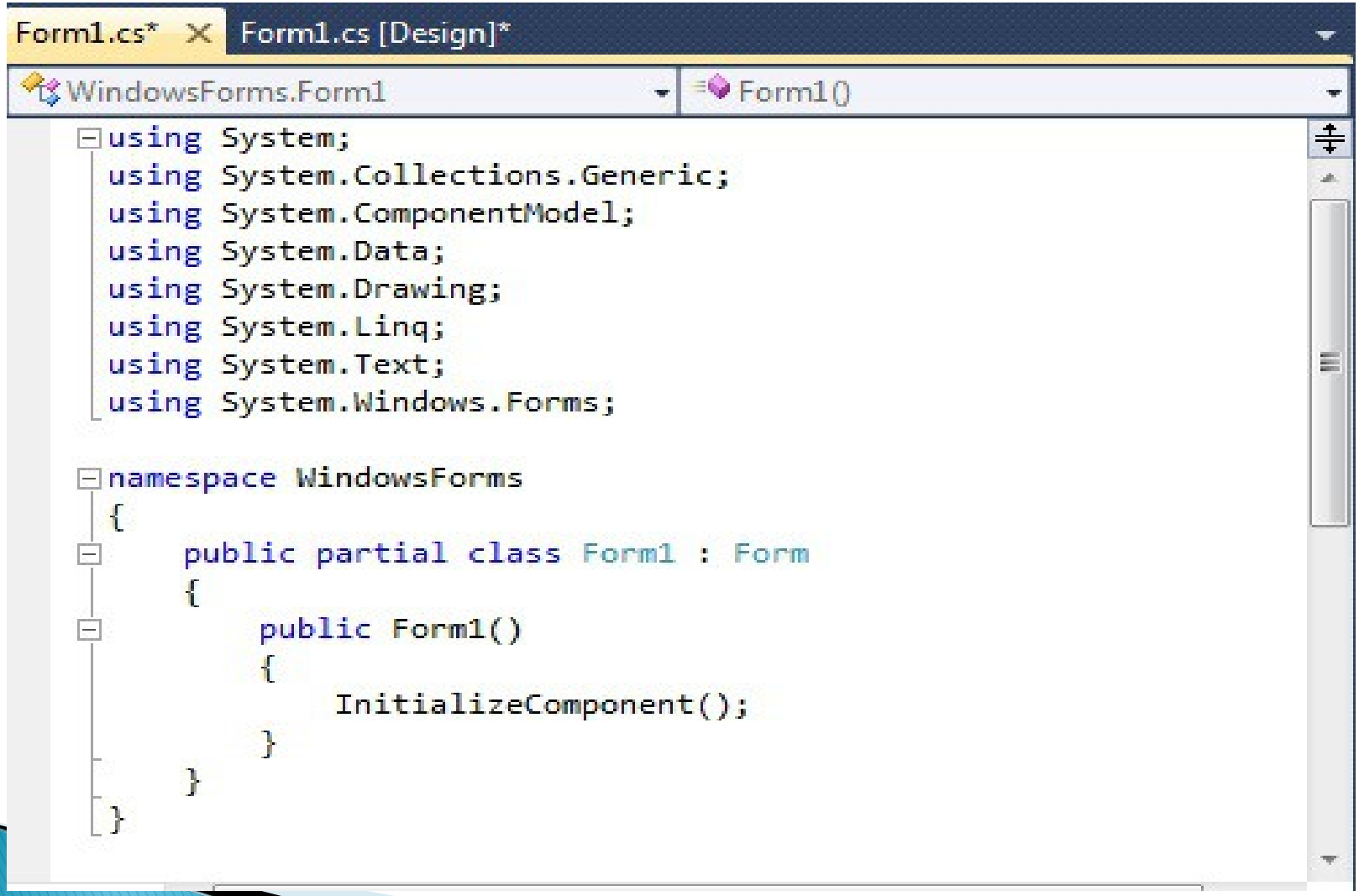

Ten plik zostanie utworzony, po dodaniu dowolnej kontrolki. Zawiera on wszystkie zasoby (grafikę, wideo, dźwięk) skojarzone z formularzem Form1.

Form1.resx

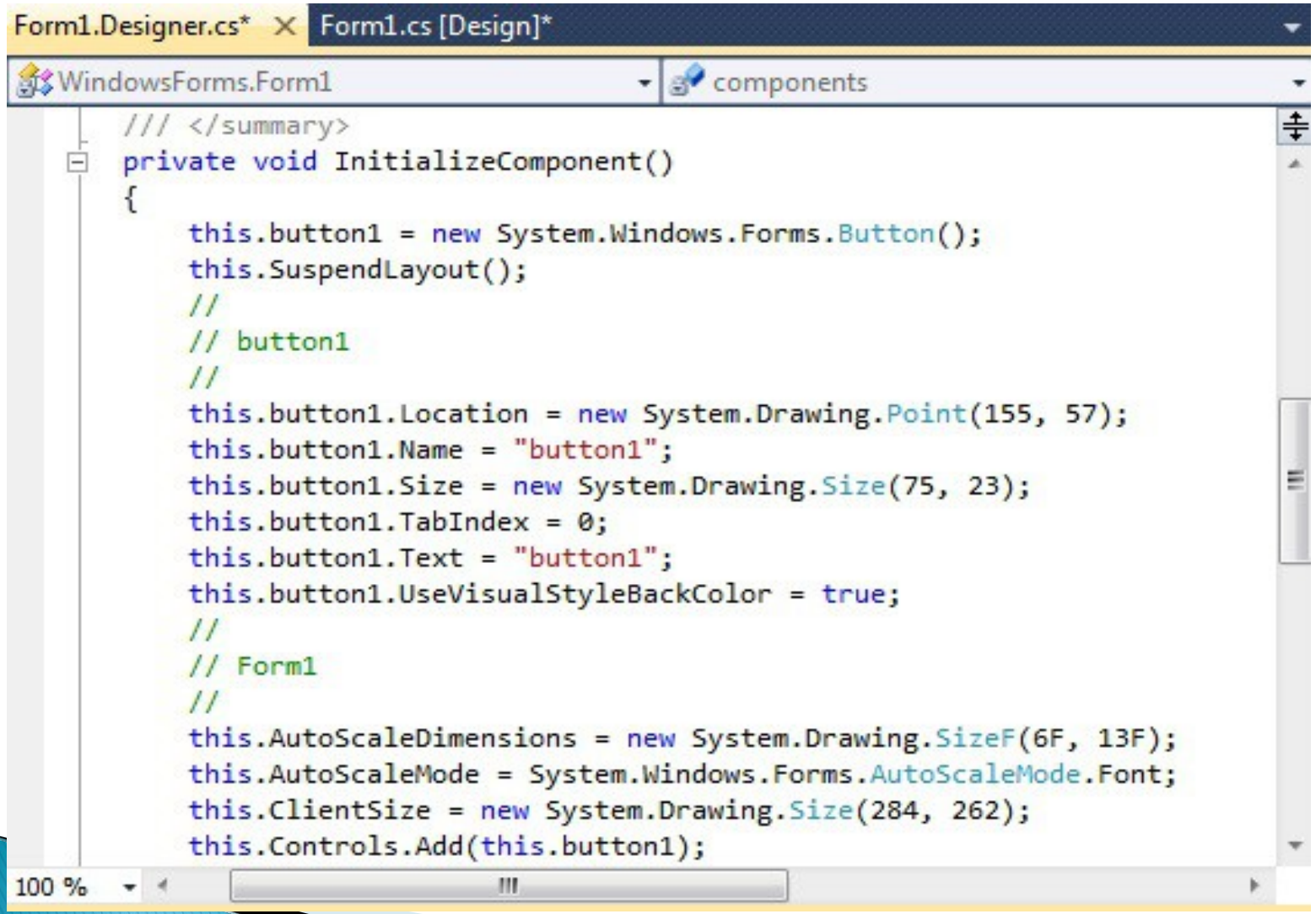
# Program.cs


Program.cs
Form1.cs [Design]*
WindowsForms.Program
Main()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace WindowsForms
{
static class Program
{
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form1());
}
}
}

# Form1.cs


Form1.cs*
Form1.cs [Design]*
WindowsForms.Form1
Form1()
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 WindowsForms
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
}
}

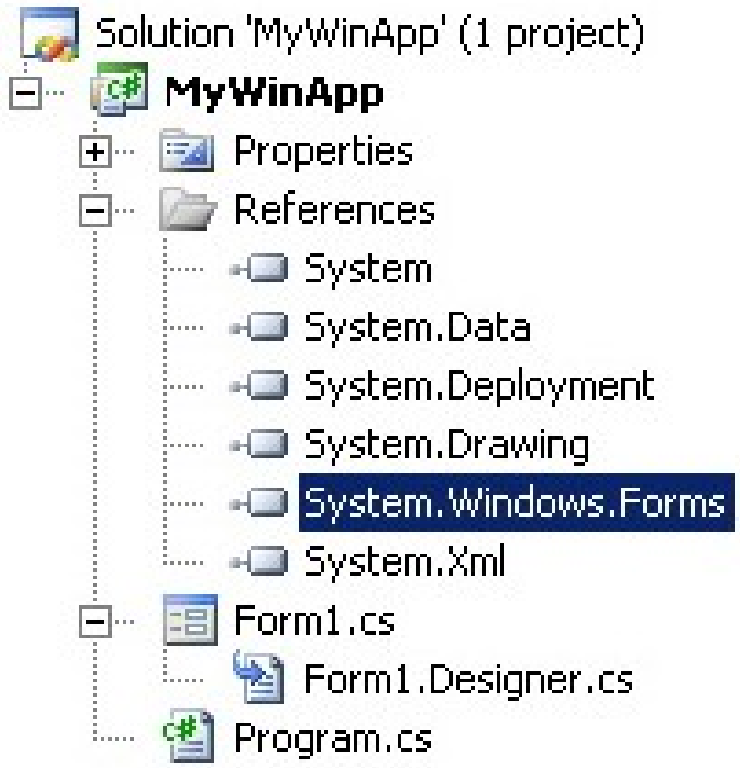
# Form1.Designer.cs



# Biblioteka *Windows Forms*

Przestrzeń nazw System.Windows.Forms znajduje się w podzespole o tej samej nazwie. Jest to podstawowa biblioteka służąca do projektowania wizualnego.

Podczas tworzenia nowego projektu WinForms odpowiednie podzespoły umożliwiające jego prawidłową kompilację są włączane automatycznie. Możesz to sprawdzić, rozwijając gałąź „References „w oknie „Solution Explorer”



Najważniejszą przestrzenią nazw w podzespole *System.Windows.Forms.dll* jest ta o takiej samej nazwie (czyli System.*Windows*.*Forms*), włączana do każdego projektu typu WinForms. Zawiera ona podstawowe klasy obsługi aplikacji, ale również wszystkie klasy komponentów. Każdy komponent jest jednocześnie klasą. Taka klasa zawiera na przykład: właściwości, metody, zdarzenia, struktury.

# Podstawowe klasy

Oczywiście cała biblioteka WinForms zbudowana jest na mechanizmie dziedziczenia, toteż istnieje pewien zestaw klas bazowych dla wszystkich komponentów.

# Właściwości klasy

Klasa Control zawiera całe mnóstwo właściwości publicznych oraz chronionych, które dziedziczone są w klasach potomnych. Właściwości te służą do określania podstawowych cech komponentów, takich jak położenie, rozmiar, kolor itp. Przykładowo:

| BackColor | Umożliwia określenie koloru tła dla komponentu. |
|---|---|
| Enabled | Określa, czy komponent będzie wyłączony (kontrolka wówczas nie reaguje na zdarzenia). |
| Font | Umożliwia ustawienie czcionki |
| Location | Właściwość typu Point umożliwia określenie położenia dla komponentu. |
| Name | Bardzo ważna właściwość. Określa nazwę komponentu. |
| Size | Właściwość typu Point umożliwia określenie rozmiaru kontrolki. |
| Visible | Umożliwia określenie, czy komponent będzie widoczny, czy też nie. |

## System.*Windows*.*Forms*.Control

Klasa Control, która jest bazowa dla każdej wizualnej kontrolki biblioteki WinForms. Co oznacza, że kontrolka jest wizualna? Przede wszystkim kontrolki tego typu tworzą interfejs aplikacji, są widoczne podczas jej działania. Dodatkowo mają one możliwość reagowania na zdarzenia użytkownika, takie jak naciskanie klawiszy klawiatury czy myszy.

# System.Windows.Forms.Application

Klasa Application ma duże znaczenie dla naszego programu. Zawiera bowiem bardzo ważną metodę odpowiedzialną za wyświetlenie głównego formularza projektu.
(Wywołanie: Application.Run(new Form1());)

Klasa Application ma wiele metod oraz właściwości odpowiadających za obsługę tzw. *komunikatów* . Klasa zawiera jednak dwie metody, które mogą okazać się bardzo przydatne. Mam tu na myśli metodę Exit(), umożliwiającą zakończenie pracy oraz Restart(), która ponownie uruchamia aplikację.

# Praca z formą

Forma jest podstawowym elementem interfejsu aplikacji okienkowej Windows. Jest to projekt okna, które zostanie wyświetlone w celu prezentacji danych użytkownikowi oraz wczytywaniu danych od użytkownika.

Po wygenerowaniu szablonu aplikacji okienkowej, zawsze tworzona jest forma podstawowa. Na formie tej w trakcie projektowania interfejsu można umieszczać szereg różnych kontrolek (np.: menu, paski narzędzi, przyciski itd..) oraz przypisywać im określone zachowania (np.: reakcję na naciśnięcie przycisku czy wybranie opcji z menu).

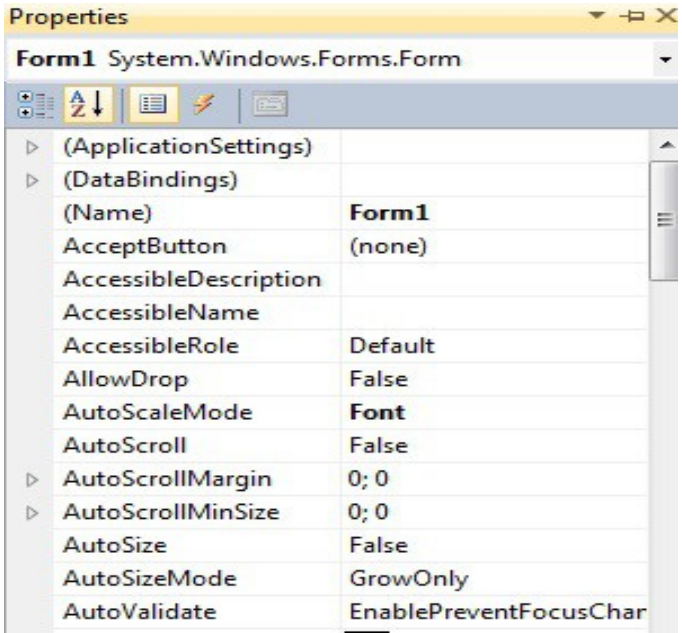
Bazową klasą dla każdej formy jest klasa **Form,** znajdująca się w przestrzeni **System.Windows.Forms**. (klasa Form dziedziczy z klasy **Control**)

# Każda forma posiada:

1. **Właściwości** – pozwalające na zmianę wyglądu formy.
2. **Metody** – pozwalające na zdefiniowanie zachowania formy.
3. **Zdarzenia** – pozwalające na interakcję z użytkownikiem.

# Właściwości formy

Forma posiada szereg właściwości pozwalających na zmianę jej wyglądu. Właściwości form ustawia się za pośrednictwem okna **Properties.** Znając nazwę właściwości, którą chcemy zmienić, należy kliknąć na jej nazwę i po prawej stronie wpisać lub wybrać z listy, określoną wartość właściwości. Właściwości okna mogą być wyświetlane w grupach funkcyjnych ( ikona ) lub w porządku alfabetycznym (ikona ).



# Metody formy

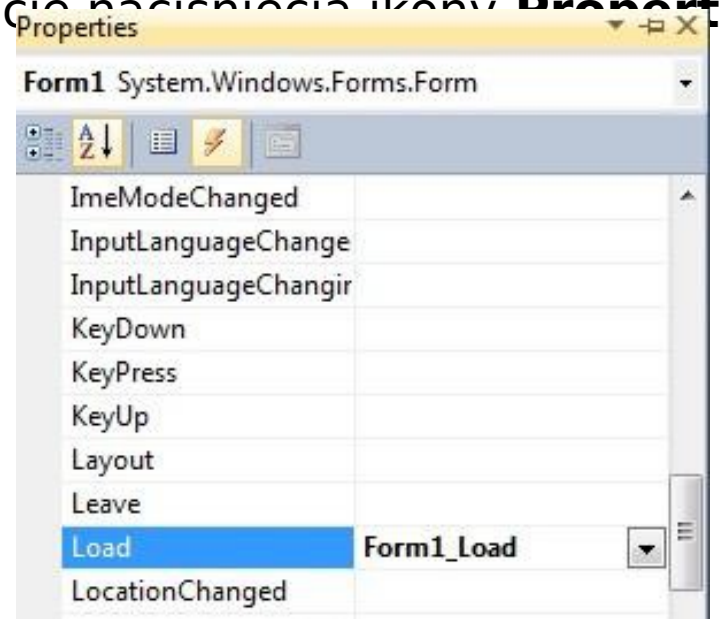
Forma posiada szereg metod, które pozwalają na zdefiniowanie zachowania formy. Poniższa tabela zawiera zestawienie najczęściej używanych metod:

| Metoda | Opis |
|---|---|
| ***Activate*** | Aktywuje formę. |
| ***Close*** | Zamyka formę. |
| ***Hide*** | Ukrywa formę. |
| ***Refresh*** | Wymusza odświeżenie (odrysowanie) całej formy i jej kontrolek. |
| ***Show*** | Pokazuje formę. |
| ***ShowDialog*** | Pokazuje formę jako modalne okno dialogowe. |
| ***Update*** | Wymusza odrysowanie widocznej części formy. |

# Obsługa zdarzeń

Forma posiada listę zdarzeń, które mogą zostać powiązane z metodą reagującą na wystąpienie określonego zdarzenia (np.: pojawienie się okna). Dodawanie metod reagujących na zdarzenie następuje za pośrednictwem okna **Properties.** Okno to pozwala zarówno na zmianę właściwości, jak i przypisywanie zdarzeniom metod. Tryb edycji właściwości jest aktywny w momencie naciśnięcia ikony **Properties**

natomiast tryb edycji zdarzeń w momencie naciśnięcia ikony **Events**



# Przykład

Przykład ustawienia tytułu okna:

```
private void Form1_Load(object sender, EventArgs e)
 {
        this.Text = "Aplikacja testowa";
 }
```

Zdarzenie może występować z określonymi parametrami, które są przekazywane do metody je obsługującej w postaci argumentów wejściowych.

