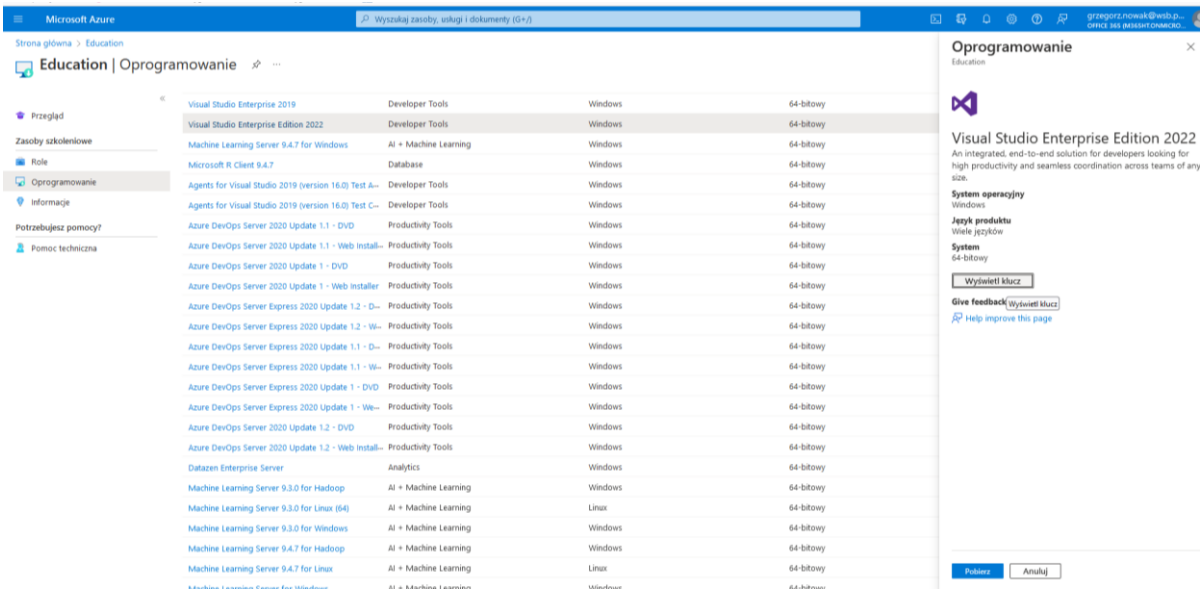
Visual Studio oraz inne oprogramowanie dla studentów jest do pobrania ze strony portal.azure.com

Następnie „Wszystkie zasoby”

Uzyskiwanie dostępu do korzyści dla studentów ->Eksploruj -> Oprogramowanie

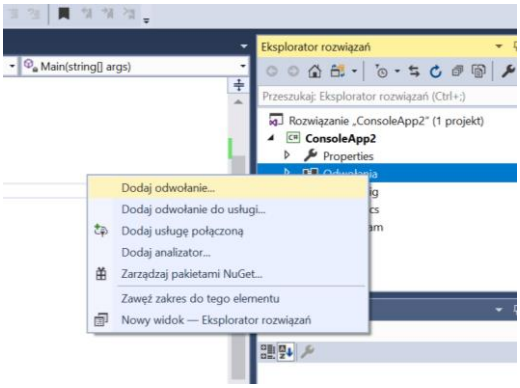


Aplikacje z interfejsem graficznym.

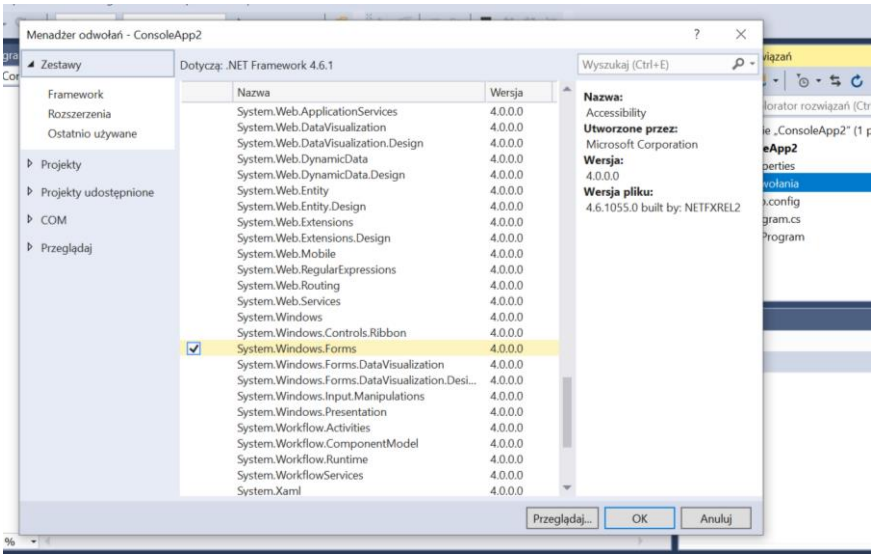
Kod wpiszemy "ręcznie", nie korzystając z pomocy narzędzi wizualnych, takich jak edytor form pakietu Visual Studio.
Dodatkowo będzie trzeba poinformować kompilator o tym, że chcemy korzystać z klas zawartych w przestrzeni `System.Windows.Forms`.
W związku z tym na początku kodu programów pojawi się dyrektywa using

```
using System.Windows.Forms;
```

Zanim ją jednak dodamy należy dodać odwołanie
W eksploratorze rozwiązań klikamy prawym przyciskiem myszki na
„Odwołania - > Dodaj odwołanie”



Zaznaczamy System.Windows.Forms i zaznaczamy ok.



Dodajemy dyrektywę using

```
using System.Windows.Forms;
```

Do utworzenia podstawowego okna będzie potrzebna klasa Form z platformy .NET. Należy utworzyć jej instancję oraz przekazać ją jako argument w wywołaniu instrukcji

`Application.Run()`.

A zatem w metodzie Main powinna znaleźć się linia:

```
Application.Run(new Form());
```

Uruchomić program

Zauważmy jednak, że taki sposób modyfikacji zachowania okna sprawdzi się tylko w przypadku wyświetlania prostych okien dialogowych. Typowa aplikacja z reguły jest znacznie bardziej skomplikowana oraz zawiera wiele różnych zdefiniowanych przez nas właściwości. Najlepiej byłoby więc wyprowadzić swoją własną klasę pochodną od Form. Tak też właśnie najczęściej się postępuje.

Stwórzmy klasę MyForm dziedziczącą po klasie Form

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ConsoleApp2
{
    class MyForm : Form
    {
    }
}
```

W metodzie Main zapiszmy:

```
Application.Run(new MyForm());
```

uruchomić program.

W klasie MyForm stwórzmy konstruktor, który umieszcza na naszej formie button.

```
class MyForm : Form
{
    private Button button1;
    public MyForm()
    {
        button1 = new Button();
        button1.Left = 40;
        button1.Top = 50;
        button1.Text = "Wyjście";
        button1.Height = 50;
        button1.Width = 80;
        this.Controls.Add(button1);
    }
}
```

Uruchomić program.

Aplikacje z graficznym interfejsem użytkownika przeznaczone na pulpit systemu Windows można tworzyć w Visual Studio, korzystając z dwóch bibliotek kontrolek: tradycyjnej Windows Forms oraz Application WPF.

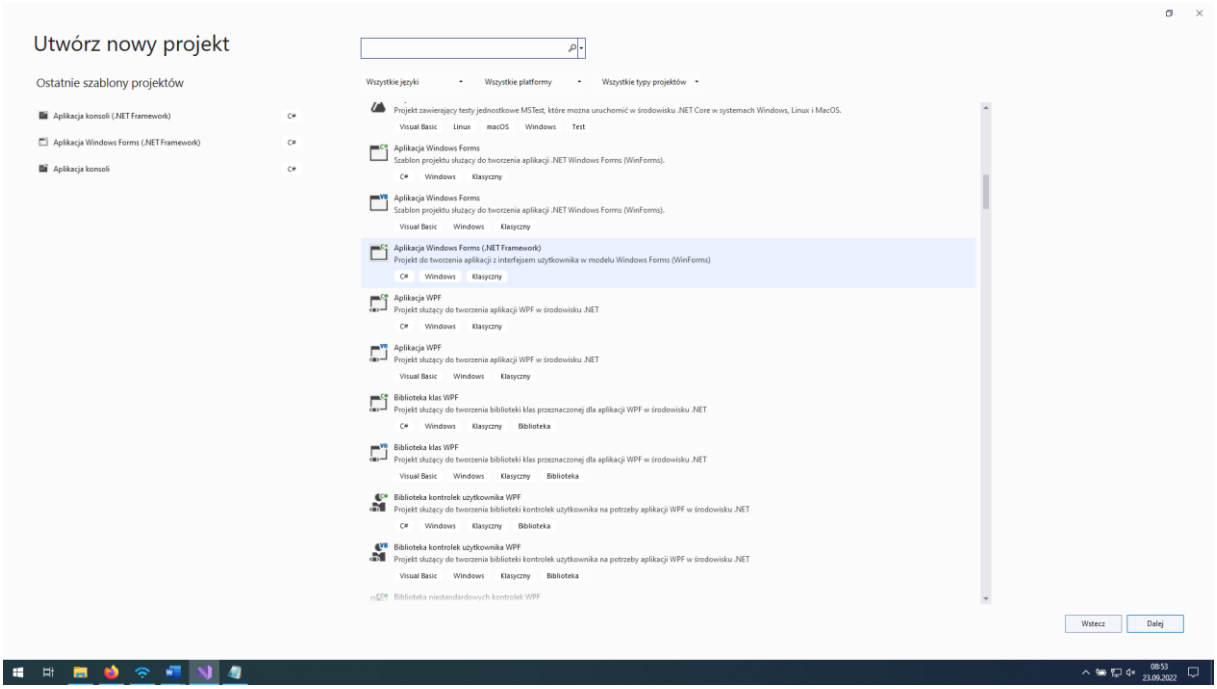
Interfejs aplikacji Windows Forms można w Visual Studio zaprojektować w pełni „wizualnie", składając go przy użyciu myszy z kontrolek umieszczanych na podglądzie okna i modyfikując ich własności udostępnione w podoknie *Właściwości*. Po utworzeniu w ten sposób interfejsu, można przejść do kolejnego etapu, w którym określane są reakcje aplikacji na różnego rodzaju akcje jej użytkownika, a więc powinien np. zdefiniować funkcję wykonywaną po kliknięciu myszą przycisku umieszczonego na oknie, po przesunięciu suwaka lub wybraniu pozycji w menu.

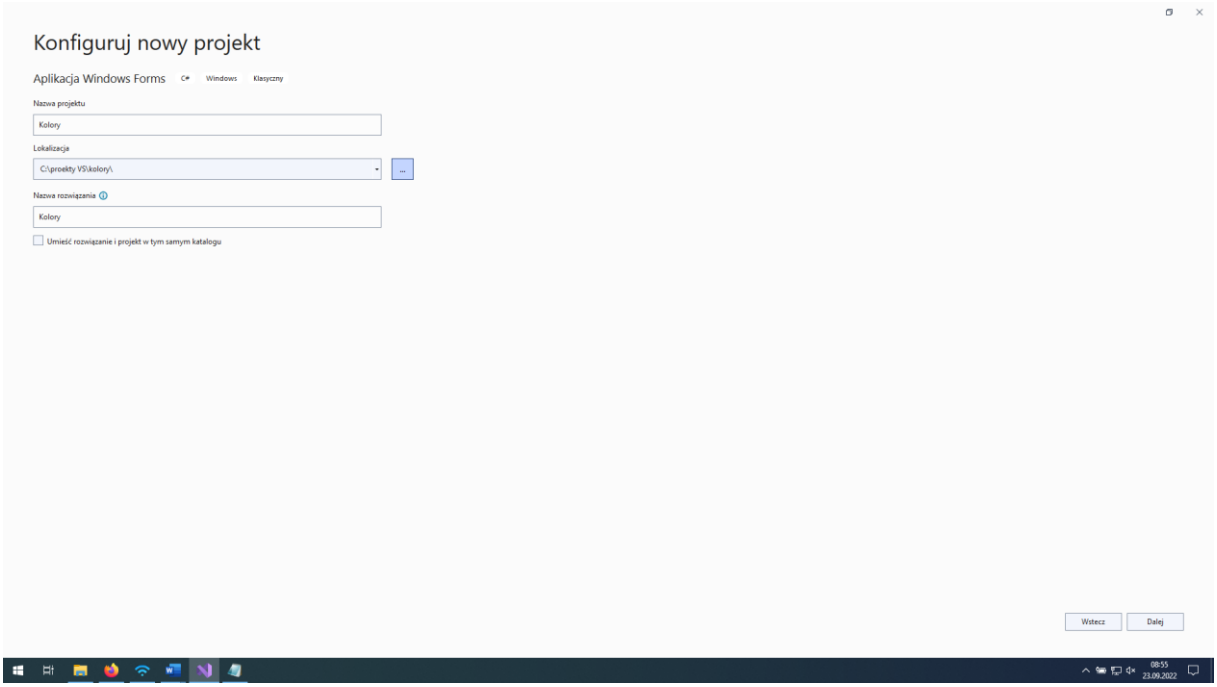
Pierwsza aplikacja Windows Forms, którą przygotujemy, będzie umożliwiać użytkownikowi zmianę koloru panelu za pomocą trzech suwaków. Jej działanie ma z założenia być dość proste.

Tworzenie projektu

Utwórzmy projekt aplikacji okienkowej (typu Aplikacja Windows Forms).

W polu nazwa wpiszmy nazwę naszego projektu „Kolory” oraz w lokalizacji wybierzmy odpowiednią ścieżkę





Utworzyliśmy projekt o nazwie *Kolory*, którego pliki zostały umieszczone we wskazanym przez nas katalogu. Plik konfiguracyjny *Kolory.sln* znajduje się w katalogu nadrzędnym.

Plik projektu nazywa się *Kolory.csproj*. Towarzyszy mu sporo dodatkowych plików zawierających kod C#, w tym *Program.cs* z metodą Main, będącą głównym punktem wejściowym programu — identycznie jak w przypadku aplikacji konsolowych. To ta metoda tworzy obiekt formy, uruchamia pętlę główną.

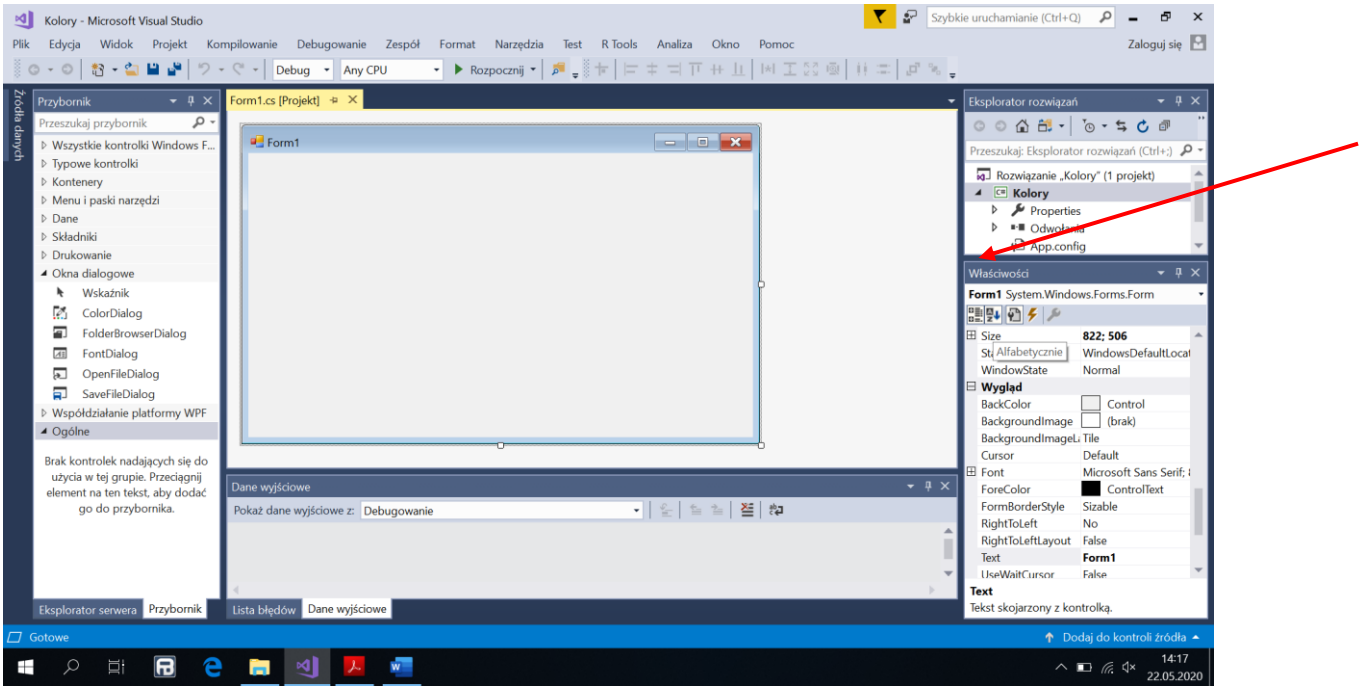
Inne pliki to m.in. *Form1.cs* i *Form1.Designer.cs* zawierające definicję klasy opisującej okno aplikacji. Ponadto można znaleźć jeszcze kilka plików pomocniczych z kodem źródłowym.

W głównym oknie dodana została nowa zakładka o nazwie *Form1.cs*, a na niej widzimy obraz pustej jeszcze formy.

Z lewej strony znajduje się podokno zawierające zbiór kontrolek „Przybornik" (*Toolbox*). Z kolei z prawej strony widoczne jest podokno zawierające listę wszystkich plików rozwiązania i projektu. Pod nim jest okno „Właściwości" (*Properties)*, które w przypadku zamknięcia możemy otworzyć, naciskając klawisz *F4*.

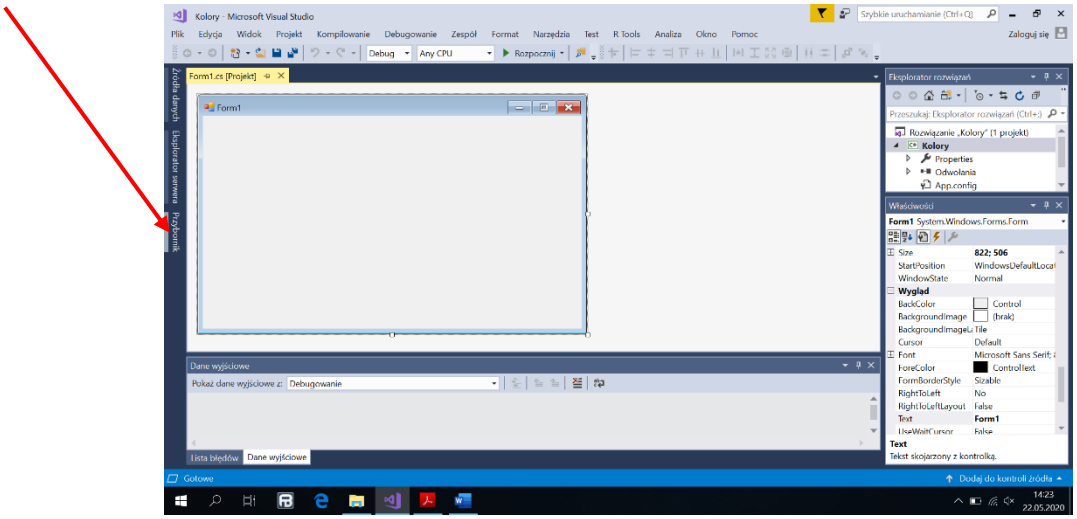
Jeżeli klikniemy podgląd formy, okno właściwości będzie pokazywać jej właściwości (klasa Form1).

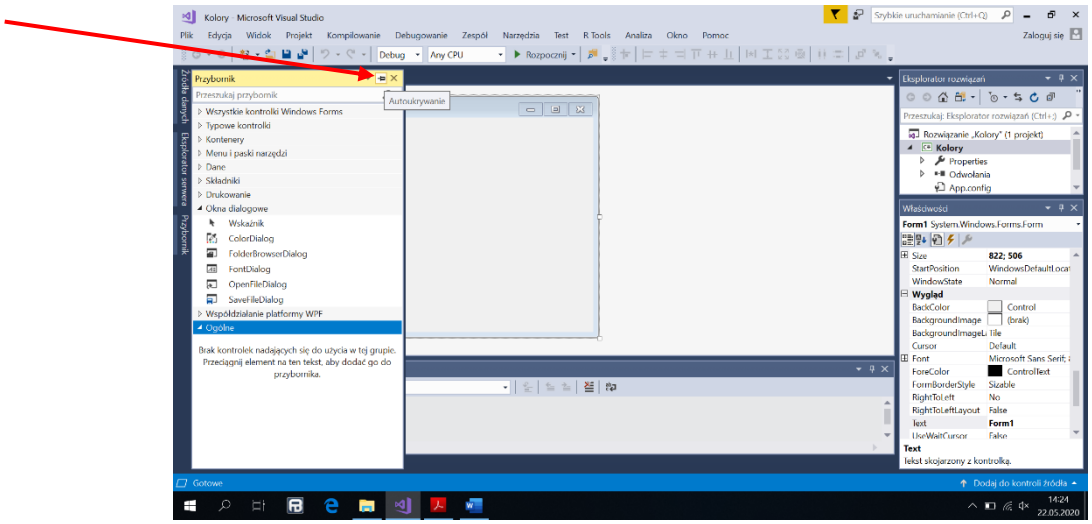
Możemy wybrać dwa sposoby ułożenia właściwości i zdarzeń w oknie właściwości: alfabetyczny oraz według kategorii.



Okno Przybornik (*Toolbox)* możemy zadokować tak aby było stale widoczne w widoku projektowania. Kliknijmy przycisk Przybornik (*Toolbox)* znajdujący się na pasku przy lewej krawędzi okna.

Następnie kliknijmy ikonę w kształcie pinezki, aby „przypiąć” rozwinięte okno na stałe i zapobiec jego ukrywaniu się.



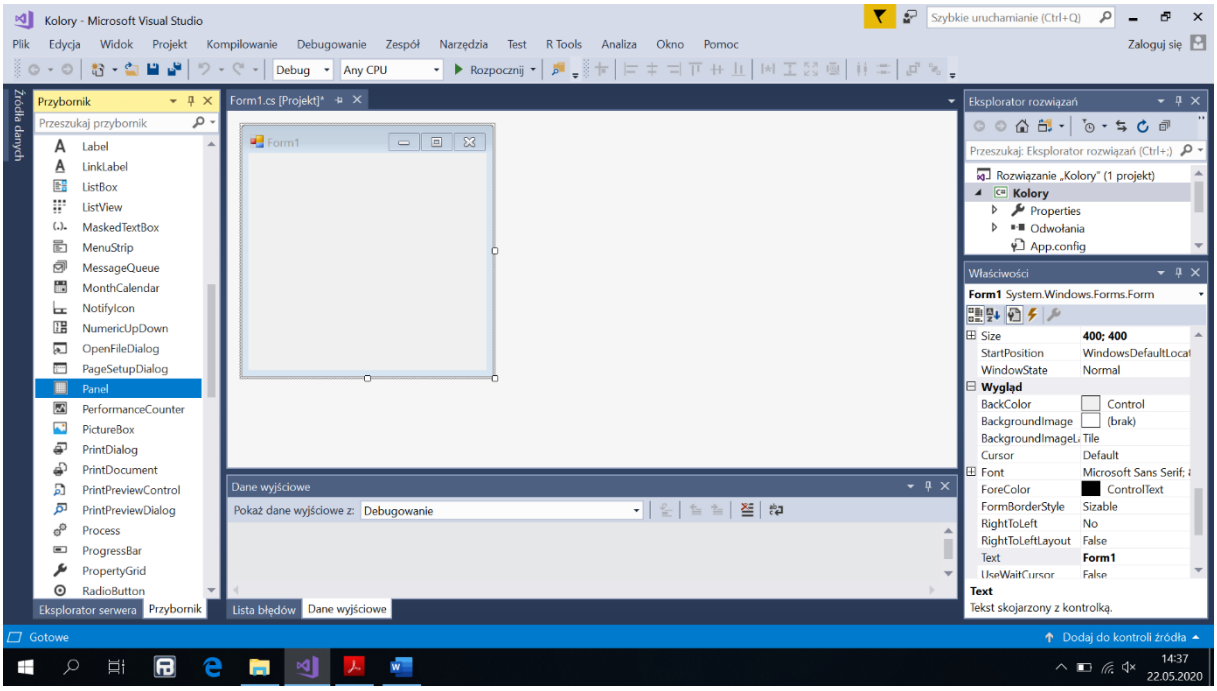


Okno *Toolbox* jest bardzo ważne. W nim zgromadzone są kontrolki, czyli „klocki”, z których złożymy interfejs aplikacji. Znajdziemy tam m.in. tak typowe kontrolki jak napisy, przyciski, pola edycyjne, pola opcji, różnego typu listy, a także komponenty bazodanowe i typowe okna dialogowe. Wszystkie te komponenty pogrupowane są w kilku rozwijanych zakładkach.
W taki sam sposób zadokujmy okno Właściwości - *Properties*

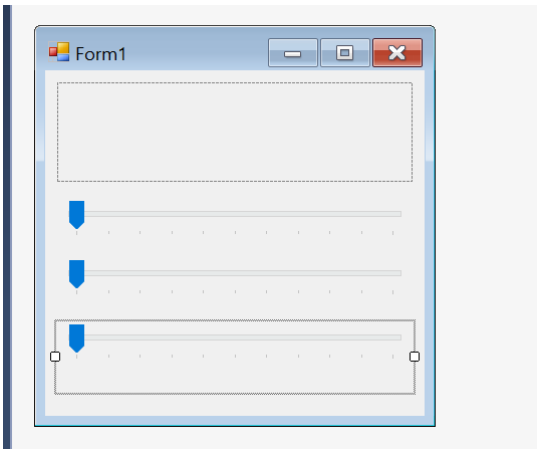
Tworzenie interfejsu za pomocą komponentów Windows Forms