Do argumentów tych należą:

- Obiekt, który wysłał zdarzenie (sender),
- Obiekt (klasa **EventArgs** lub klasa potomna tej klasy) zawierający parametry wywołania zdarzenia (np.: dla zdarzenia **KeyDown** jest to klasa **KeyEventArgs**, która zawiera parametry dotyczące kodu naciśniętego klawisza).

# Tworzenie okien

Zdarzają się sytuacje gdy jedno okno nam nie wystarcza. Środowisko visual C# pozwala nam stworzyć nowe okno w bardzo prosty sposób.

1.Kliknij prawym przyciskiem myszy na nazwie projektu z menu "*solution explorer*"

2.Wybierz *add* → *new form*, albo (jeśli twoja wersja visual C# nie ma takiej opcji) *add* → *new item* → *windows form*

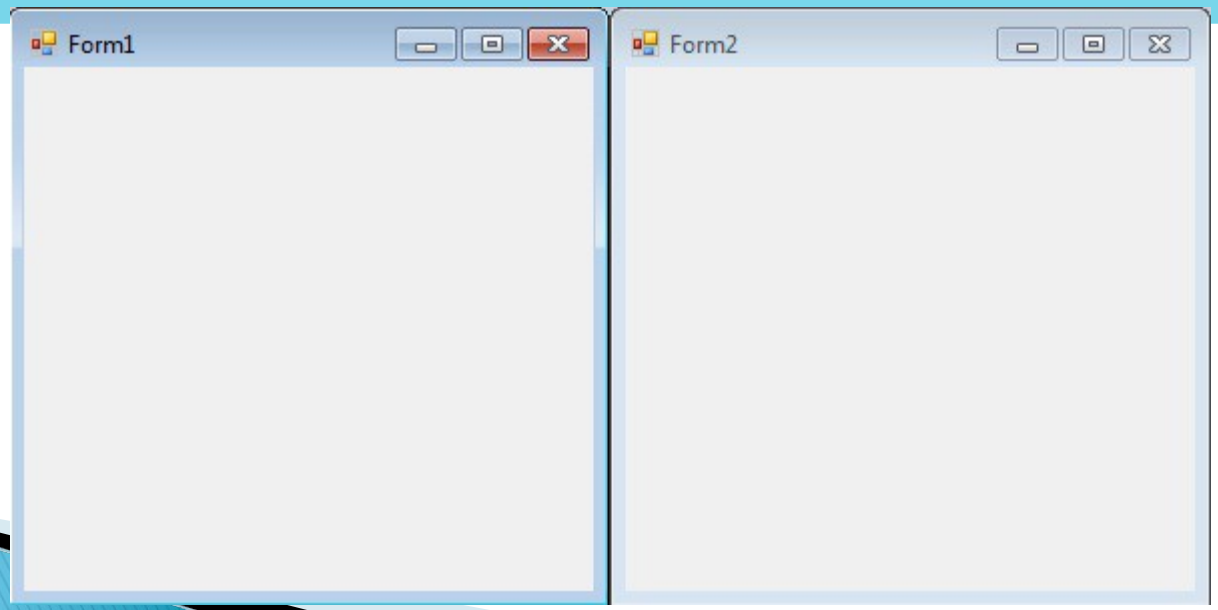
3.Nazwij ją *Form2.cs*. W ten sposób stworzyliśmy nową formę.

# Przykładowa aplikacja

Napiszmy teraz prostą aplikację złożoną z dwóch form (głównej i dodatkowej), które po uruchomieniu aplikacji pojawią się na ekranie obok siebie.

1. Stwórzmy nowy projekt aplikacji okienkowej.
2. Dodajmy nową formę. (według poprzedniego slajdu)
3. Klikamy na projekcie formy podstawowej (**Form1**) i w oknie **Properties** wybieramy zdarzenie **Load**.
4. Tworzymy metodę do zdarzenia i łączymy ją ze zdarzeniem podwójnie klikając na puste pole po prawej stronie nazwy zdarzenia. Zostanie utworzona metoda **Form1_Load**, która będzie wywoływana po wystąpieniu zdarzenia **Load**.
5. Wpisujemy kod, który utworzy nowe okno i ustawi jego początkowe położenie.

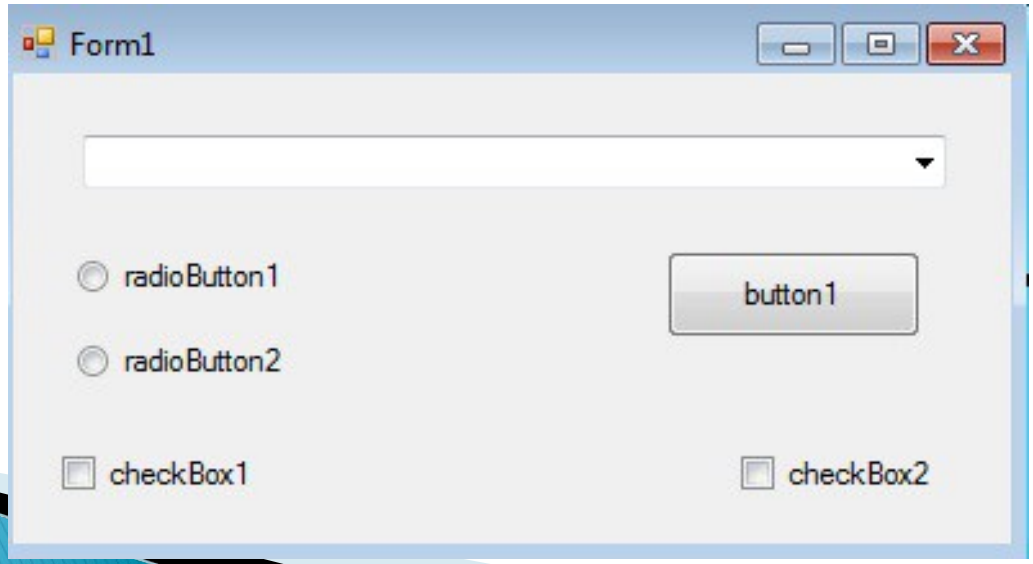
```
private void Form1_Load(object sender, EventArgs e)
{
    Form2 form2 = new Form2(); //tworzymy instancję
                               // nowej formy

    form2.Show(); //wyświetlamy formę
    form2.Location = new Point(this.Location.X + this.Size.Width,
                     this.Location.Y); //zmieniamy jej położenie
}
```



# Korzystanie z prostych kontrolek

Kontrolki są obiektami, które są umieszczone na formie. Przykładami kontrolek są: przyciski, pola tekstowe, etykiety, listy rozwijane itp.

Dodanie kontrolki do formy odbywa się poprzez wybór z okna Toolbox właściwej kontrolki i przeciągnięcie jej na formę. Rozmiar i położenie kontrolki można dopasować zarówno w czasie przeciągania, przed lub też po upuszczeniu jej na formę.



# Organizowanie kontrolek na formie

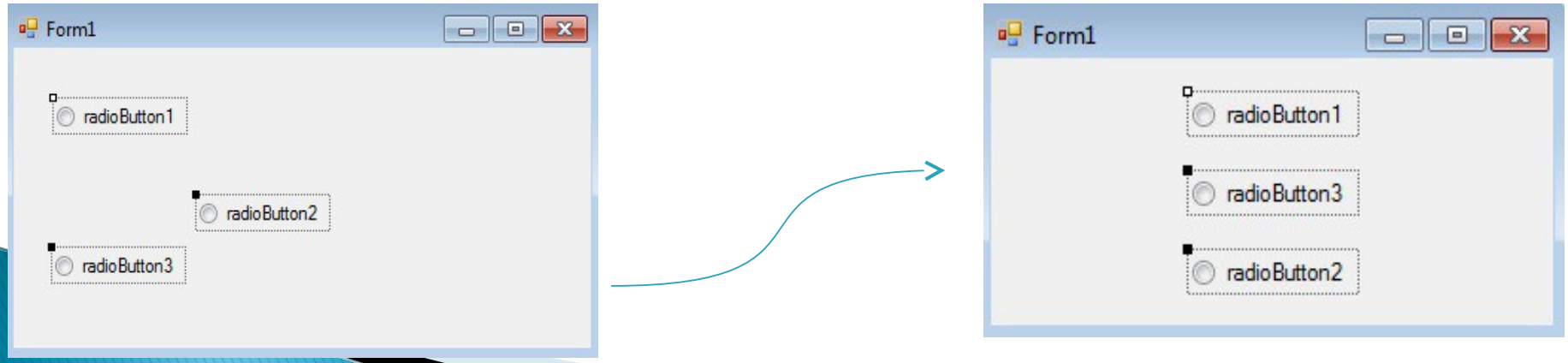
Po umieszczeniu kontrolek na formie można je dowolnie formatować.

W celu ich równomiernego rozmieszczenia oraz dopasowania wielkości, można posłużyć się jedną z wielu opcji formatowania, dostępnych w menu **Format.**

Należy zaznaczyć kontrolkę lub grupę kontrolek (grupę wybiera się trzymając wciśnięty klawisz **Ctrl** i wskazując kolejne kontrolki). Następnie wybieramy opcję formatowania.

Menu Format zawiera dużą ilość opcji pogrupowanych w podmenu, które służy do formatowania grupy kontrolek.

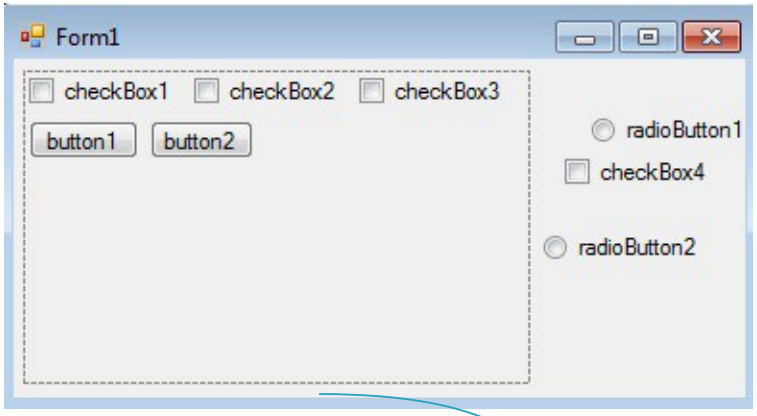
- **Align** – pozwala na wyrównanie kontrolek do jednej linii,
- **Make Same Size** – pozwala na wyrównanie rozmiarów kontrolek,
- **Horizontal Spacing** – pozwala na zmianę poziomego rozmiaru odstępu między kontrolkami,
- **Vertical Spacing** – pozwala na zmianę pionowego rozmiaru odstępu między kontrolkami,
- **Center in Form** – pozwala na umieszczenie kontrolek w centrum formy,
- **Order** – pozwala na zmianę porządku kontrolek,
- **Lock Controls** – pozwala na zablokowanie kontrolek do modyfikacji.



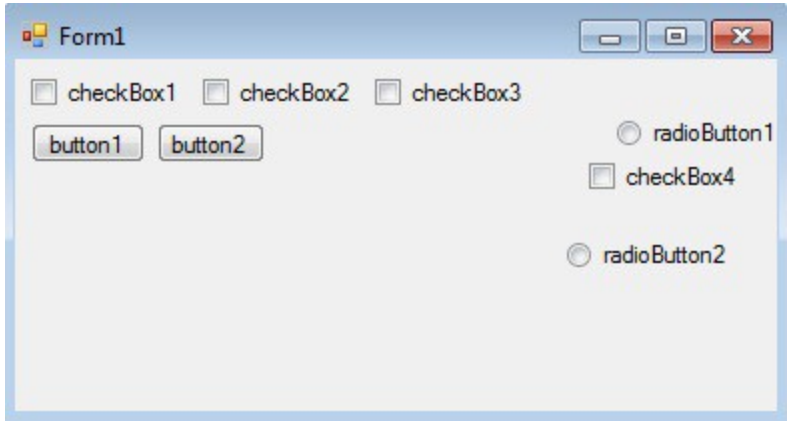
# Inny sposób na zachowanie porządku

FlowLayoutPanel

Kontrolki w obszarze FlowLayoutPanel są automatycznie ustawiane „jadna za drugą”, co nie pozwala nam na rozrzucenie kontrolek po formie.



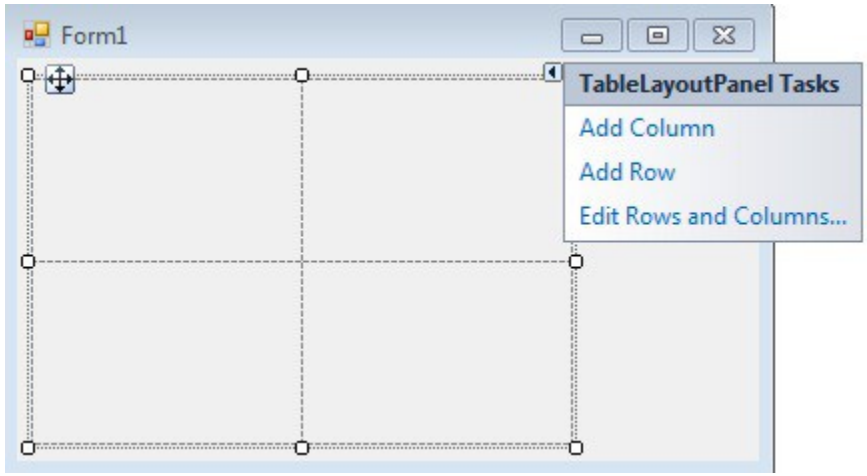

Po uruchomieniu, nie widzimy obramowania kontrolki FlowLayoutPanel.

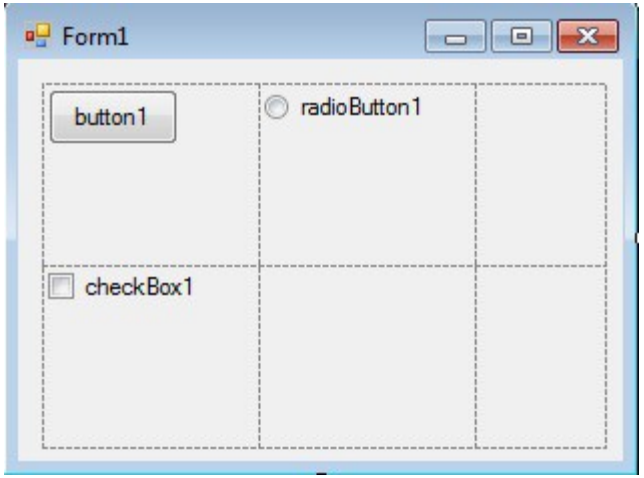


# Jeszcze inaczej...

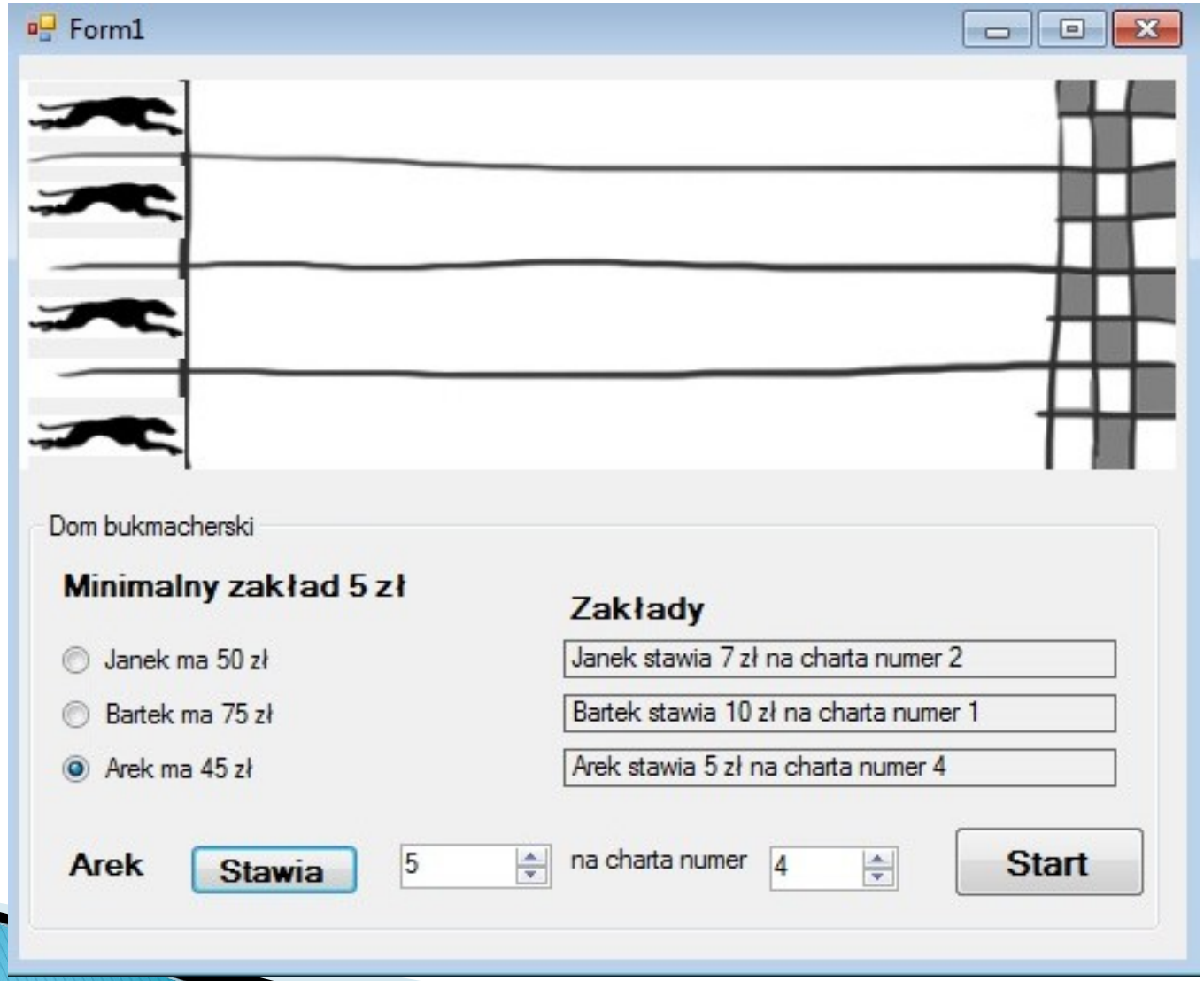
TableLayoutPanel

Umożliwia dodawanie kolumn i rzędów, w których kontrolki będą automatycznie wyrównywane do górnego lewego rogu. W danej komórce, może być tylko jedna kontrolka.



# Prezentacja programu



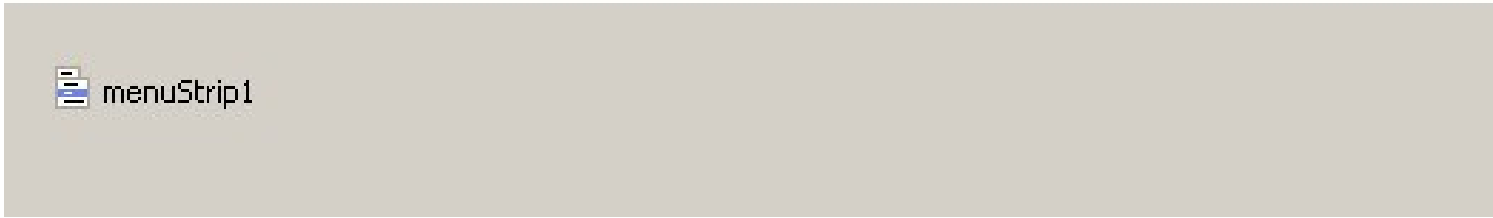
# Z innej beczki...

# Poznajmy bliżej kilka kontrolek

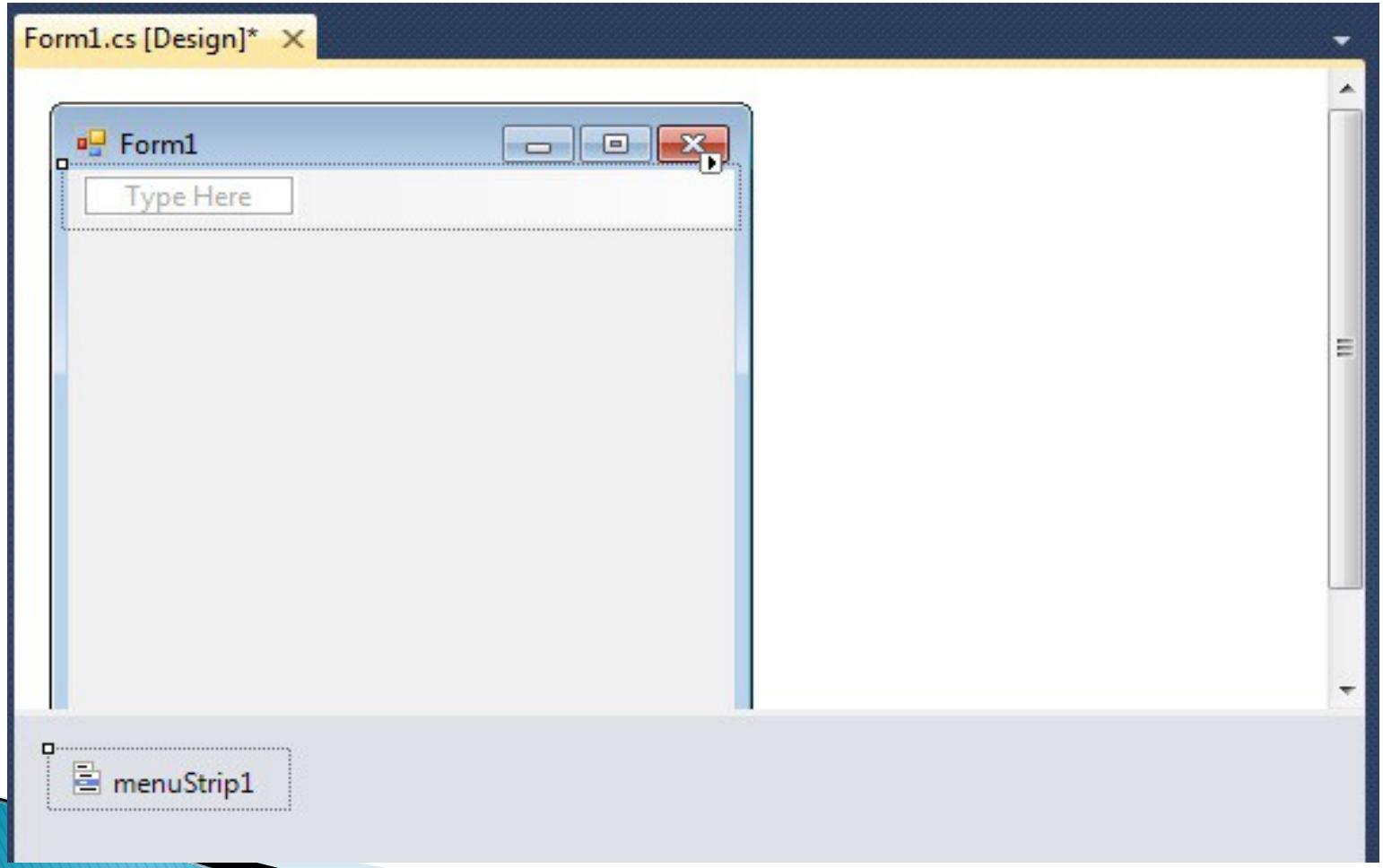
# Tworzenie menu

W wielu aplikacjach, nie tylko tych przeznaczonych dla systemu *Windows*, obecne jest menu główne. Projektowanie menu jest bardzo proste i przyjemne.

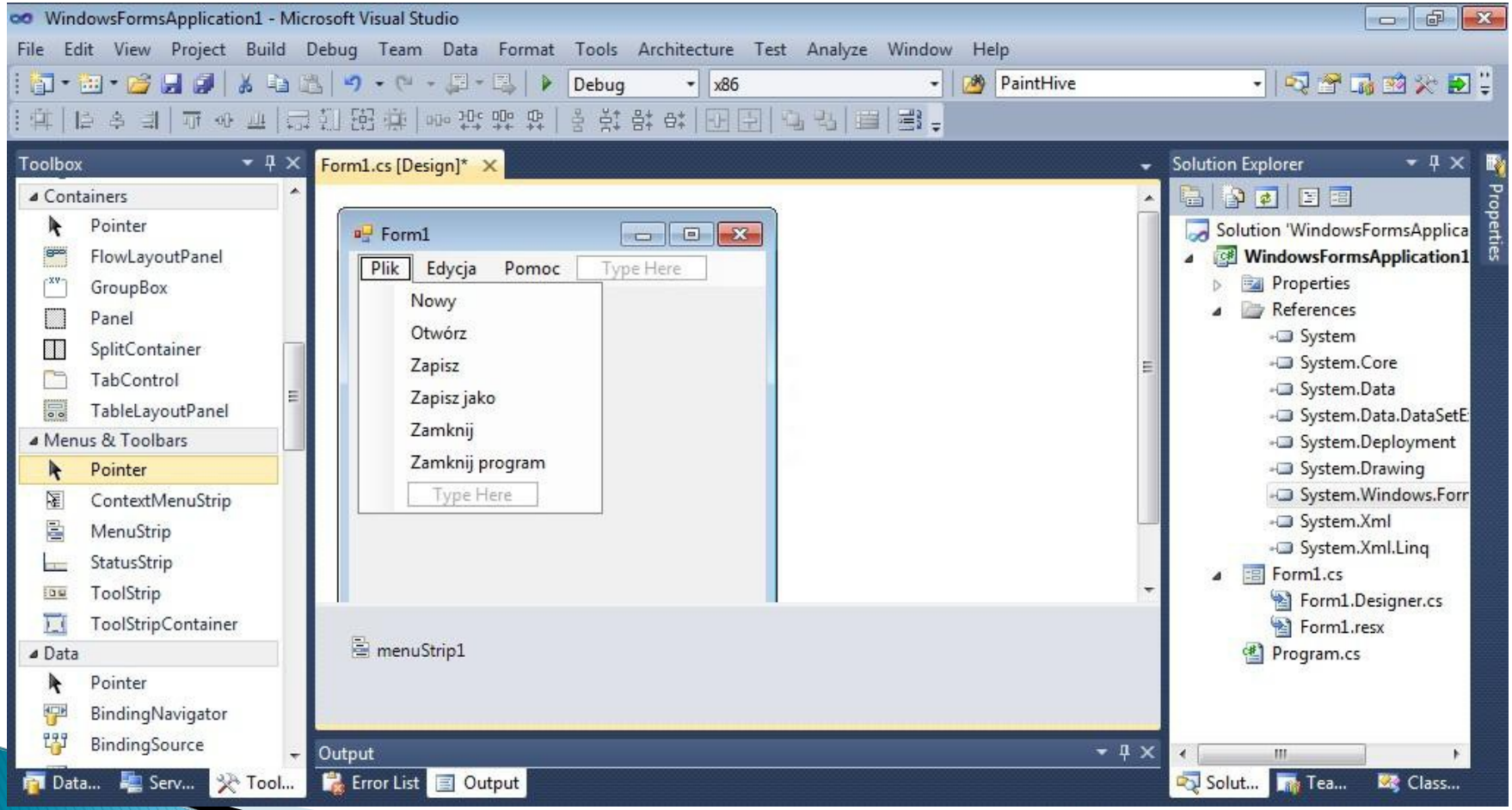
Za wyświetlanie i obsługę menu odpowiada komponent MenuStrip. Umieść ten komponent na formularzu. Zostanie on umieszczony na pasku podręcznym, gdyż należy do komponentów niewizualnych.



Po zaznaczeniu tego komponentu u góry formularza wyświetlona zostanie pozycja z napisem „Type here”, służąca do wpisania etykiety pierwszego menu



Po kliknięciu tej pozycji kursor zostanie zmieniony, co da nam możliwość wpisania etykiety. Po naciśnięciu klawisza *Enter* środowisko da nam możliwość dodania kolejnego elementu menu.



# Właściwości menu

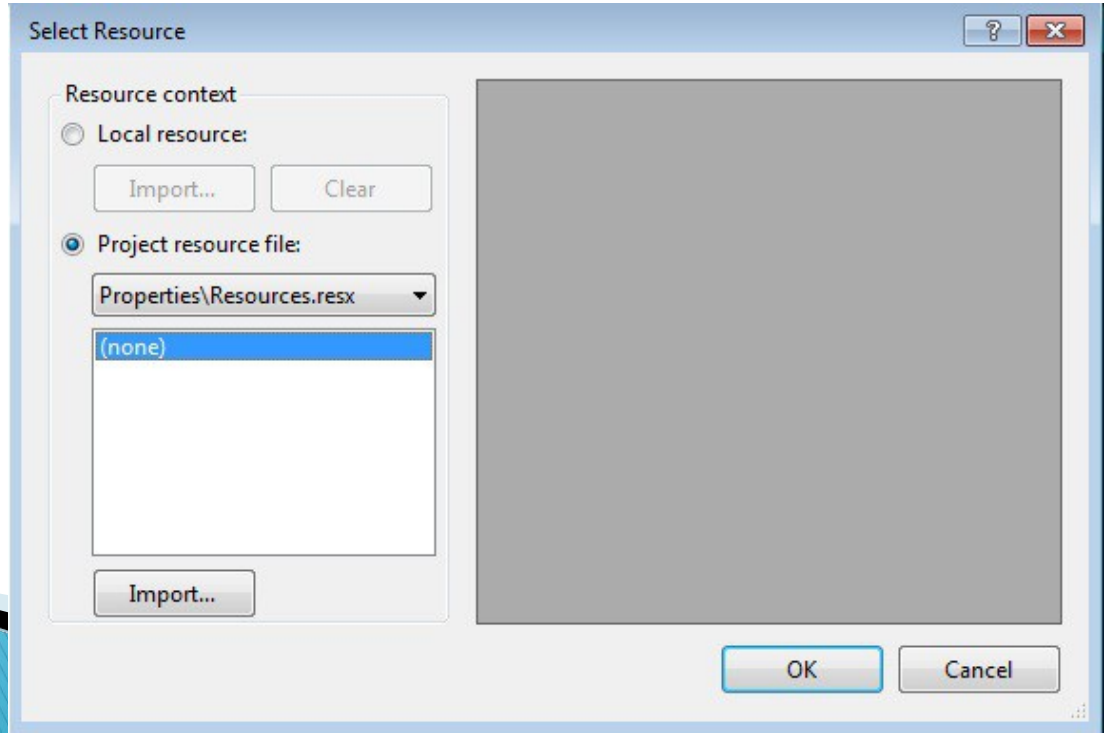
Menu można również edytować, jeśli klikniesz prawym przyciskiem ikonę komponentu i wybierzesz pozycję *Properties*. Mamy wówczas możliwość określenia szczegółowych właściwości dla menu. Przykładowo:

| BackColor | Umożliwia określenie koloru tła pozycji menu. |
|---|---|
| BackgroundImage | Umożliwia określenie obrazka tła. |
| RightToLeft | Określa, czy pozycje w menu będą wyrównane do prawej. |
| ShortcutKeys | Umożliwia określenie skrótu klawiaturowego dla elementu. |

# Ikony dla menu

Aby nasze menu było bardziej intuicyjne i ładne, dodamy do niego ikony, które będą zdobić jego pozycje. Obrazek może być zapisany w formacie *.bmp, *.gif, *.jpeg lub *.png.

Nowy obrazek możemy dodać bardzo prosto, korzystając z właściwości BackgroundImage. Po jej zaznaczeniu wyświetlona zostanie mała ikona — gdy ją naciśniemy otwarte zostanie okno *Select Resource* .

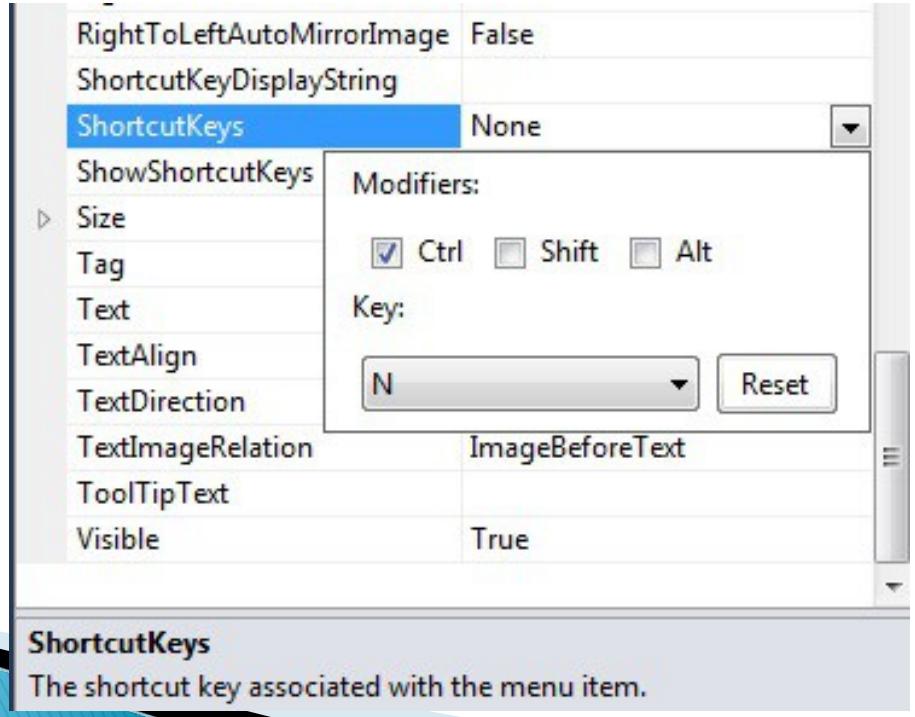


Zaznacz pozycję *Project resource file*, a następnie kliknij przycisk *Import*. Zostaniesz poproszony o wskazanie pliku z obrazem, który zostanie dodany do listy. Po naciśnięciu OK dany obrazek zostanie przypisany do pozycji menu.

Obrazki przypisane do menu będą zapisane w tzw. zasobach programu. Zasoby (takie jak np. obrazy, teksty, dźwięki) są dołączane do pliku wykonywalnego w trakcie kompilacji. Nie ma więc potrzeby dołączania do aplikacji dodatkowych plików graficznych czy dźwiękowych. Informacje o zasobach są przechowywane, w pliku Resources.resx, który znajduje się w podkatalogu Properties Twojego projektu.

# Skróty klawiaturowe

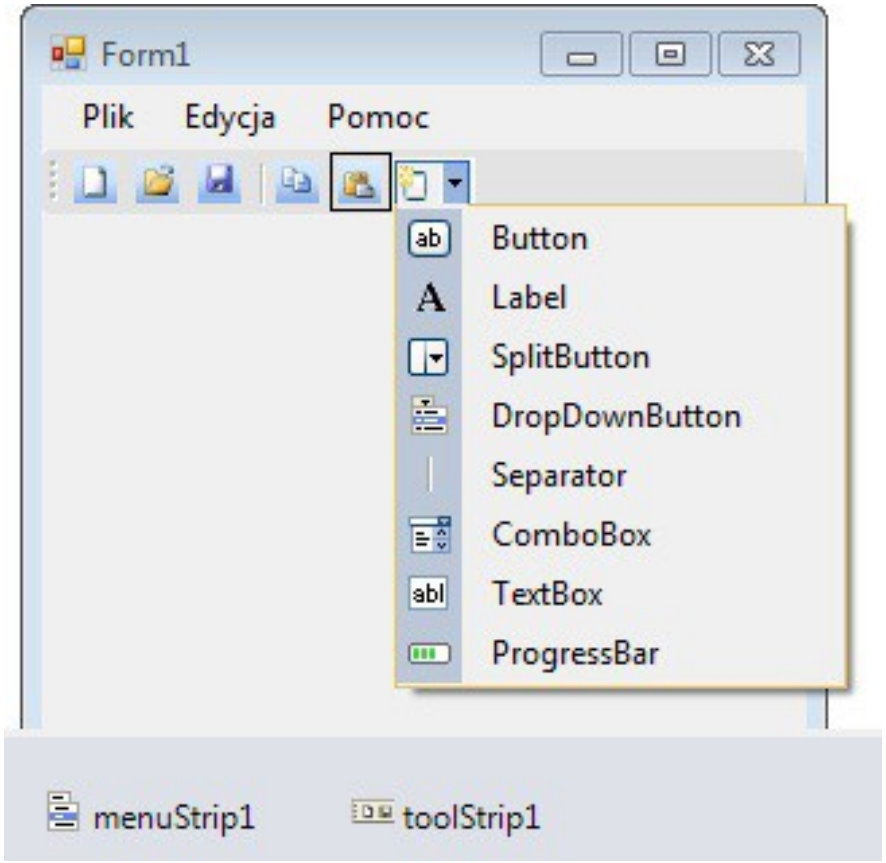
Aby nadać naszemu menu bardziej profesjonalny charakter, należy niektórym pozycjom przypisać skróty klawiaturowe. Umożliwi to szybkie wybranie danej pozycji przy pomocy klawiatury, co znacznie ułatwi pracę z aplikacją. Do ustawiania skrótu klawiaturowego służy właściwość *ShortcutKeys*. Dzięki niej możemy wybrać dowolną kombinację klawiaturową, która spowoduje wybranie danej pozycji.