Zmieńmy wielkość formy w widoku projektowania (zakładka *Form1.cs [Projekt]*) do rozmiarów 400×400.

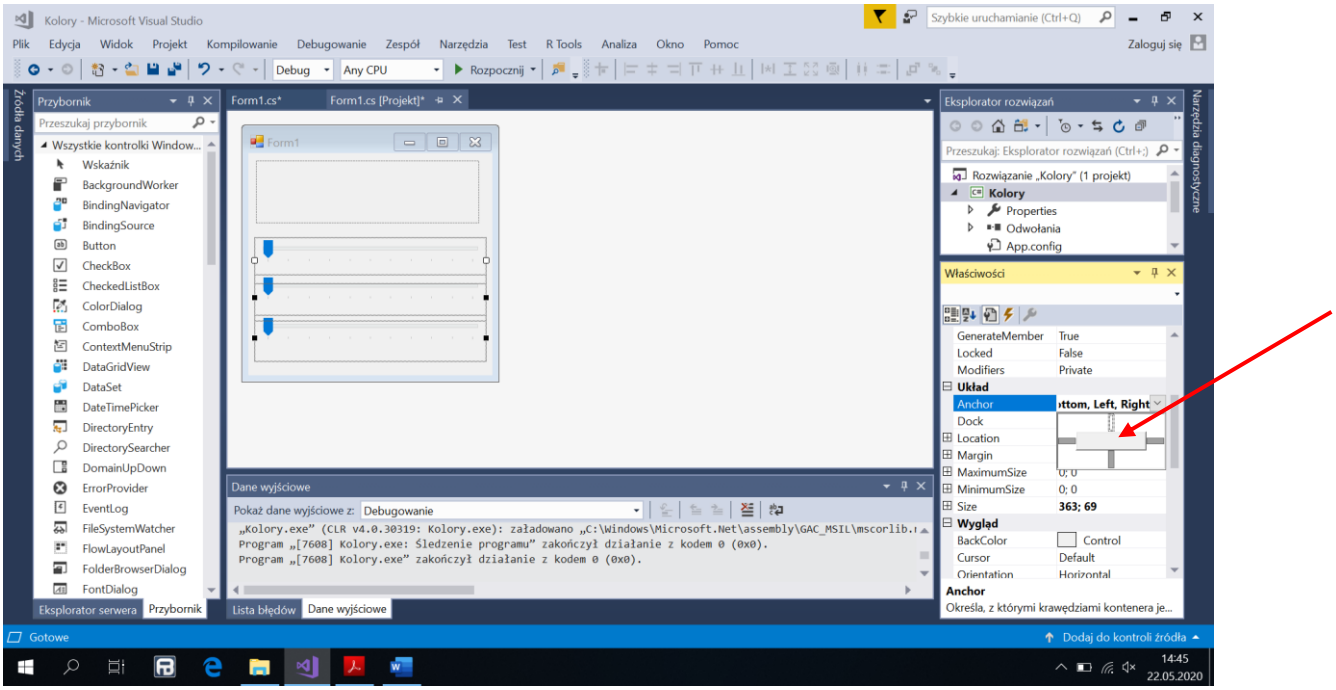
Rozwiń grupę „Wszystkie kontrolki Windows” - *All Windows Forms* w Przyborniku.
Odnajdź w niej komponent Panel i zaznacz go. Następnie umieść go na podglądzie formy, przeciągając myszą (z wciśniętym lewym przyciskiem) nad obszarem, na którym ma być umieszczony.



Postępując tak samo, umieść na formie trzy suwaki (kontrolka TrackBar) i rozmieść je tak jak poniżej



Następnie zaznacz wszystkie trzy suwaki (przytrzymując klawisz *Shift*) i za pomocą okna własności zmodyfikuj ich własność Anchor (zakotwiczenie) tak, aby kontrolki zaczepione zostały do lewej, dolnej i prawej krawędzi okna.



Natomiast panel zakotwicz do wszystkich czterech krawędzi formy.

Po zaznaczeniu komponentu w Przyborniku kursor myszy zmienia się i pokazuje ikonę wybranego komponentu. Po umieszczeniu go na formie kursor wraca do normalnego wyglądu, co oznacza, że żaden komponent nie jest aktualnie wybrany.

Własność Anchor kontrolek jest bardzo wygodnym narzędziem zwiększającym elastyczność interfejsu. Ustawienie zakotwiczenia między kontrolką a krawędzią okna wymusza stałą odległość tychże, co będzie ważne podczas zmiany rozmiaru okna. Własność Anchor nie przysuwa kontrolki do całej krawędzi, a jedynie „pilnuje" odległości. Za przysuwanie kontrolki do krawędzi odpowiada własność Dock.

Umieszczone na formie komponenty można dopasowywać. Można zmieniać ich rozmiar i położenie, modyfikując je bezpośrednio na podglądzie formy.

Poza tym wszystkie ich własności dostępne są w oknie Właściwości.

Podgląd naszej formy widoczny jest w zakładce *Form1.cs [Projekt]*. Możemy nacisnąć klawisz *F7*, co spowoduje przeniesienie do edytora kodu klasy opisującej projektowaną formę (nowa zakładka *Form1.cs* w oknie głównym Visual Studio). Naciśnięcie klawisza *Shift+F7* przeniesie nas z powrotem do widoku projektowania formy.

Kod klasy opisującej projektowaną przez nas formę znajduje się w dwóch plikach (inaczej niż w projektach aplikacji konsolowej).
Rozdzielenie przez Visual C# klasy na dwa pliki lub więcej możliwe jest dzięki słowu kluczowemu partial. Zasadnicza część klasy znajduje się w pliku *Form1.cs*. Obejmuje definicję klasy Form1 wraz z definicją jej konstruktora - funkcją składową o nazwie identycznej z nazwą klasy, uruchamianą po utworzeniu obiektu tej klasy. Klasa ta należy do przestrzeni nazw Kolory
Tu będziemy umieszczali wszystkie metody zdarzeniowe.

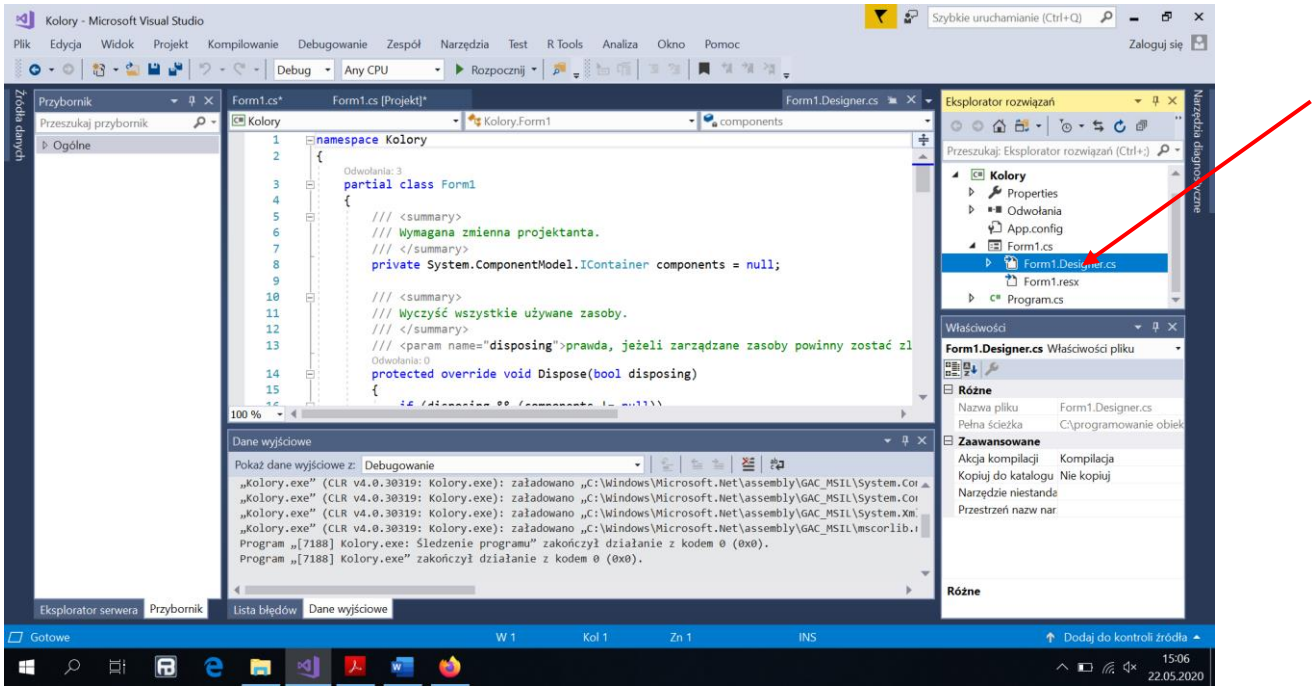
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Kolory
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

W przestrzeni nazw Kolory zadeklarowana jest jedna klasa o nazwie Form1. Jest to klasa publiczna, czyli dostępna także spoza przestrzeni nazw Kolory.
Klasa Form1 dziedziczy po klasie Form, która jest prototypem typowego pustego okna z paskiem tytułu, z możliwością zmiany rozmiaru i kilkoma funkcjonalnościami typowymi dla okien systemu Windows.
Obecnie w tej części klasy Form1, która umieszczona jest w pliku *Form1.cs*, znajduje się jedynie definicja konstruktora.

W tej chwili jedynym zadaniem konstruktora klasy Form1 jest wywołanie metody InitializeComponents, której definicja umieszczona jest w pliku *Form1.Designer.cs* razem z kodem tworzonym automatycznie przez Visual C#.

Plik ten można wczytać do edytora, korzystając z Eksporatotra rozwiązań. Należy rozwinąć gałąź *Form1.cs* i dwukrotnie kliknąć odpowiedni plik.



Dzięki podzieleniu klasy kod, który modyfikuje programista, jest oddzielony od kodu tworzonego automatycznie w momencie wprowadzania zmian w widoku projektowania. To znacznie zwiększa przejrzystość tej części, która jest przeznaczona do edycji, ułatwia panowanie nad całością, a przede wszystkim zapobiega niezamierzonej modyfikacji kodu odpowiedzialnego za wygląd interfejsu aplikacji.