# Paski narzędziowe

Paski narzędziowe (ang. *tool bar*) to również często spotykany element interfejsu aplikacji. Zawiera przyciski z najczęstszymi opcjami programu ozdobione ikoną. Podczas projektowania aplikacji w menu powinieneś umieszczać wszystkie opcje, jakie oferuje program, a na pasku jedynie te najczęściej używane.

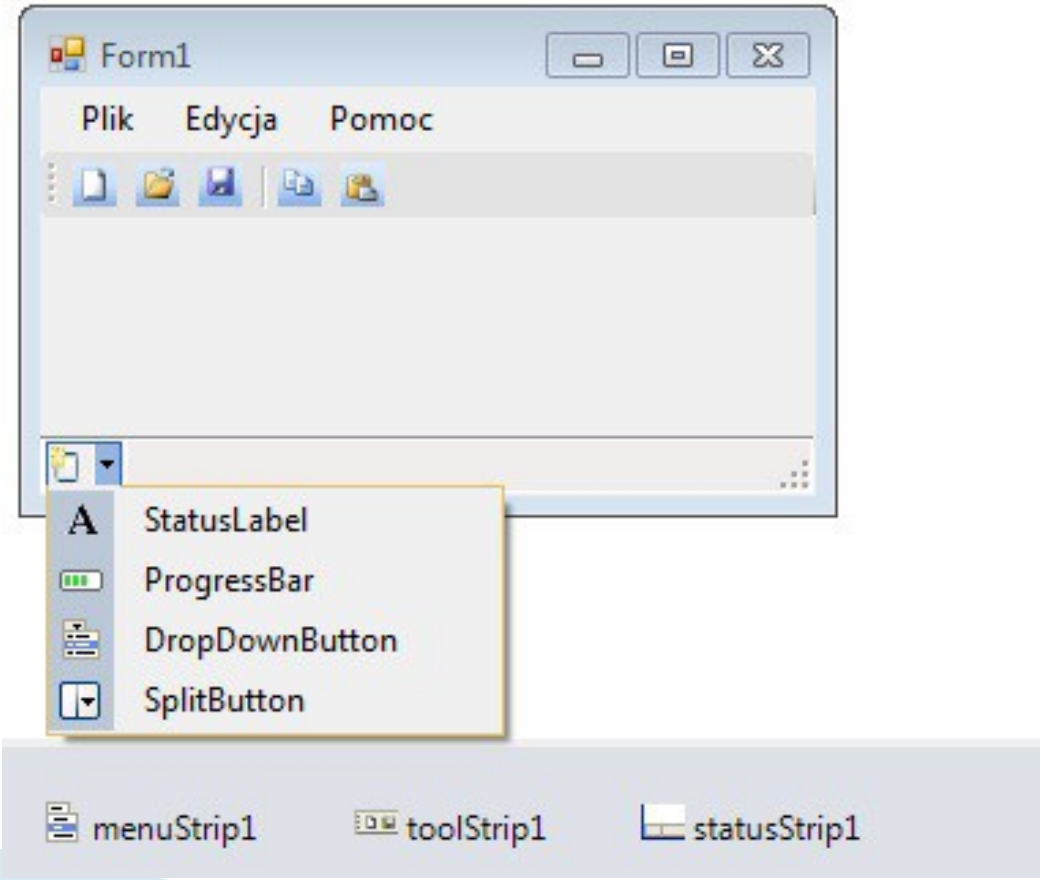
W bibliotece WinForms do tworzenia pasków narzędziowych możemy wykorzystać komponent ToolStrip. Umieszczony na formularzu, automatycznie dopasowuje się do jego górnej krawędzi.



# Pasek statusu

Pasek statusu również często gości we wszelkiego rodzaju aplikacjach okienkowych. Jest to pasek, który automatycznie dopasowuje się do dolnej krawędzi okna i służy do wyświetlania podstawowych informacji w trakcie działania programu.

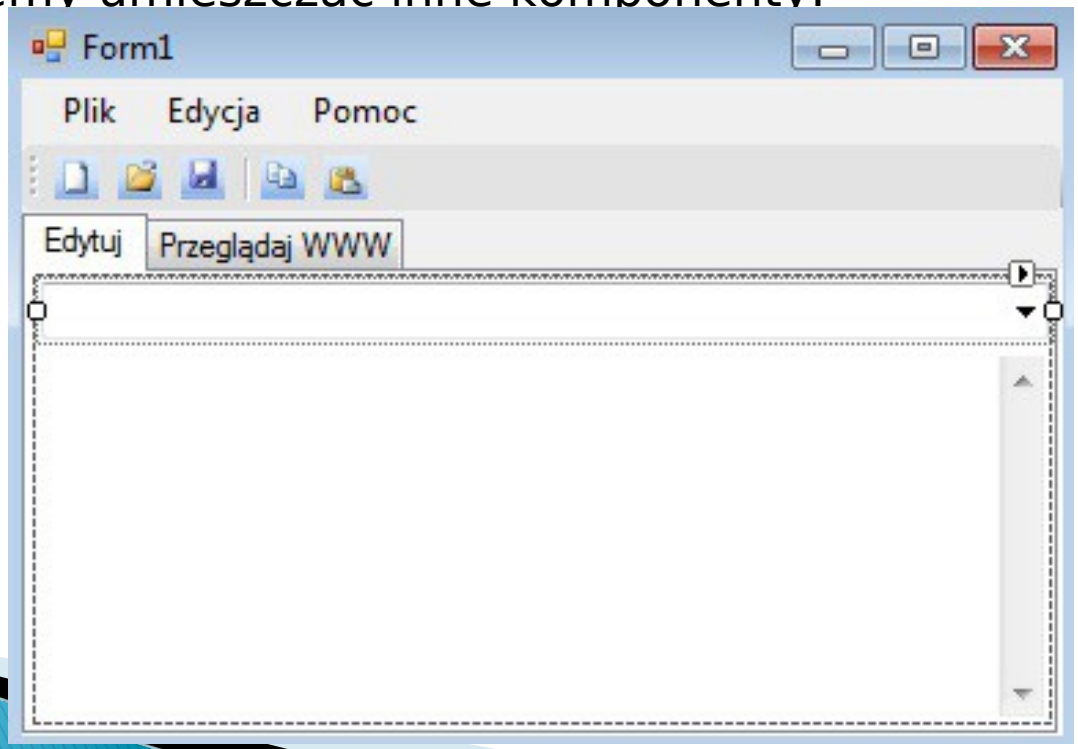
W *Windows Forms* za Wyświetlanie paska statusu odpowiada komponent StatusStrip.



# Zakładki

Często gdy mamy rozbudowany interfejs, zachodzi potrzeba podzielenia go na kilka części. Świetnie nadaje się do tego mechanizm zakładek, które możemy tworzyć dzięki kontrolce TabControl.

TabControl jest komponentem-rodzicem. Oznacza to, że w jego wnętrzu możemy umieszczać inne komponenty.



TabControl domyślnie posiada dwie zakładki, które możemy dodawać lub usuwać. Wystarczy je kliknąć i wybrać z menu podręcznego pozycję *Add* lub *Remove*. Jeżeli chcesz edytować właściwości danej zakładki, musisz wybrać właściwość *TabPages* z okna właściwości *Properties*. Kilka właściwości komponentu TabControl prezentuje tabela:

| **Właściwość** | **Opis** |
|---|---|
| *Alingment* | Określa położenie zakładek. Domyślnie są one położone u góry |
| *HotTrack* | Jeżeli właściwość ma wartość true, zakładka zostanie podświetlona, gdy naprowadzimy na nią kursor. |
| *Multiline* | Określa, czy tytuł zakładki może mieścić się w więcej niż jednej linii (domyślnie false — nie może). |

# Kontrolki tekstowe

Można podzielić kontrolki tekstowe na jednowierszowe i wielowierszowe. Wielowierszową kontrolką edycyjną jest komponent RichTextBox. Jednowierszową kontrolką edycyjną jest np. TextBox. Do często używanych kontrolek można zaliczyć również: ComboBox, ListBox.

Ta pierwsza prezentuje kolejne linie tekstu w formie listy rozwijanej. Komponent ListBox działa bardzo podobnie, tyle że kolejne linie prezentowane są w formie listy nierozwijanej. Na nowej zakładce komponentu TabControl mamy listę rozwijaną (ComboBox) oraz komponent WebBrowser.

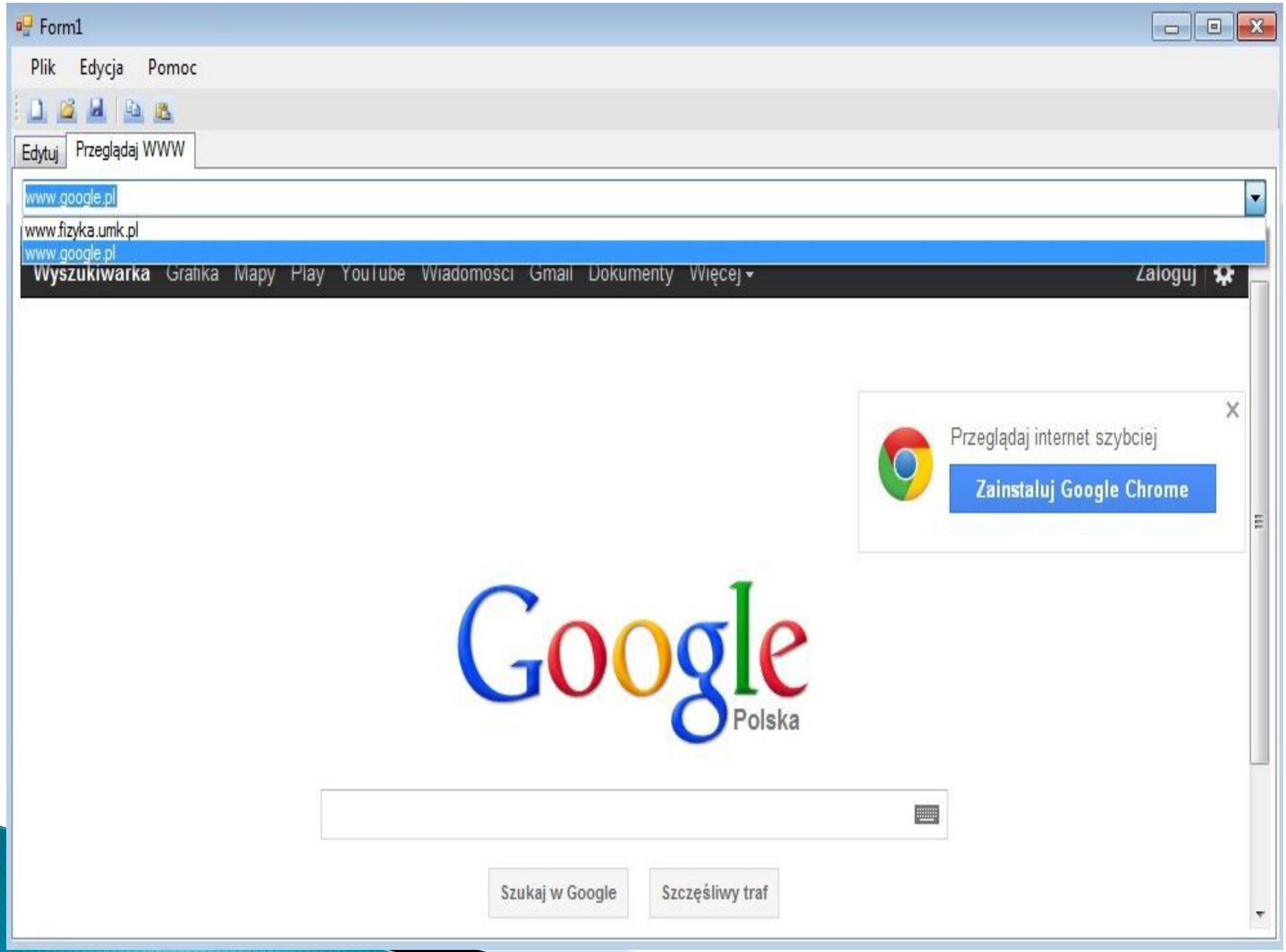
Komponent ComboBox będzie umożliwiał wpisanie adresu strony, która ma zostać wyświetlona w komponencie typu WebBrowser. Wpisane adresy stron WWW będą zapamiętywane i umieszczane na liście rozwijanej. Oprogramujmy więc zdarzenie KeyDown komponentu ComboBox. Musimy wychwycić moment naciśnięcia klawisza Enter. Jeżeli to nastąpi, ładujemy stronę o określonym URL, a sam adres dodajemy do listy rozwijanej

Procedura zdarzenia KeyDown wygląda następująco:

```
private void comboBox1_KeyDown(object sender, KeyEventArgs e)
 {
      if (e.KeyValue == (int)Keys.Enter)
      {
         webBrowser1.Navigate(comboBox1.Text);
         comboBox1.Items.Add(comboBox1.Text);
       }
}
```

Jeżeli warunek zostanie spełniony, wywołana będzie metoda Navigate(), która nakazuje przejść do określonego URL wpisanego w komponencie ComboBox. Kolejna instrukcja dodaje wpisany adres do listy. Musimy się w niej odwołać do właściwości Items, która reprezentuje kolekcję. Za dodanie nowej pozycji odpowiada metoda Add(). Kilka właściwości komponentu ComboBox zostało zaprezentowanych w tabeli.

| Właściwość | Opis |
|---|---|
| DropDownStyle | Reprezentuje styl listy rozwijanej. |
| DropDownHeight | Reprezentuje wysokość listy (w pikselach). |
| MaxDropDownItems | Oznacza maksymalną liczbę pozycji, jaka może znaleźć się na liście rozwijanej. |
| Sorted | Wartość true oznacza, że pozycje na liście będą sortowane. |

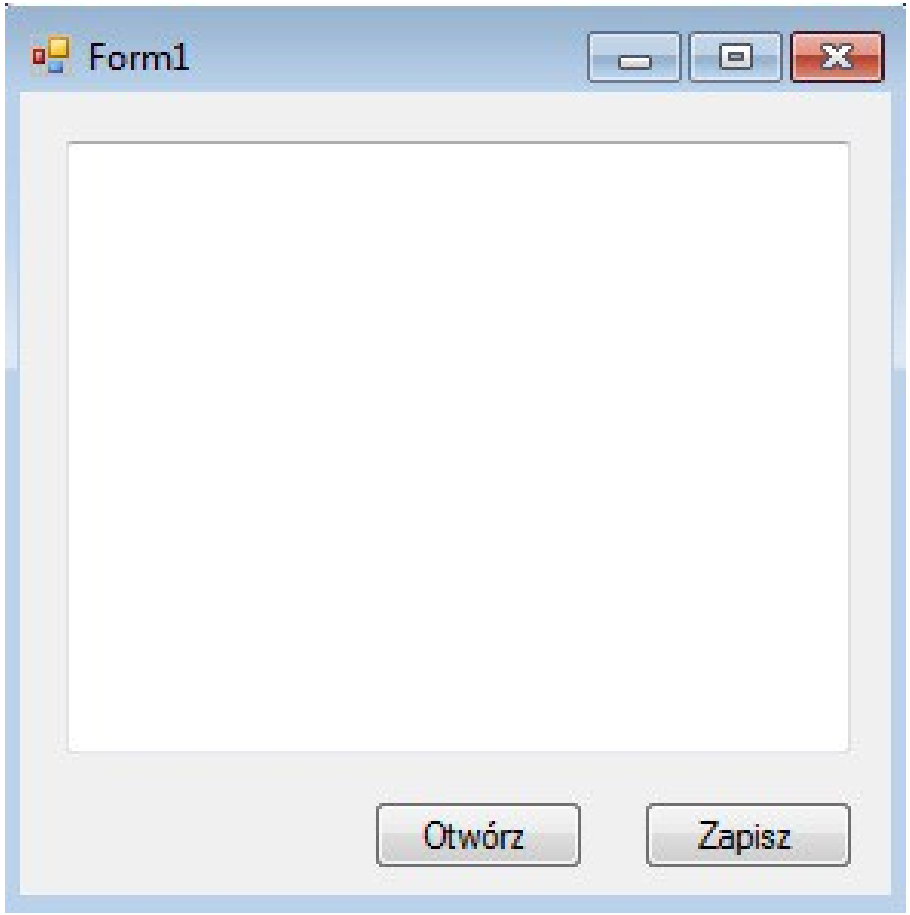

Form1
Plik
Edycja
Pomoc
Edytuj
Przeglądaj WWW
www.google.pl
www.fizyka.umk.pl
www.google.pl
Wyszukiwarka
Grafika
Mapy
Play
YouTube
Wiadomości
Gmail
Dokumenty
Więcej
Zaloguj
Przeglądaj internet szybciej
Zainstaluj Google Chrome
Google
Polska
Szukaj w Google
Szczęśliwy traf

# Okna dialogowe

Nasza przykładowa aplikacja ma pozycje w *menu* służące do otwierania, zapisywania pliku. Aby zrealizować to zadanie i nadać naszej aplikacji więcej profesjonalizmu, możemy skorzystać z okien dialogowych *Otwórz...* oraz *Zapisz...* Są to standardowe okna systemu *Windows*, które zapewne nieraz widziałeś. Umożliwiają one wskazanie pliku, który zostanie otwarty lub zapisany.

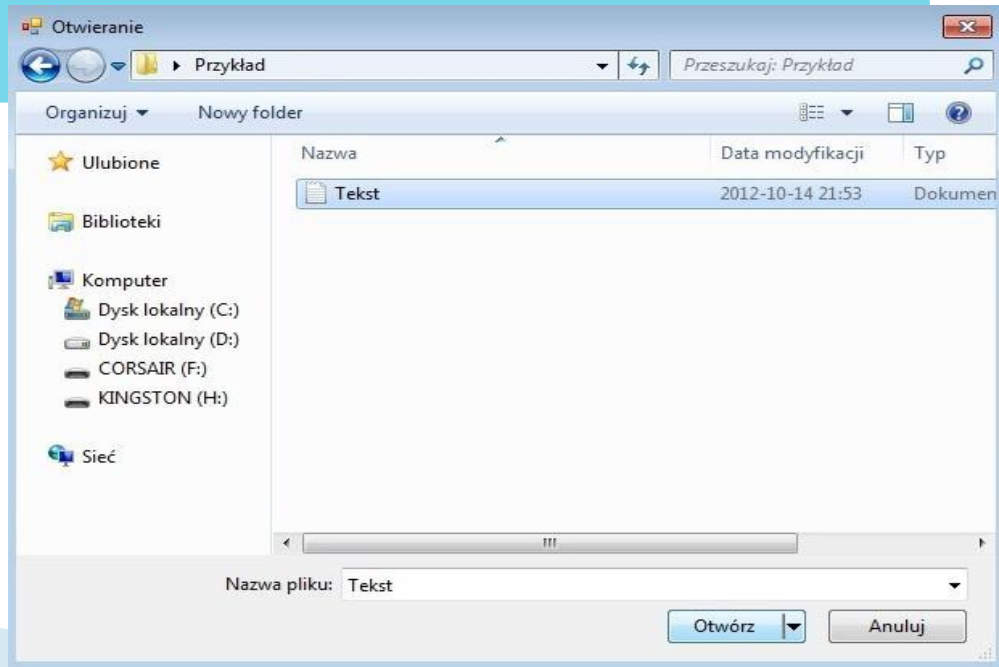
Komponenty umożliwiające wyświetlanie okien dialogowych znajdują się w kategorii *Dialogs* okna Toolbox. Utwórz na formularzu komponenty OpenFileDialog oraz SaveFileDialog. Są to komponenty niewizualne, więc reprezentujące je ikony zostaną umieszczone w panelu podręcznym zakładki projektowania.

Utwórz prosty formularz na wzór poniższego. Formularz ma składać się z dwóch przycisków oraz kontrolki TextBox. Właściwość tej kontrolki o nazwie Multiline musi być ustawiona na true.



Wygeneruj zdarzenie Click przycisku Otwórz. Kod metody zdarzeniowej może wyglądać tak:

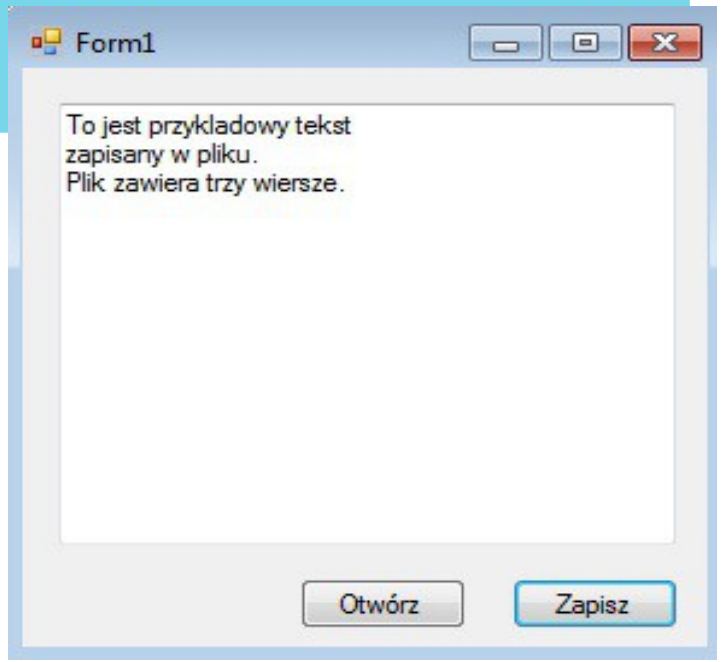
```
private string name;
private void button1_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        name = openFileDialog1.FileName;
        textBox1.Clear();
        textBox1.Text = File.ReadAllText(name);
    }
}
```



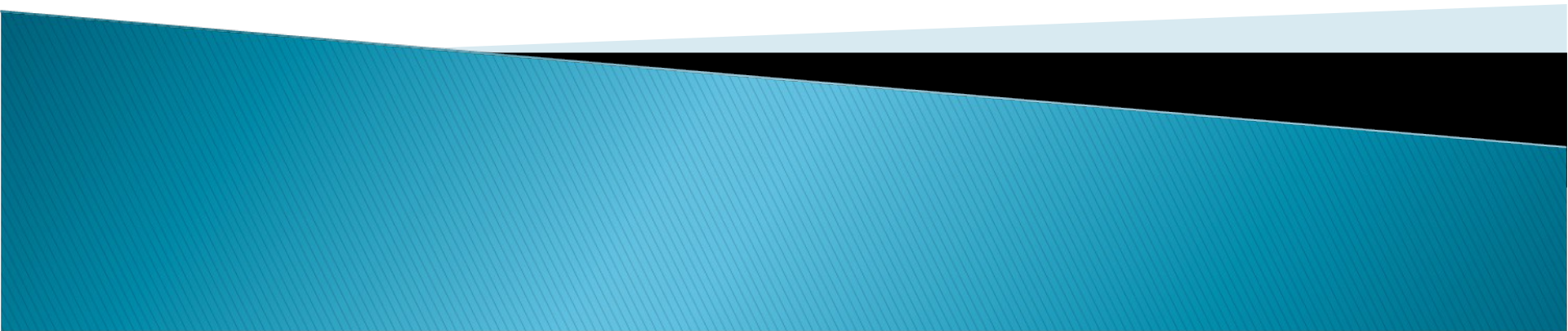

Na górze pliku trzeba dodać:
using System.IO;

Wygeneruj zdarzenie Click przycisku Zapisz. Kod metody zdarzeniowej może wyglądać tak:

```
private void button2_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        name = saveFileDialog1.FileName;
        File.WriteAllText(name+".txt", textBox1.Text);
     }
}
```



# Animowane kontrolki



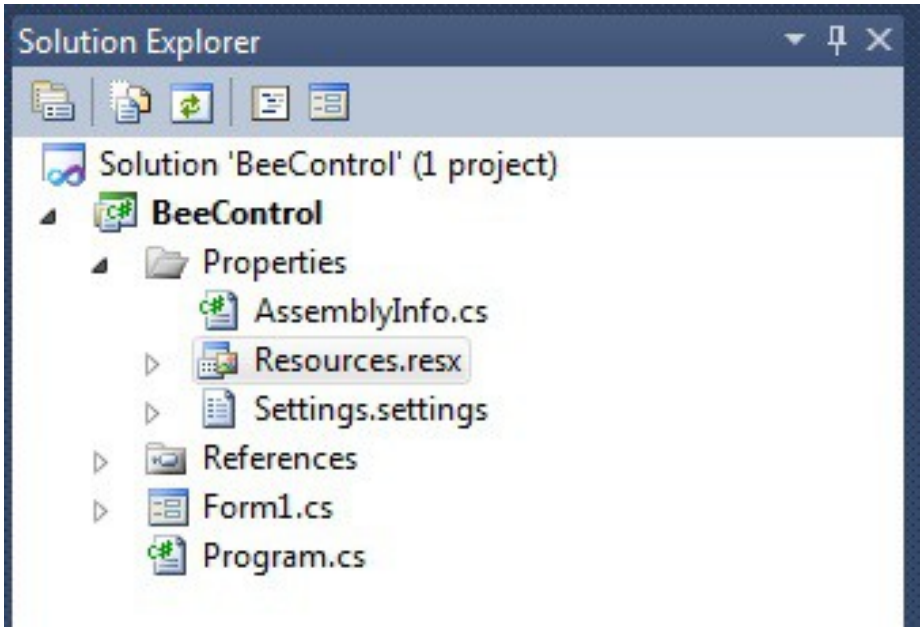
# Stwórz swoją pierwszą animowaną kontrolkę

Utworzymy własną kontrolkę, która będzie rysowała animowaną pszczołę. Na czym to polega??

Rysujesz sekwencję obrazków, jeden po drugim, stwarzając iluzję ruchu. Naszą kontrolkę nazwiemy BeeControl

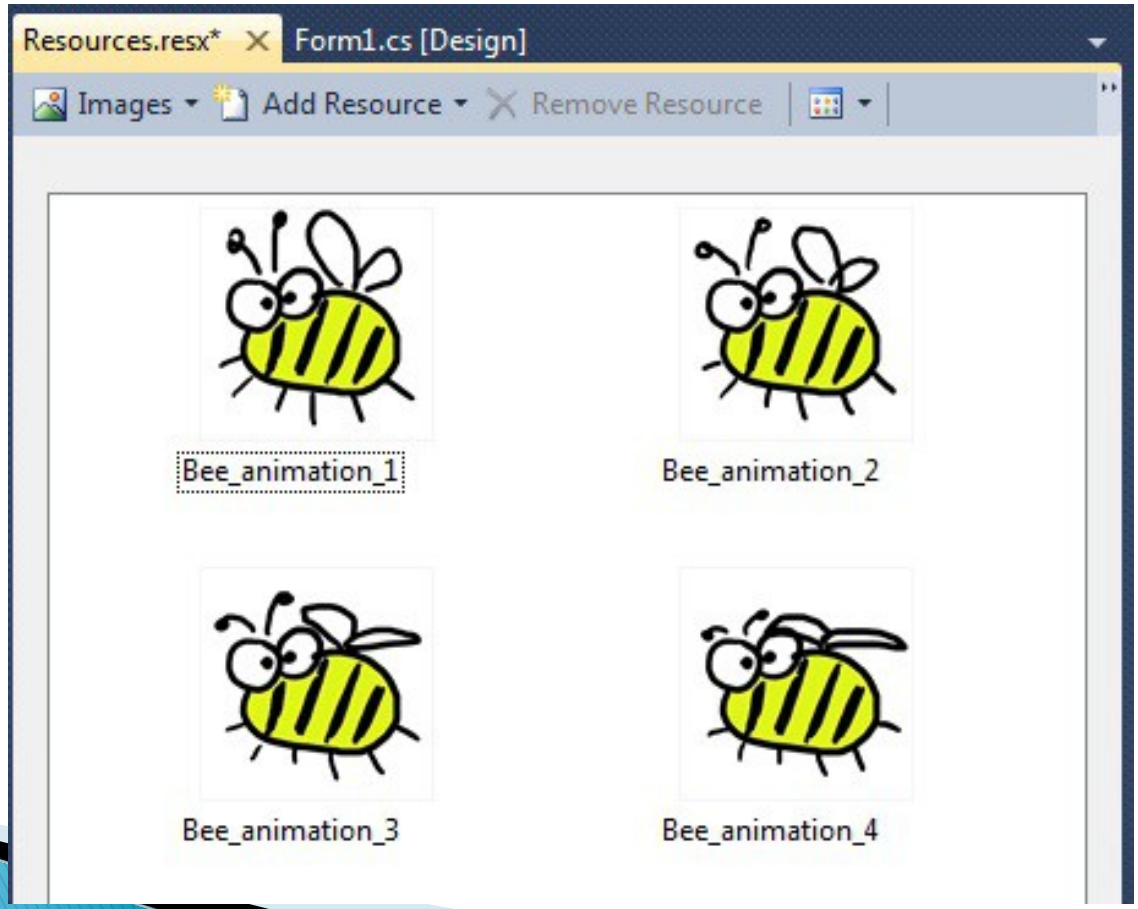
# 1. Utwórz nowy projekt…

I dodaj do niego cztery klatki animacji. Należy dodać je do zasobów projektu. W oknie Solution Explorer odszukaj zawierający je plik – Resources.resx (znajdziesz go w katalogu Properties). Kliknij go dwukrotnie, by wyświetlić okno z zasobami.



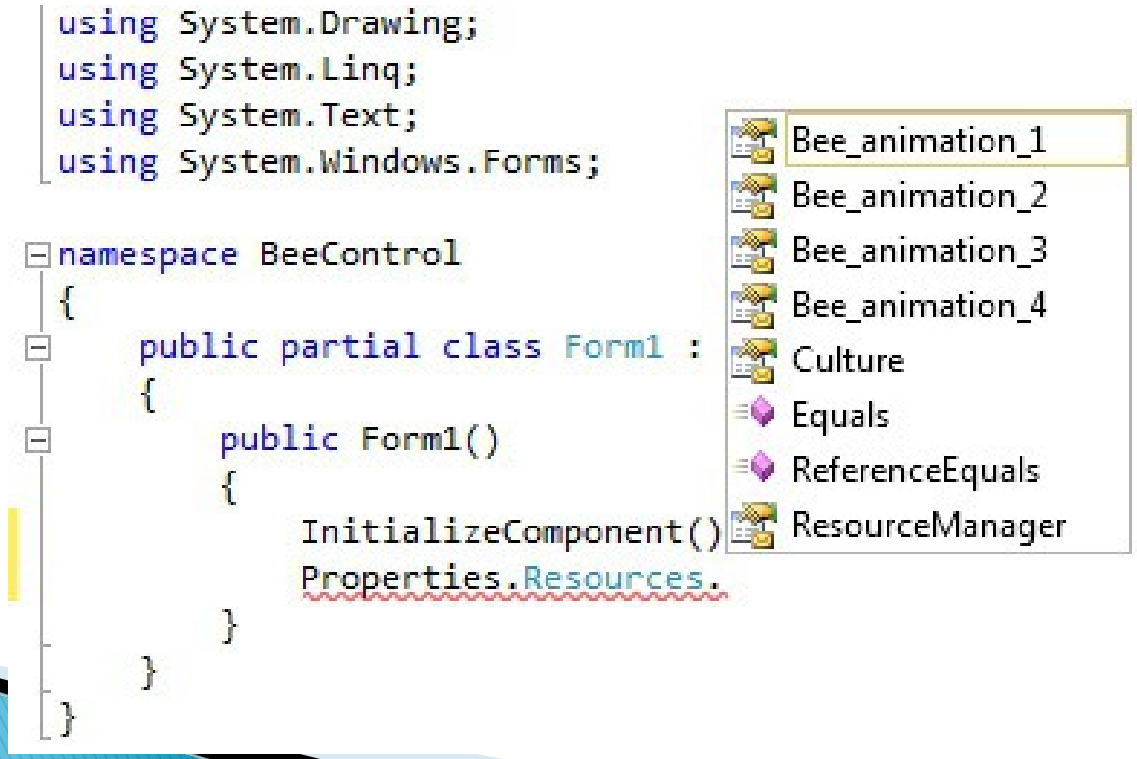
# 2. Dodajemy rysunki

Przejdź do zakładki Resources, wybierz Images z listy rozwijalnej znajdującej się w górnej części ekranu i kliknij Add Existing File na liście Add Resource.



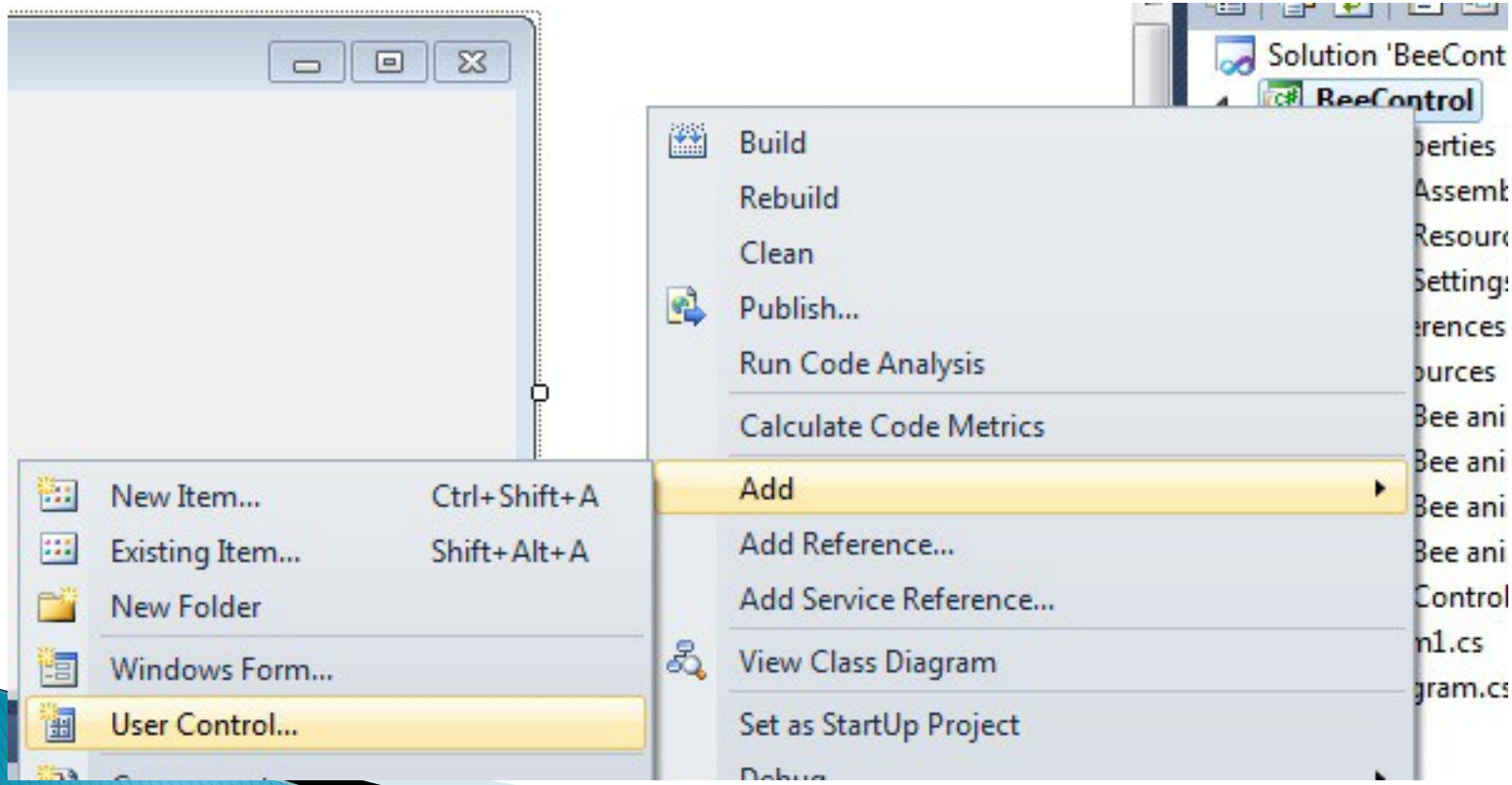
# 3. Properties.Resources

Kiedy dodajesz obrazki lub inne zasoby do pliku zasobów projektu, możesz uzyskać do nich dostęp poprzez klasę Properties.Resources. Przejdź po prostu do jakiegokolwiek wiersza w kodzie i wpisz Properties.Resources. – zaraz po tym zobaczysz rozwijaną listę, która pokaże wszystkie zaimportowane obrazki.