```
namespace Kolory
{
    partial class Form1
    {
        /// <summary>
        /// Wymagana zmienna projektanta.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Wyczyść wszystkie używane zasoby.
        /// </summary>
        /// <param name="disposing">prawda, jeżeli zarządzane zasoby powinny zostać
zlikwidowane; Fałsz w przeciwnym wypadku.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Kod generowany przez Projektanta formularzy systemu Windows

        /// <summary>
        /// Metoda wymagana do obsługi projektanta — nie należy modyfikować
        /// jej zawartości w edytorze kodu.
        /// </summary>
        private void InitializeComponent()
        {
            this.panel1 = new System.Windows.Forms.Panel();
            this.trackBar1 = new System.Windows.Forms.TrackBar();
            this.trackBar2 = new System.Windows.Forms.TrackBar();
            this.trackBar3 = new System.Windows.Forms.TrackBar();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar2)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar3)).BeginInit();
            this.SuspendLayout();
            //
            // panel1
            //
            this.panel1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
            | System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.panel1.Location = new System.Drawing.Point(12, 12);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(340, 110);
            this.panel1.TabIndex = 0;
            //
            // trackBar1
            //
            this.trackBar1.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.trackBar1.Location = new System.Drawing.Point(12, 147);
            this.trackBar1.Name = "trackBar1";
            this.trackBar1.Size = new System.Drawing.Size(349, 69);
            this.trackBar1.TabIndex = 1;
```