# 4. Dodaj teraz BeeControl

Kliknij projekt w oknie Solution Explorer i wybierz Add/User Control. Następnie trzeba dodać kontrolkę użytkownika o nazwie BeeControl. Środowisko graficzne otworzy ja w oknie projektanta formularzy.



# 5. Timer

Przeciągnij Timer na swoją kontrolkę użytkownika. Użyj okna Properties aby nadać zegarowi nazwę animationTimer i ustawić jego właściwość Interval na 150, a właściwość Enable na true. Kliknij ją dwukrotnie, aby dodać procedurę zdarzenia Tick. Dodaj kod:

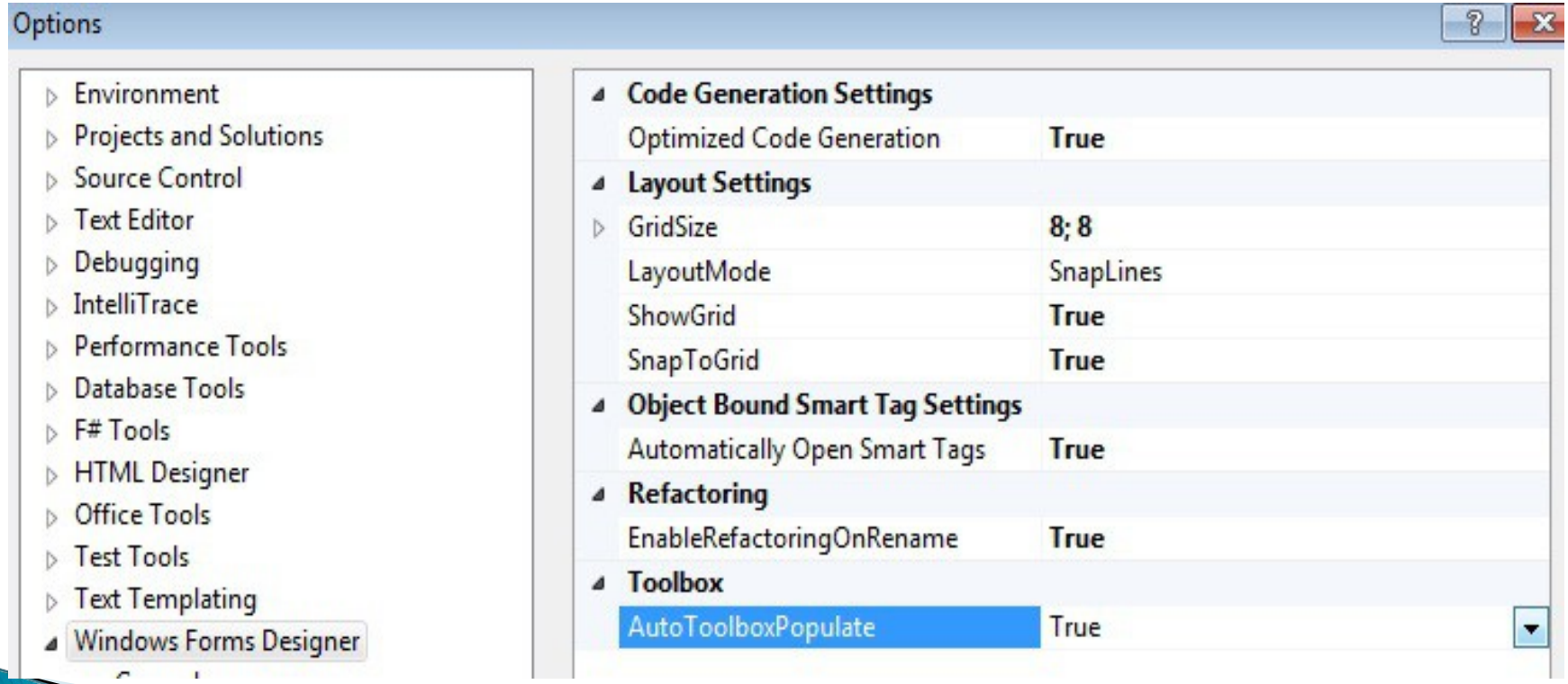
```
private int cell = 0;
void animationTimer_Tick(object sender, EventArgs e) {
    cell++;
    switch (cell){
        case 1: BackgroundImage = Properties.Resources.Bee_animation_1;
                break;
        case 2: BackgroundImage = Properties.Resources.Bee_animation_2;
                break;
        case 3: BackgroundImage = Properties.Resources.Bee_animation_3;
                break;
        case 4: BackgroundImage = Properties.Resources.Bee_animation_4;
                break;
       default: BackgroundImage = Properties.Resources.Bee_animation_2;
                cell = 0; break;
     }
}
```

# 6. Zmodyfikuj konstruktor

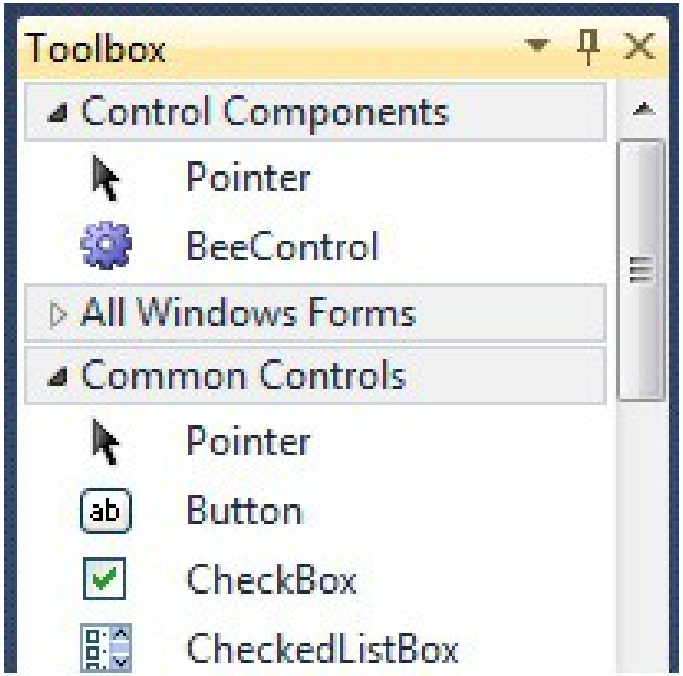
```
Public BeeControl()
{
        InitializeComponent();
        BackColor=Color.Transparent;
        BackgroundImageLayout = ImageLayout.Stretch;
}
```

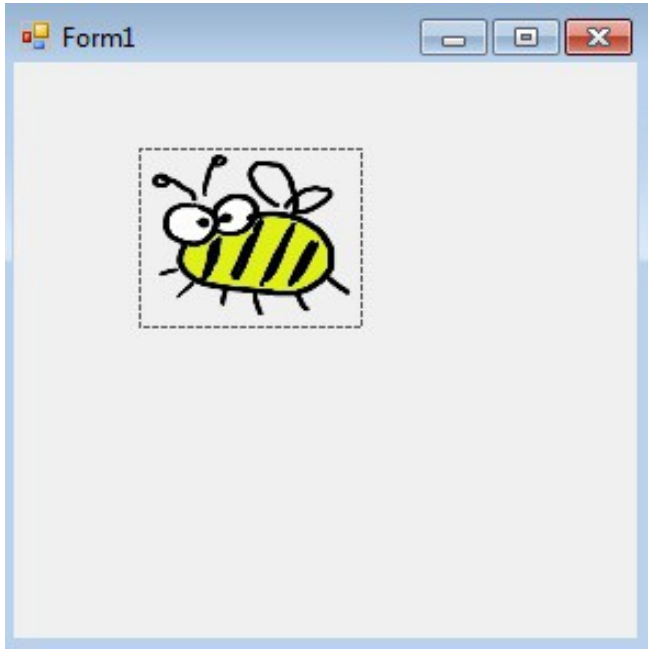
# 7. Ostatnie poprawki

Aby kontrolka była widoczna w ToolBox wybieramy:
Tools ->Options -> Windows Forms Dwsigner
Po czym wartość AutoToolboxPopulate ustawiamy na true.

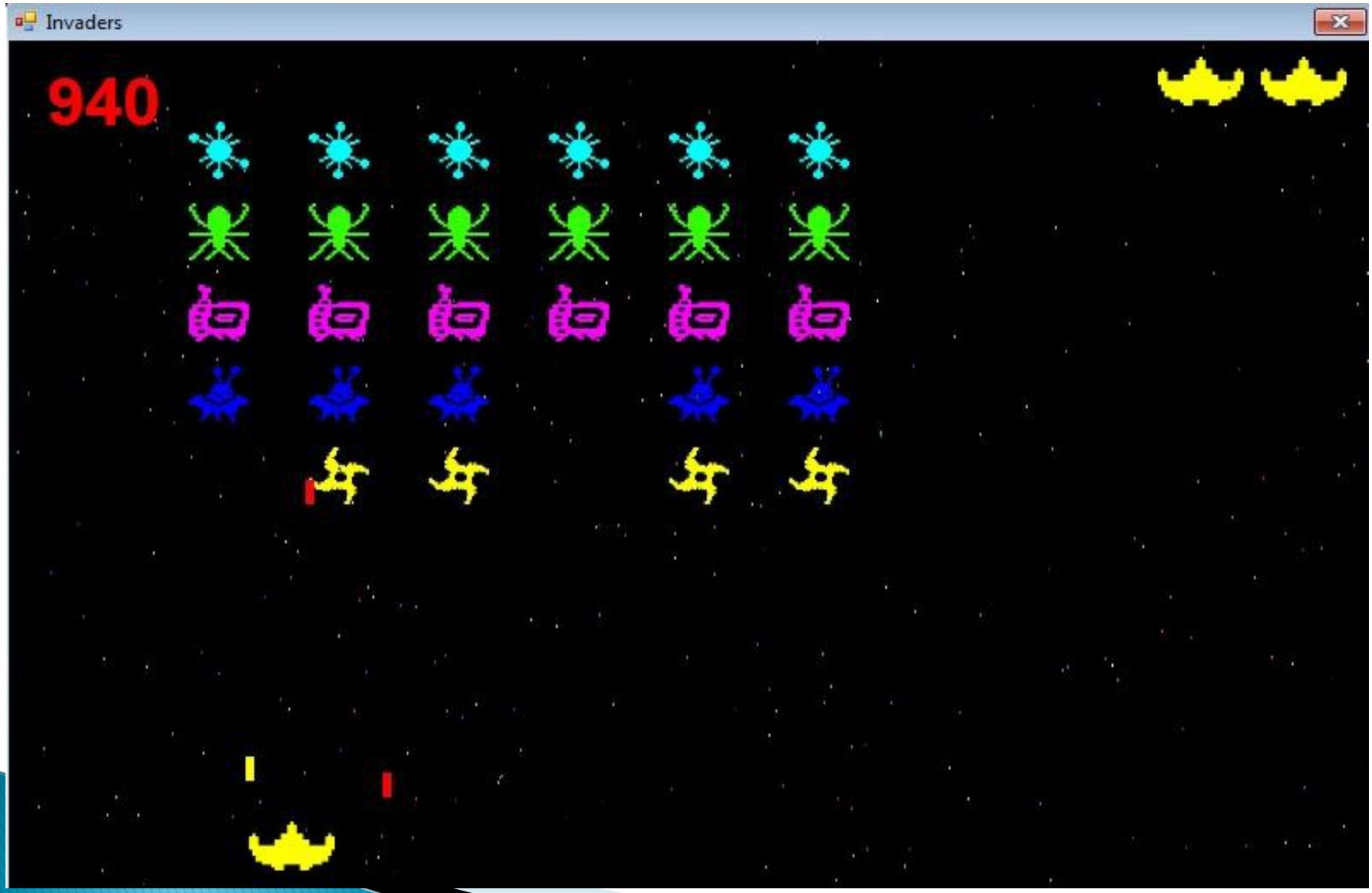


Program należy przebudować. Gdy zmieniasz kod kontrolki, to aby zobaczyć zmiany w projektancie formularzy, musisz przebudować program.

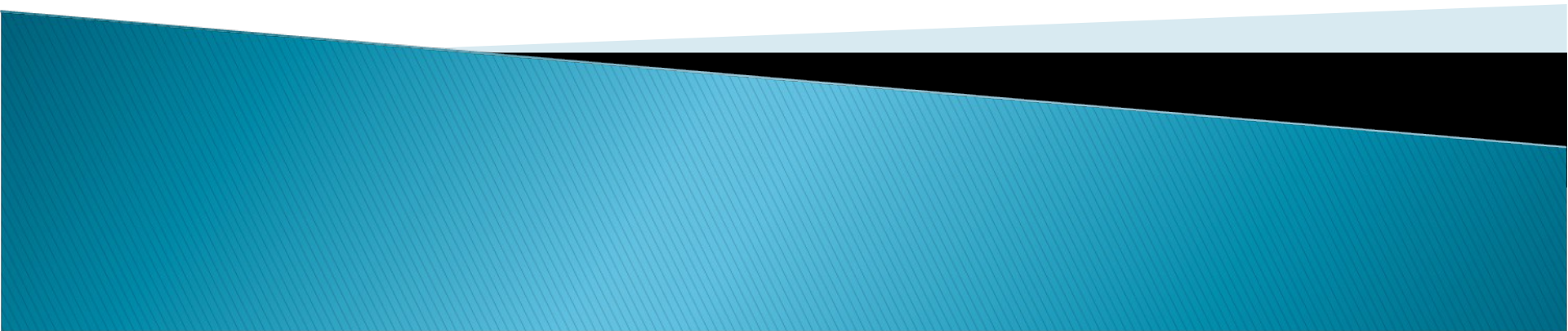


# A to był wstęp do...



# OpenGL



# Jak pracować z OpenGL?

**Potrzebne biblioteki można pobrać ze strony:**

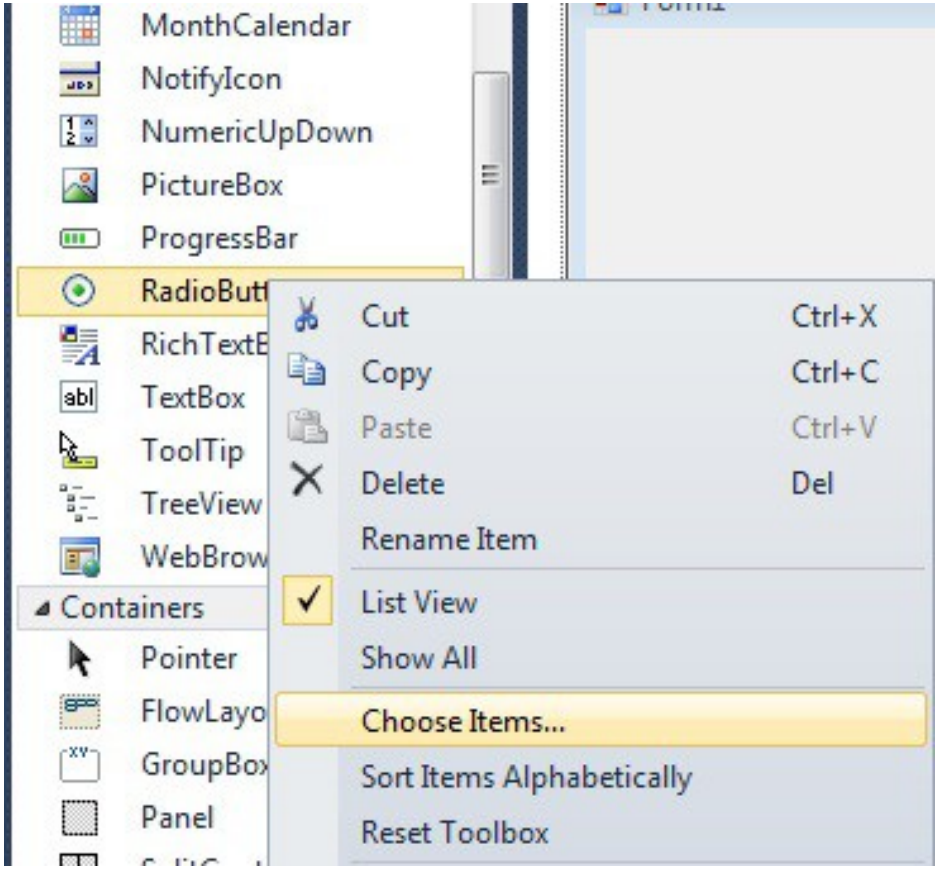
http://sourceforge.net/projects/csgl/files/

Po rozpakowaniu, należy przekopiować dwa pliki (csgl.dll, csgl.native.dll) do C:/Windows/System32

Napiszemy teraz pierwszą aplikacje korzystającą z OpenGL która będzie pewnego rodzaju szkieletem dla następnych przykładów. Utwórzmy nowy projekt.

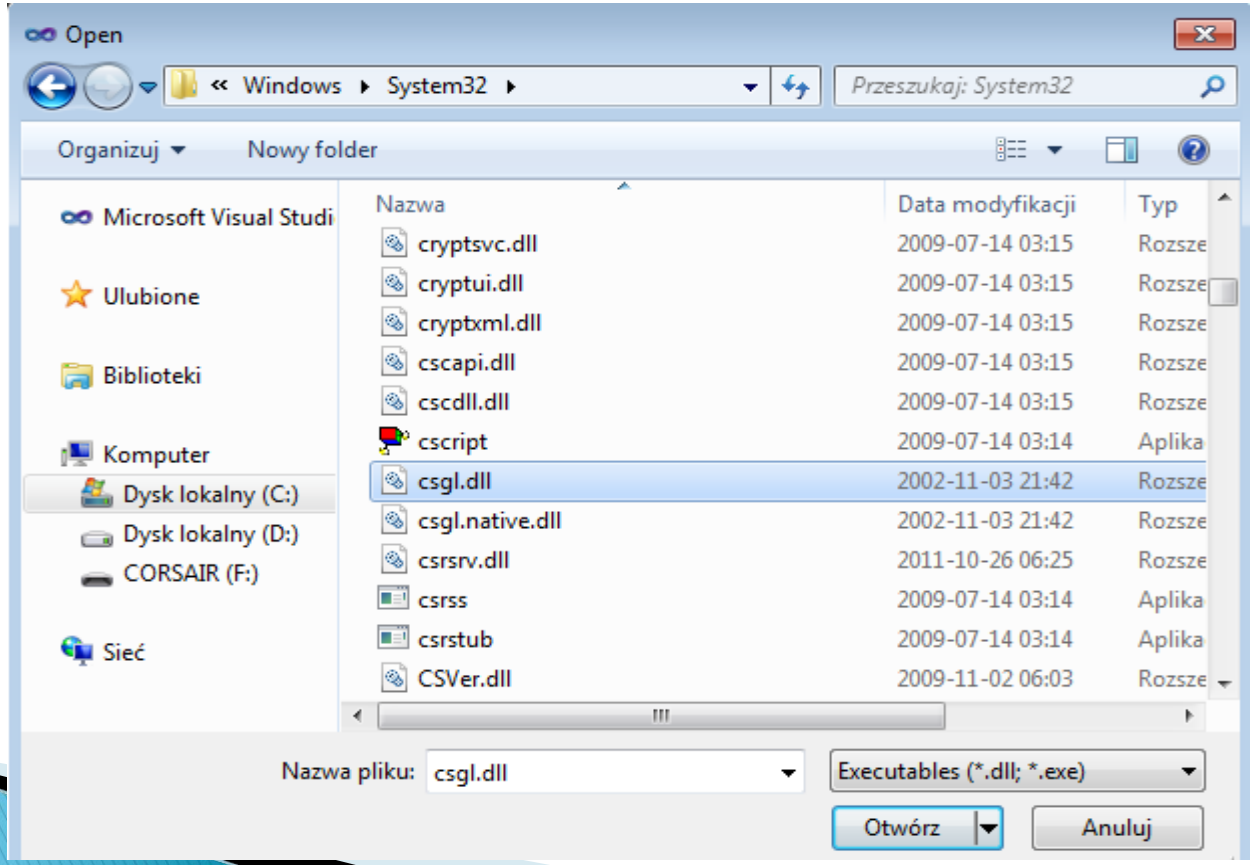
# Tworzymy ikonkę

Klikamy prawym przyciskiem myszy na dowolną ikonkę w oknie Toolbox, po czym wybieramy Choose Items...



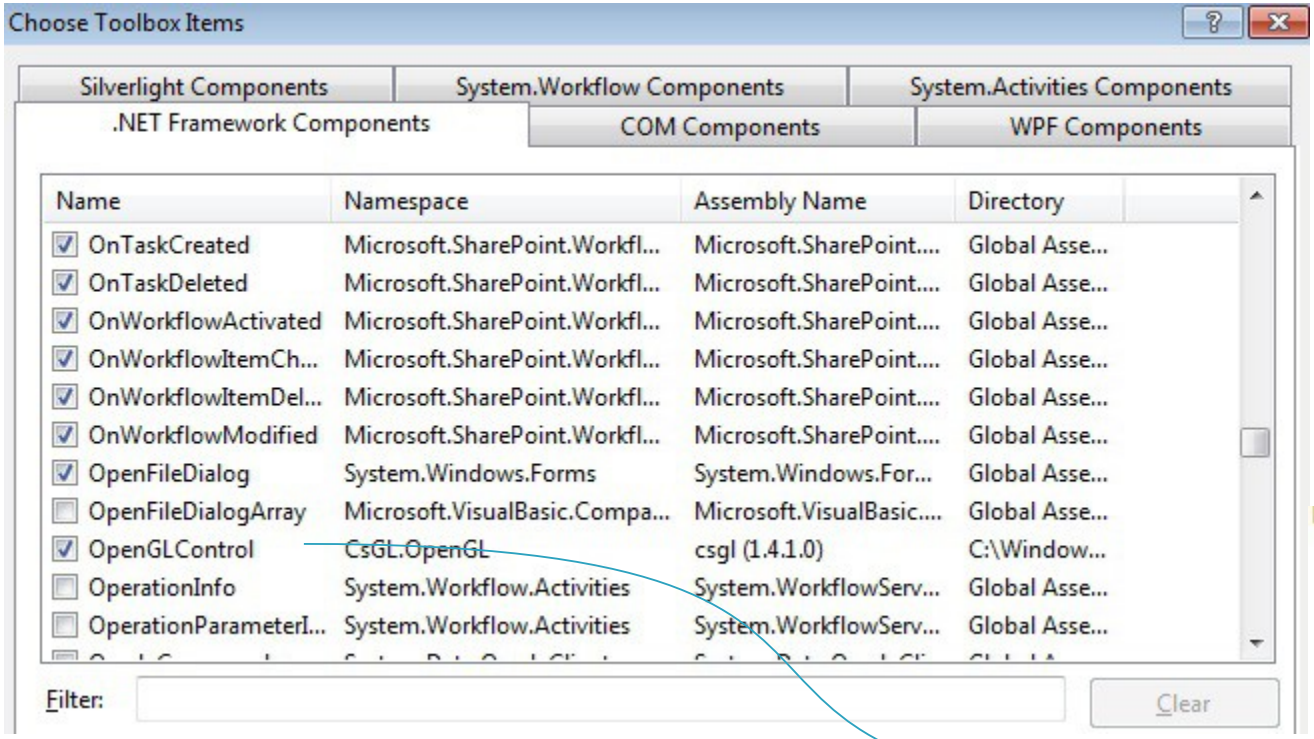
# Dodajemy bibliotekę

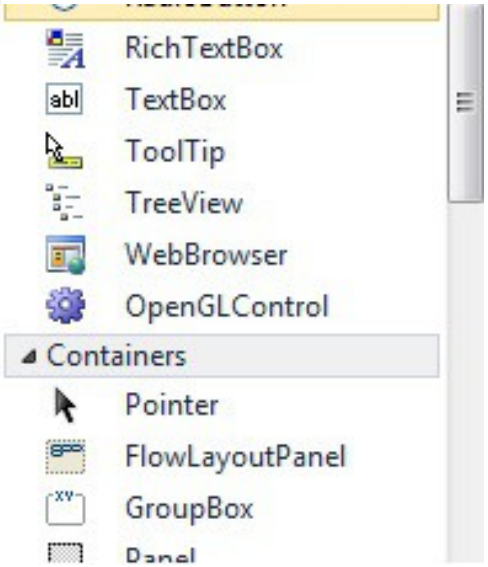
1. W zakładce .NET Framework Components wybieramy Browse…
2. Odnajdujemy bibliotekę csgl.dll.
3. Klikamy ok.



W zakładce .NET Framework Components wybieramy OpenGLControl.

W Toolbox powinna pojawić się nowa ikonka



Przeciągamy na nasz formularz kontrolkę OpenGLControl i w oknie Properties ustawiamy Dock na Fill. W ten sposób, kontrolka wypełni nam cały formularz.

Następnie dodajmy zegar i ustawmy jego właściwość Enabled na true. Kliknijmy na niego dwa razy, aby dodać kod obsługi zdarzenia.

Wróćmy do formularza, otwórzmy Preperties kontrolki OpenGL i kliknijmy dwa razy na zdarzenie Paint. Formularz sygnalizuje to zdarzenie za każdym razem, gdy musi przerysować swoją zawartość.

Na górze pliku Form1.cs dodajmy następującą linijkę:

**using CsGL.OpenGL;**

Następnie uzupełniamy zdarzenie timer1_Tick, dodając jedną linijkę:

**this.openGLControl1.Invalidate();**

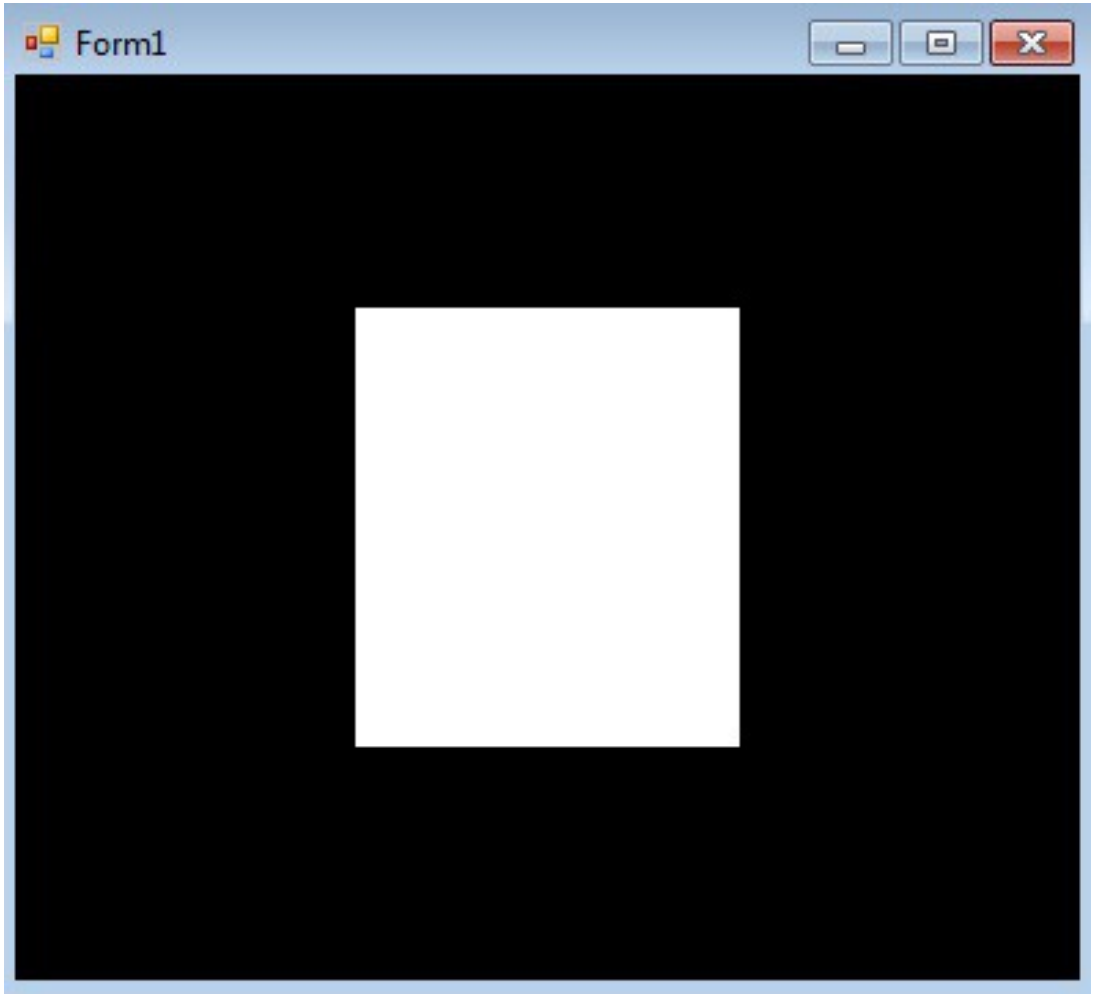
Do konstruktora dodajemy:

```
public Form1( )
{
    InitializeComponent();
    GL.glClearColor(0, 0, 0, 0);

    GL.glMatrixMode(GL.GL_PROJECTION);
    GL.glLoadIdentity();
    GLU.gluPerspective(45.0f, (double)800 /(double)600, 0.01f, 5000.0f);
}
```

Pozostaje tylko uzupełnić nasze zdarzenie, o polecenia rysujące kwadrat:

```
private void openGLControl1_Paint(object sender, PaintEventArgs e)
 {
    GL.glClear(GL.GL_COLOR_BUFFER_BIT |
GL.GL_DEPTH_BUFFER_BIT);

    GL.glMatrixMode(GL.GL_MODELVIEW);
    GL.glLoadIdentity();

     GL.glTranslated(0, 0, -5);
     GL.glBegin(GL.GL_QUADS);
     GL.glVertex3d(-1, 1, 0);
     GL.glVertex3d(-1, -1, 0);
     GL.glVertex3d(1, -1, 0);
     GL.glVertex3d(1, 1, 0);
     GL.glEnd();
}
```

Po kompilacji powinniśmy zobaczyć coś takiego



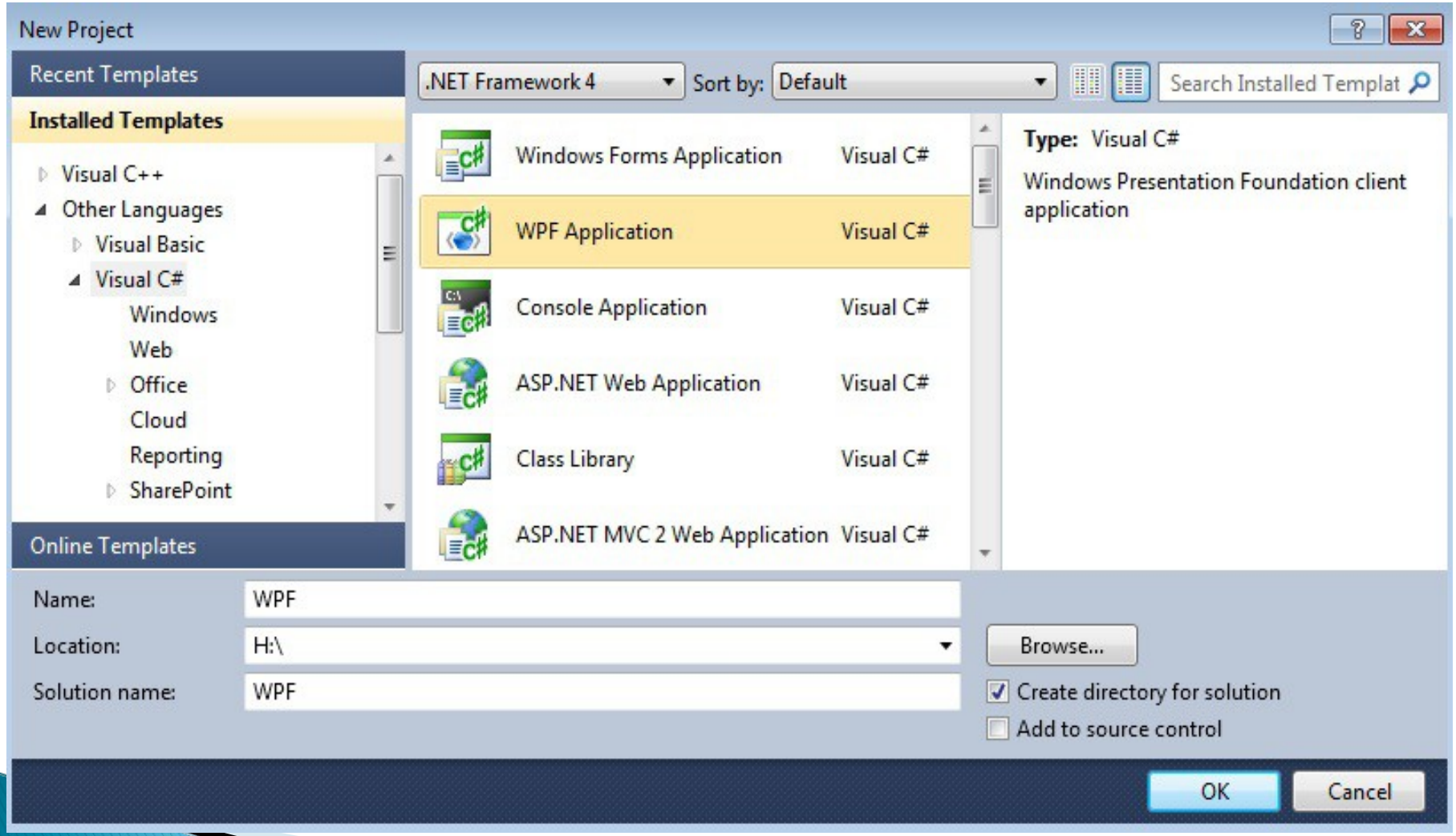
# WPF

**Windows Presentation Foundation** (WPF), to platforma służąca do budowania aplikacji wizualnych. Opiera się na XML, skalowalnych kontrolkach, całkowicie nowych kontrolkach systemowych, animacji 2D i 3D, rozkładzie tekstu i formatowaniu dokumentów.

WPF jest stosunkowo nową technologią. Zapewnia wszelakie wymagane funkcjonalności do stworzenia bardzo atrakcyjnych interfejsów.

Największą zaletą aplikacji używających WPF jest fakt, iż są oparte o grafikę wektorową. Oznacza to, że zawartość okna możemy dowolnie skalować, np. na potrzeby osób z wadą wzroku można ją dwukrotnie, trzykrotnie powiększyć bez straty jakości.

# Stworzenie aplikacji graficznej WPF



Visual Studio teraz wyświetla puste okno aplikacji WPF. Jak widzisz za pomocą paska możesz wielkość aplikacji powiększać i pomniejszać w tym oknie.

Na dole widzisz okno z językiem XAML , który opisuje wygląd aplikacji. Nie będziemy się nim zajmować.