```
            //
            // trackBar2
            //
            this.trackBar2.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.trackBar2.Location = new System.Drawing.Point(12, 206);
            this.trackBar2.Name = "trackBar2";
            this.trackBar2.Size = new System.Drawing.Size(349, 69);
            this.trackBar2.TabIndex = 2;
            //
            // trackBar3
            //
            this.trackBar3.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)
            | System.Windows.Forms.AnchorStyles.Right)));
            this.trackBar3.Location = new System.Drawing.Point(12, 270);
            this.trackBar3.Name = "trackBar3";
            this.trackBar3.Size = new System.Drawing.Size(349, 69);
            this.trackBar3.TabIndex = 3;
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 20F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(373, 363);
            this.Controls.Add(this.trackBar3);
            this.Controls.Add(this.trackBar2);
            this.Controls.Add(this.trackBar1);
            this.Controls.Add(this.panel1);
            this.Name = "Form1";
            this.Text = "Form1";
            ((System.ComponentModel.ISupportInitialize)(this.trackBar1)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar2)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.trackBar3)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.TrackBar trackBar1;
        private System.Windows.Forms.TrackBar trackBar2;
        private System.Windows.Forms.TrackBar trackBar3;
    }
}
```

Zwrócić wpierw uwagę na ostatnie linie tej części klasy:

```
private Panel panel1;
private TrackBar trackBar1;
private TrackBar trackBar2;
private TrackBar trackBar3;
```

Zdefiniowane są tam prywatne pola klasy odpowiadające komponentom, które umieściliśmy na formie, projektując jej interfejs. Polecenie Panel panel1 widoczne powyżej oznacza oczywiście jedynie zadeklarowanie referencji (zmiennej obiektowej) typu Panel. Nie powstaje w tym momencie obiekt, do którego utworzenia konieczne byłoby wykorzystanie operatora new. Jest tak, ponieważ Panel jest klasą, czyli typem referencyjnym.

Na górze mamy definicję metody Form1.Dispose.

```
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
```

Nadpisuje ona metodę z klasy bazowej (tj. metodę Form.Dispose). Metoda ta jest wykorzystywana przy usuwaniu obiektu z pamięci.

Kolejną metodą klasy Form1 umieszczoną w tym pliku jest wywoływana z konstruktora metoda InitializeComponents. W niej znajdują się polecenia tworzące obiekty komponentów i konfigurujące ich własności.

```
private void InitializeComponent()
{
    this.panel1 = new System.Windows.Forms.Panel();
    this.trackBar1 = new System.Windows.Forms.TrackBar();
    this.trackBar2 = new System.Windows.Forms.TrackBar();
    this.trackBar3 = new System.Windows.Forms.TrackBar();
      …

}
```

Pierwsze linie tej metody tworzą paski przewijania za pomocą operatora new i następujących po nim wywołań konstruktorów domyślnych (tj. bezargumentowych) klas Panel i TrackBar. Kolejność tych poleceń zależy od kolejności umieszczania kontrolek na podglądzie formy.

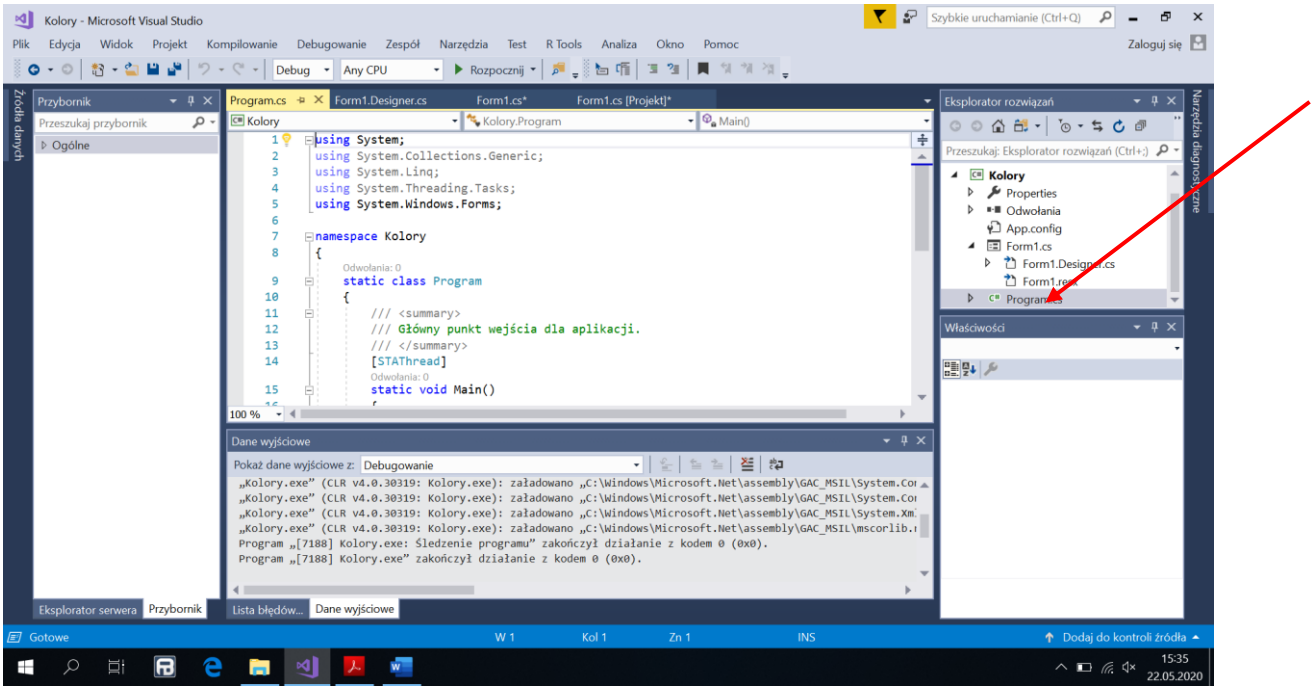
Następne linie tej metody określają kolejno: własność Anchor (omówioną wcześniej), położenie (własność Location) i rozmiar (Size) komponentu, jego nazwę oraz własność TabIndex, odpowiadającą za kolejność przejmowania przez komponenty „focusu” przy korzystaniu z klawisza *Tab* do nawigacji na formie.

```
            // panel1
             //
             this.panel1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Bottom)
             | System.Windows.Forms.AnchorStyles.Left)
             | System.Windows.Forms.AnchorStyles.Right)));
             this.panel1.Location = new System.Drawing.Point(12, 12);
             this.panel1.Name = "panel1";
             this.panel1.Size = new System.Drawing.Size(340, 110);
             this.panel1.TabIndex = 0;
```

Na końcu zebrana jest seria wywołań metody this.Controls.Add, która gotowe komponenty przypisuje do formy.

```
             this.Controls.Add(this.trackBar3);
             this.Controls.Add(this.trackBar2);
             this.Controls.Add(this.trackBar1);
             this.Controls.Add(this.panel1);
```

Istotny z punktu widzenia uruchamiania programu jest jeszcze jeden plik zawierający kod źródłowy, a mianowicie *Program.cs*. Możemy go otworzyć, korzystając z Eksploratora rozwiązań *(Solution Explorer)*.



Zawiera on (podobnie jak to miało miejsce w aplikacjach konsolowych) prostą klasę statyczną Program w której znajduje się tylko jedna metoda statyczna o nazwie Main — główne wejście do aplikacji.