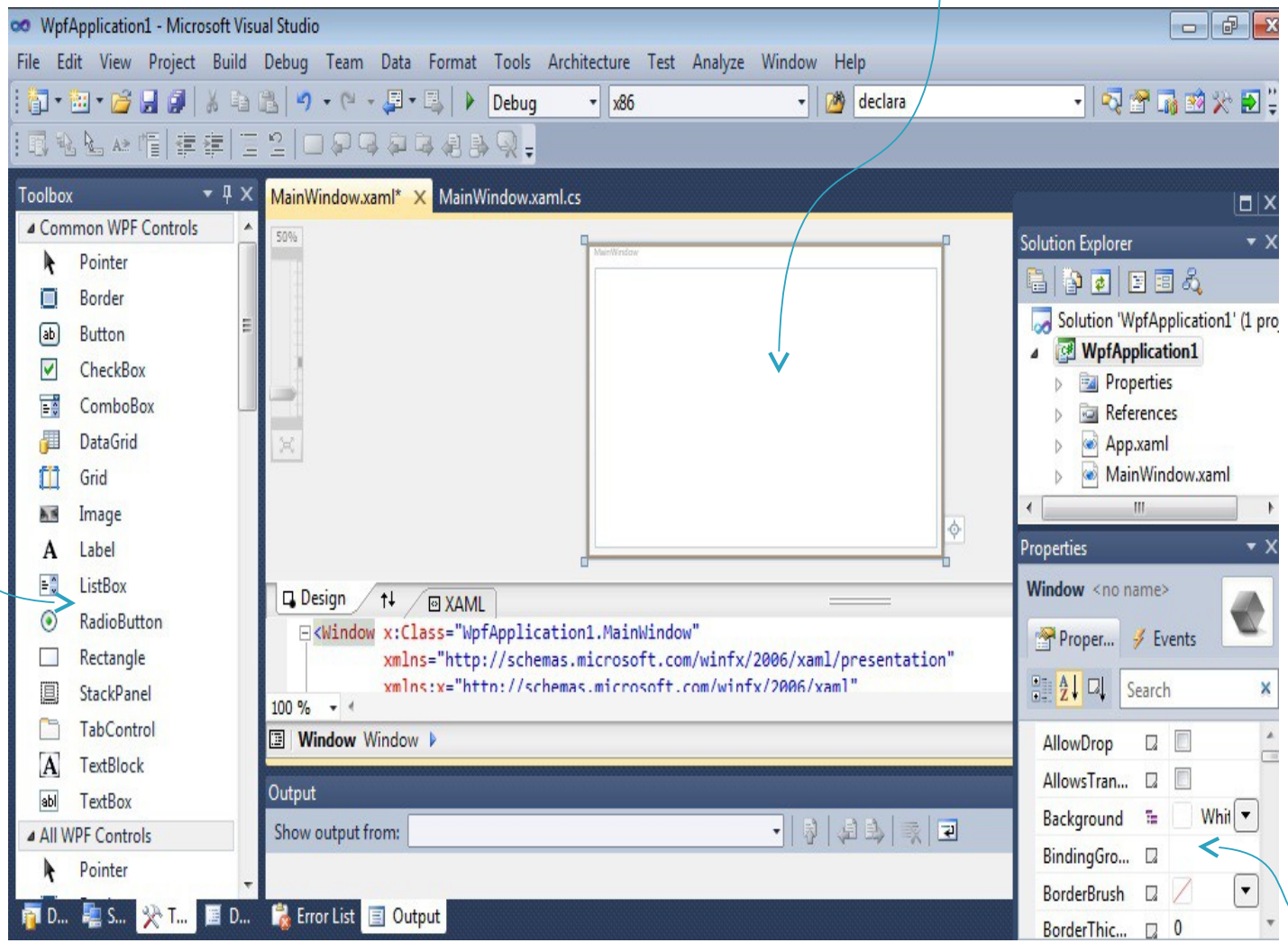
Możesz przełączać się pomiędzy oknem “XAML” a **“Desing”** za pomocą widocznych przycisków.

Powinieneś zobaczyć okno “Toolbox” w nim znajduje się lista kontrolek dostępnych w WPF.

W Solution Explorer widzisz pliki, które są dostępne w tym projekcie. Najważniejszy z nich to **“MainWindows.xaml”** , który oglądasz obecnie. Jest to plik z opisem wyglądu głównego okna. Plik z kodem w C# znajduje się pod nim.

Największą zauważalną różnicą jest wygląd projektanta formularzy.

Nowy zestaw kontrolek




Właściwości wyglądają zupełnie inaczej. To dlatego, że używamy ich do zmiany atrybutów w pliku XAML, a nie modyfikacji właściwości obiektów.

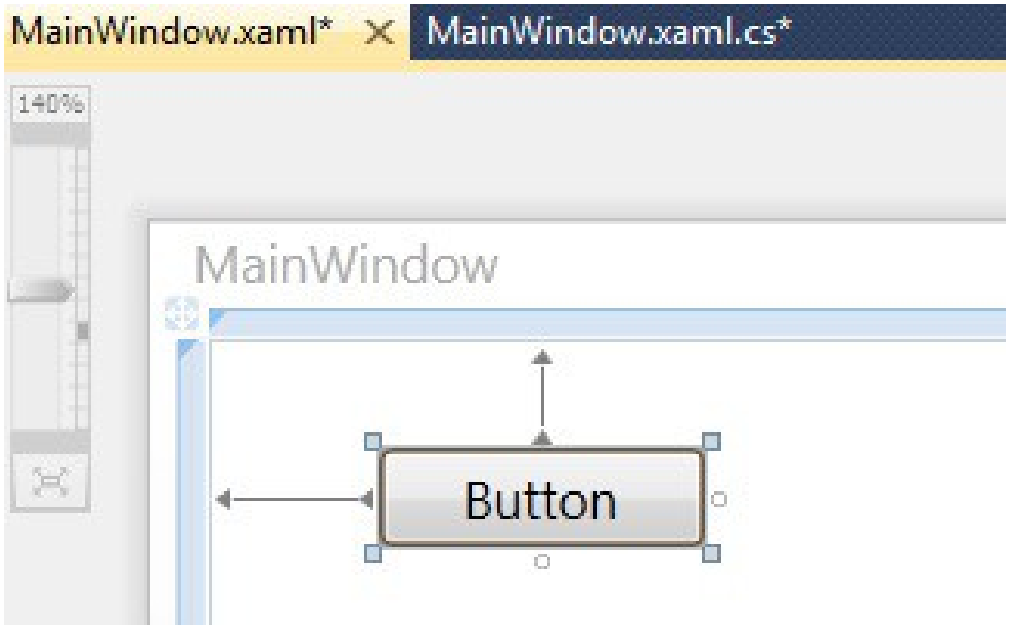
Dalej widzisz deklaracje klasy **MainWindow**, która odnosi się do okna głównego. Wewnątrz tej klasy jest tylko jej konstruktor. Zawiera on w sobie głównie kod potrzebny do  inicjalizacji klasy.

Wewnątrz konstruktora jest wywoływana metoda **“InitializeComponent()”,** która zajmuje się scalaniem widoku aplikacji XAML z jej kodem w C#.

```
namespace WPF
{
    /// <summary>/// Interaction logic for MainWindow.xaml ///
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

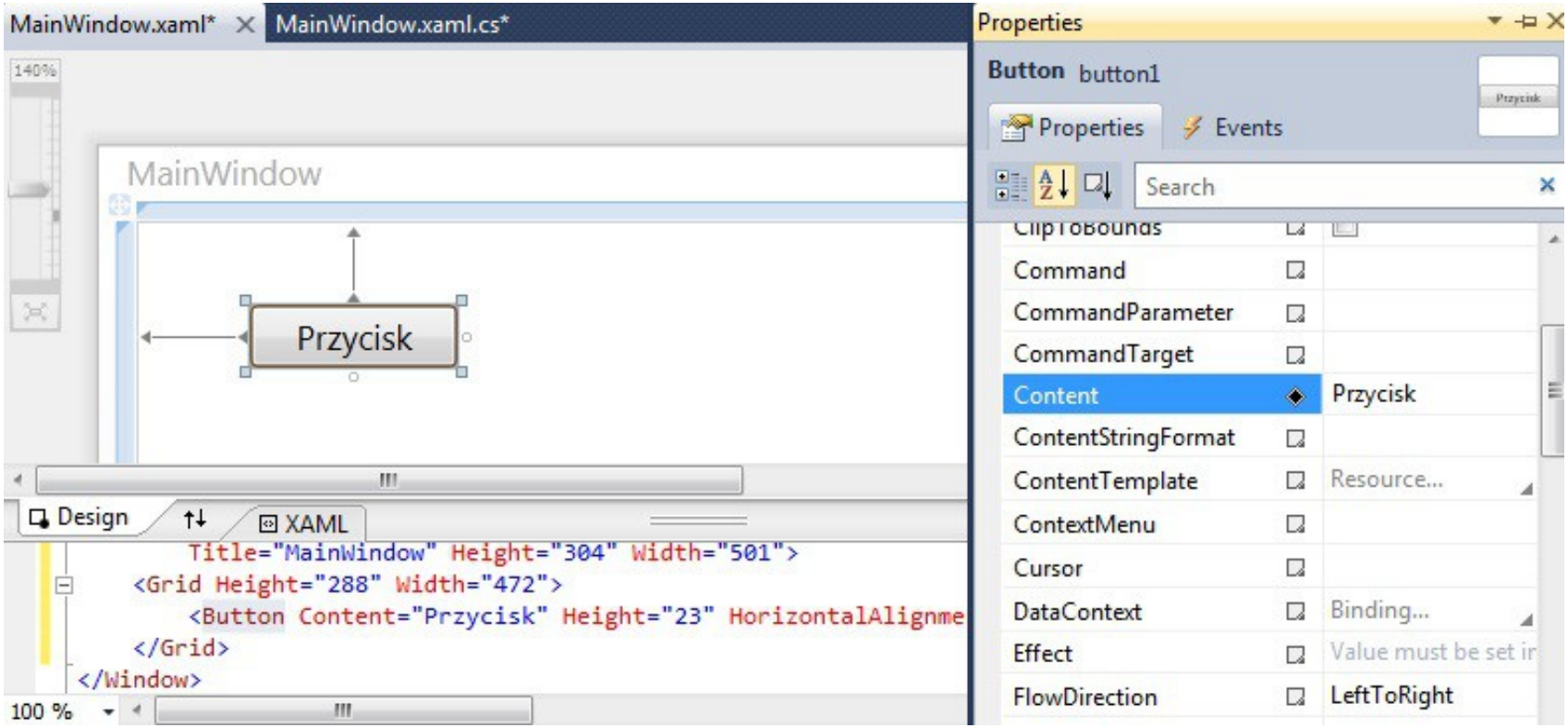
# Dodawanie kontrolek

Najłatwiejszy sposób dodania kontrolki polega na przyciśnięciu jej w menu “Toolbox” i upuszczeniu jej w oknie MainWindow.

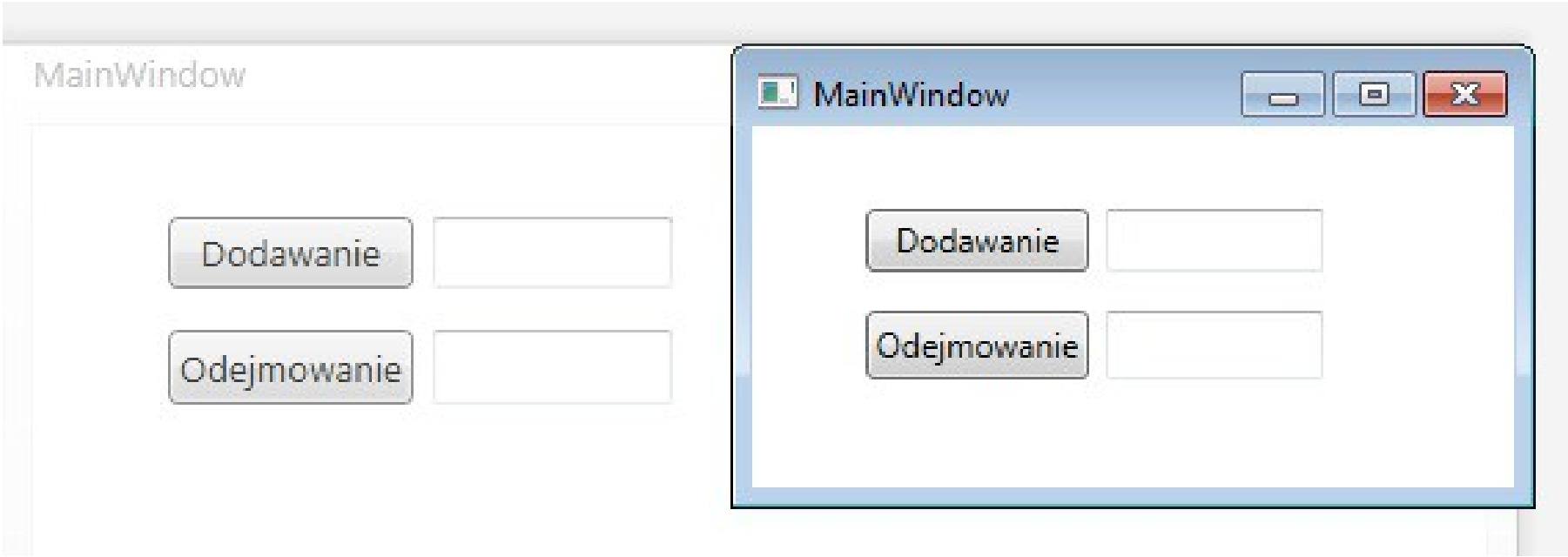


Przeciągnęliśmy przycisk z okna Toolbox na formularz. Gdyby była to aplikacja Windows Forms, do Form1.Designer.cs zostałby dodany kod, który wstawia kontrolkę do obiektu Form1. WPF jest inne – korzysta z opartego na XML języka zwanego XAML i w nim właśnie definiowany jest interfejs użytkownika.

Jeśli upuściłeś kontrolkę w złym miejscu możesz ją wciąż przemieszczać w oknie w ten sam sposób.
Edycja właściwości kontrolki może być zrobiona w XAML ,albo za pomocą okna “Properties”.



Powtarzając te czynności możesz stworzyć następujący layout aplikacji składający się z 4 przycisków i dwóch TextBox-ów. Uruchamiając aplikacje bez debugowanie za pomocą skrótu Ctr-F5 możesz zobaczyć jak aplikacja wygląda, ale obecnie nie robi ona nic.

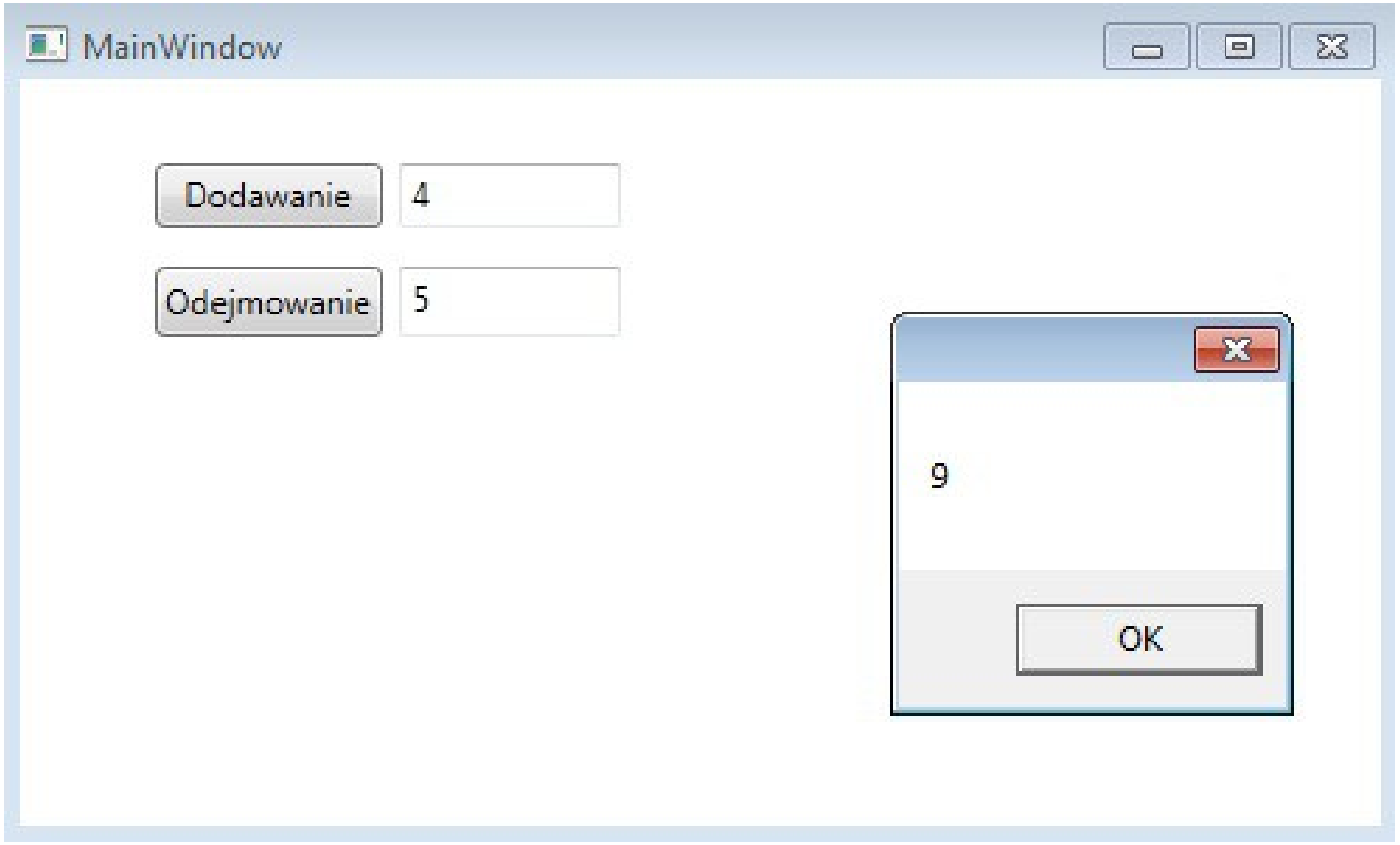


# Dodawanie zdarzeń

Najłatwiejszym sposobem dodania zdarzenia “kliknięcia” dla kontrolki *button* jest dwukrotne kliknięcie na nią. W ten sposób stworzysz zdarzenie  do kodu w C# , które będzie miało nazwę kontrolki plus **“_Click”.** Kod wewnątrz niej będzie się wykonywał tylko wtedy, gdy użytkownik kliknie na dany przycisk.  Napiszmy poniższy kod:

```
 int a, b, c;

private void button1_Click(object sender, RoutedEventArgs e)
{
      a = int.Parse(TXB_1.Text);
      b = int.Parse(TXB_2.Text);
      c = a + b;
      MessageBox.Show(c.ToString());
}
```

Efekt końcowy



# Dziękuję za uwagę