```
namespace Kolory
{
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI
settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            Application.Run(new Form1());
        }
    }
}
```

To właśnie metoda Main jest wywoływana przez system i platformę .NET po próbie uruchomienia aplikacji przez użytkownika. W metodzie Main powinny znaleźć się polecenia, które zbudują okno aplikacji, a więc w naszym przykładzie polecenie utworzenia obiektu klasy Form1.

Wyrażenie [STAThread] znajdujące się bezpośrednio przed sygnaturą metody Main jest atrybutem, czyli pewną wiadomością dla kompilatora, a po skompilowaniu również dla platformy .NET, na której będzie uruchomiona aplikacja *Kolory*. W tym przypadku atrybut informuje, że aplikacja powinna komunikować się z systemem w trybie pojedynczego wątku.

W katalogu mamy jeszcze plik *AssemblyInfo.cs*. Przechowuje on informacje o numerze wersji, zastrzeżonych znakach towarowych itp. Możliwość edytowania go mamy w oknie opcji: menu *Project ->Właściwości Kolory,* zakładka *Aplikacja*, przycisk *Informacje o zestawie* (*Assembly Information)*.

Esencją „projektowania wizualnego” jest intuicyjne tworzenie interfejsu aplikacji, oraz możliwość łatwego określania reakcji aplikacji na zdarzenia. Przede wszystkim chodzi o reagowanie na czynności wykonane przez użytkownika, np. zmianę pozycji suwaka.

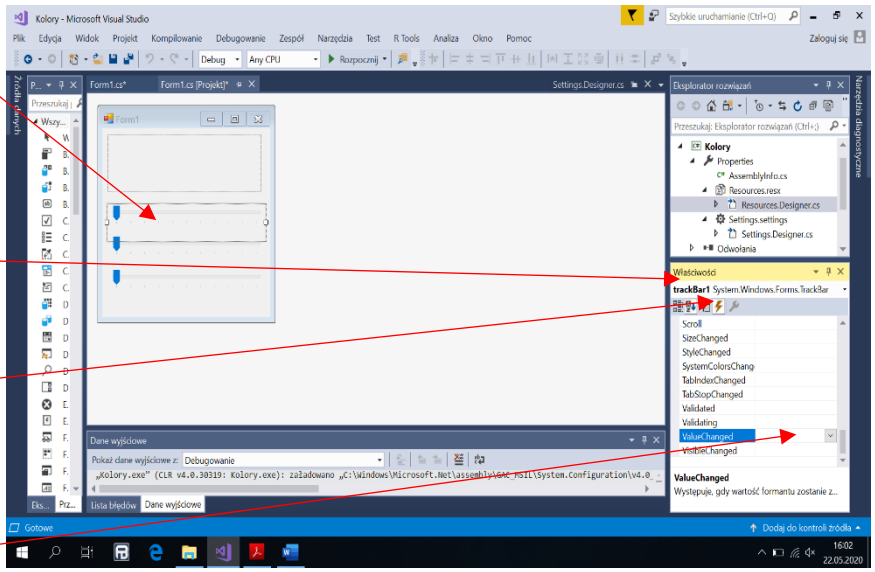
Metoda uruchamiana w przypadku wystąpienia zdarzenia kontrolki

Utwórzmy metodę związaną ze zdarzeniem ValueChanged kontrolki trackBar1.

Na podglądzie formy zaznacz pierwszy suwak o nazwie trackBar1. Nazwa zaznaczonego komponentu powinna stać się widoczna w oknie Właściwości.

Na pasku narzędzi samego okna własności kliknij ikonę *Events* (zdarzenia).

Odnajdź zdarzenie ValueChanged i dwukrotnie kliknij związane z nim pole edycyjne.



W efekcie z powrotem przenieśliśmy się do edytora kodu (zakładka Form1.cs). Kursor został ustawiony w nowo utworzonej metodzie trackBar1_ValueChanged, dokładnie w miejscu, w którym będziemy musieli napisać kod określający reakcję aplikacji na zmianę pozycji suwaka. Umieścimy tu polecenie zmieniające kolor panelu.

Po utworzeniu metody zdarzeniowej w pliku *Form1.Designer.cs* w metodzie `InitializeComponent()` znajdziemy nową linię.

```
this.trackBar1.ValueChanged += new System.EventHandler(this.trackBar1_ValueChanged);
```

Została ona dodana w momencie tworzenia metody zdarzeniowej. Jest odpowiedzialna za przypisanie metody zdarzeniowej do zdarzenia ValueChanged paska przewijania.

A teraz zmodyfikujmy metodę tak, aby po poruszeniu pierwszym suwakiem panel zmieniał kolor na fioletowy.

Do definicji metody zdarzeniowej dopisz polecenie zmiany koloru panelu,

```
private void trackBar1_ValueChanged(object sender, EventArgs e)
{
    panel1.BackColor = Color.Purple;
}
```

Skompiluj i uruchom projekt, naciskając klawisz *F5*.

Skompilowany plik *Kolory.exe* został umieszczony w podkatalogu *bin\Debug* katalogu projektu.

Jeżeli poruszymy drugim lub trzecim suwakiem, nic się nie stanie, ale jeżeli zmienimy pozycję pierwszego suwaka, kolor panelu zmieni się.

Do określenia koloru panelu w metodzie zdarzeniowej wykorzystaliśmy strukturę Color, a dokładnie jej element Purple.

Przypisywanie istniejącej metody do zdarzeń komponentów

Przypiszemy teraz metodę trackBar1_ValueChanged do zdarzenia ValueChanged dwóch pozostałych suwaków:

Przytrzymując klawisz *Ctrl*, zaznacz dwa suwaki o nazwach trackBar2 i trackBar3.

Odszukaj zdarzenie ValueChanged w oknie własności i z rozwijanej listy znajdującej się przy nim wybierz metodę trackBar1_ValueChanged.

Ponownie skompiluj i uruchom aplikację (*F5*).

Możemy się przekonać, że po wykonaniu ostatniego zadania poruszenie którymkolwiek suwakiem powoduje zmianę koloru panelu. Oznacza to, że metoda zdarzeniowa związana jest teraz ze zdarzeniem ValueChanged wszystkich trzech suwaków.

Edycja metody zdarzeniowej

Nadajmy zatem metodzie zdarzeniowej trackBar1_ValueChanged jej ostateczną postać, uzależniając kolor panelu od pozycji suwaków. W tym celu w edytorze kodu zmieniamy polecenie metody

```
panel1.BackColor = Color.FromArgb(trackBar1.Value, trackBar2.Value, trackBar3.Value);
```

Do wykonania zadania wykorzystaliśmy metodę statyczną Color.FromArgb. Metoda ta jest wielokrotnie przeciążona, tzn. możemy ją wywoływać, podając różne zestawy argumentów.

Tu wykorzystaliśmy taką jej wersję, która pobiera trzy liczby typu int.

Oznaczają one wówczas składowe R, G i B koloru, a więc intensywność komponentów czerwieni, zieleni i niebieskiego.

Argumentami metody Color.FromArgb są własności Value poszczególnych suwaków informujące o ich pozycji.

Modyfikowanie własności komponentów

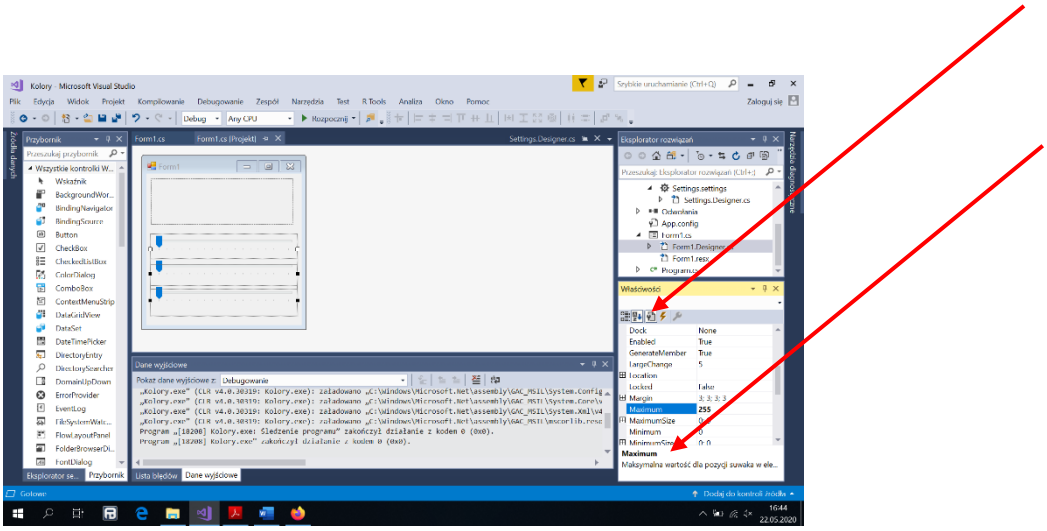
Argumenty metody Color.FromArgb mogą przyjmować wartości od 0 do 255.

Zmieńmy własność Maximum suwaków tak, aby przyjmowały wartości do 255.

Zaznacz wszystkie trzy paski przewijania (przytrzymując klawisz *Ctrl*).

Znajdź własność Maximum i zmień jej wartość na 255.

Zmień również własność LargeChange suwaków na 1 (liczba pozycji o jaką zostanie przesunięty suwak przy kliknięciu myszką).



Wywoływanie metody zdarzeniowej z poziomu kodu

Ostatnią czynnością będzie uzgodnienie koloru panelu z pozycją suwaków. W tej chwili paski przewijania wskazują na kolor czarny, a panel ma kolor identyczny z płaszczyzną formy i w efekcie jest niewidoczny. Z tego powodu po poruszeniu któregokolwiek z suwaków następuje nagła zmiana koloru. Rozwiążemy ten problem, wymuszając uzgodnienie koloru panelu tuż po uruchomieniu aplikacji przez wywołanie metody zdarzeniowej suwaków w konstruktorze formy.

W tym celu do konstruktora klasy Form1 dodamy wywołanie metody trackBar1_ValueChanged.

```
public Form1()
{
    InitializeComponent();
    this.trackBar1_ValueChanged(this, null);
}
```

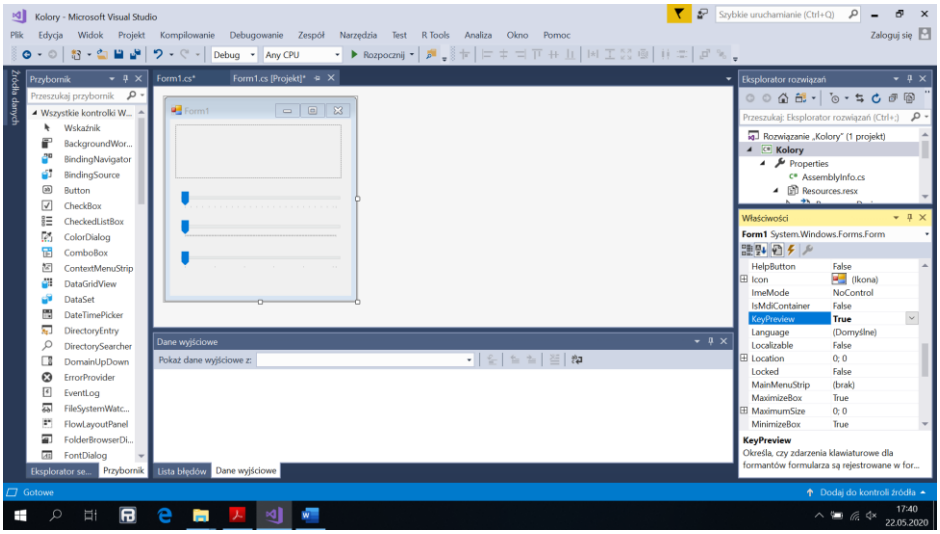
Wywołując metodę zdarzeniową, należy jako jej pierwszy argument podać obiekt, który wysyła zdarzenie. Najprościej można to zrobić, korzystając z referencji this. Drugi argument pozwala przekazać dodatkowe informacje do metody. Podaliśmy pustą referencję null. Nie ma to w naszym przypadku większego znaczenia, bo w metodzie zdarzeniowej trackBar1_ValueChanged w ogóle nie korzystamy z jej argumentów.

Reakcja aplikacji na naciskanie klawiszy

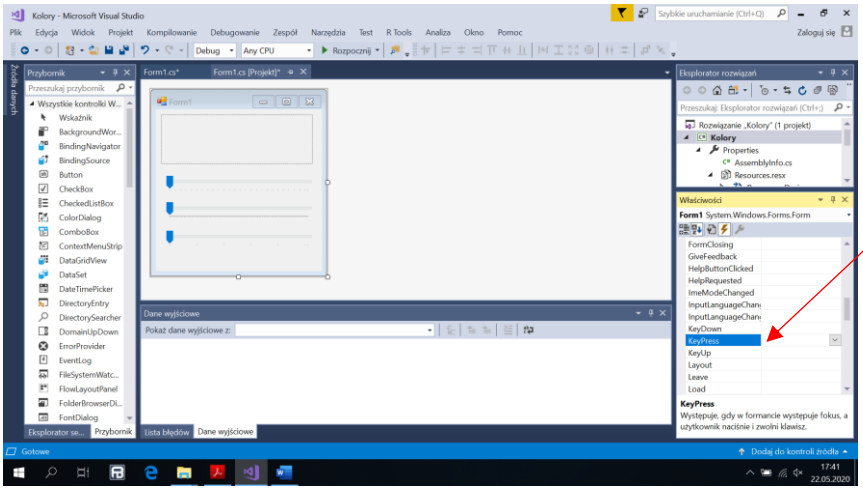
Do obsługi aplikacji okienkowej można oprócz myszy używać klawiatury. Często spotykanym rozwiązaniem jest np. zamykanie okna aplikacji po naciśnięciu przez użytkownika klawisza *Esc*. Aby uzyskać taki efekt w naszej aplikacji, należy wykorzystać zdarzenie formy KeyPress, które sygnalizuje naciśnięcie (wciśnięcie i zwolnienie) któregokolwiek z klawiszy poza klawiszami specjalnymi (tj. *Ctrl*, *Alt* i *Shift*).

Zaznacz formę, (najłatwiej kliknąć jej pasek tytułu).

W oknie Właściwości, zmień własność KeyPreview formy na true.



Zmień zakładkę na *Zdarzenia* i kliknij dwukrotnie pole przy zdarzeniu KeyPress.



W utworzonej w ten sposób metodzie zdarzeniowej umieść polecenie

```
if (e.KeyChar == '\u001B') Close();
```

Sprawdzamy, czy wciśnięty klawisz, który można odczytać z własności e.KeyChar drugiego z argumentów metody, jest klawiszem *Esc*.
Klawiszowi temu nie odpowiada żaden znak alfabetu, w zestawie znaków ASCII numer 27, czyli w układzie szesnastkowym 1B. Właśnie dlatego naciśnięty klawisz przyrównywany jest do \u001B, czyli znaku, któremu w tablicy znaków Unicode odpowiada kod 27 (1B). Jeżeli warunek zostanie spełniony, wywoływana jest metoda Close formy, która ją zamyka.

Możemy uniknąć bezpośredniego odwoływania do kodu ASCII, jeżeli zamiast zdarzenia KeyPress użyjemy zdarzenia KeyDown. W jego metodzie zdarzeniowej drugi argument zawiera między innymi własność KeyCode typu Keys – typu wyliczeniowego zawierającego wszystkie kody znaków. Wówczas warunek z instrukcji mógłby być zapisany

```
if (e.KeyCode == Keys.Escape) Close();
```