## 

## **Notatnik.NET**

Będzie to prosty projekt napisany w Windows Forms, w którym stworzymy prosty edytor — odpowiednik systemowego notatnika.

### Projektowanie interfejsu aplikacji i menu główne

Przygotowanie interfejsu wymaga staranności — jest to przecież jedyny kanał komunikacji aplikacji z jej użytkownikiem. Musimy zatem wziąć pod uwagę możliwość zmiany rozmiaru okna, zmiany kolorów w środowisku Windows, zmiany domyślnej wielkości czcionki w systemie itp. Część z tych „niespodzianek” uwzględniona została już na poziomie kontrolek, z których budujemy interfejs. Dla przykładu domyślny kolor komponentów jest ustalany na podstawie parametrów odczytanych z systemu i przechowywanych w klasie SystemColor (wartości własności ForeColor i BackColor komponentów).

Interfejs naszego notatnika nie będzie odbiegał od standardu prostych edytorów. Oznacza to, że jego podstawą będzie pole tekstowe TextBox wypełniające niemal całą wolną przestrzeń na formie (własność Dock komponentu ustawimy na Fill). Nad nim znajdzie się tylko menu główne, a pod nim — pasek stanu.

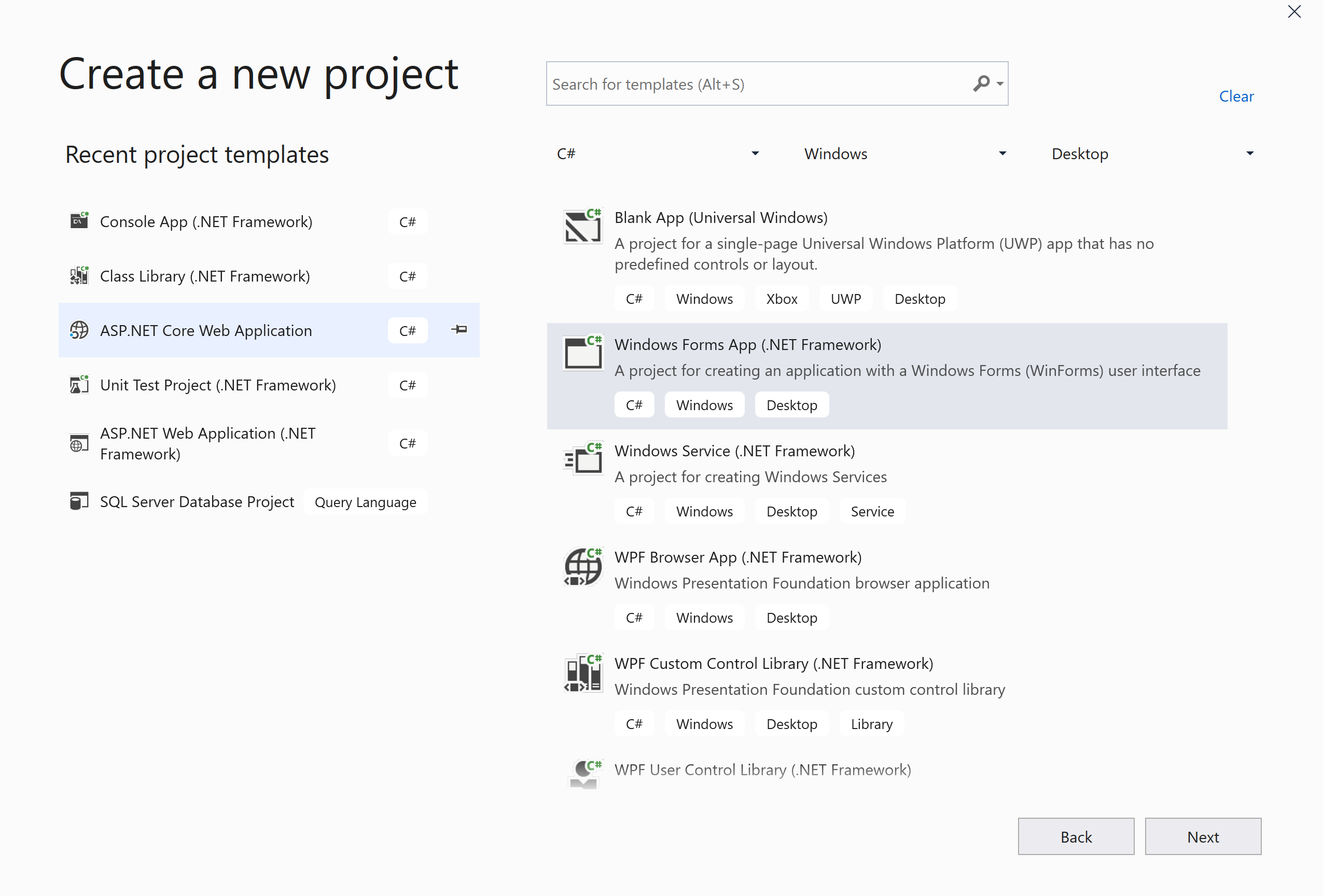
##### Tworzenie projektu aplikacji i jej interfejsu

Zacznijmy od utworzenia projektu aplikacji z polem tekstowym wypełniającym cały obszar użytkownika formy.

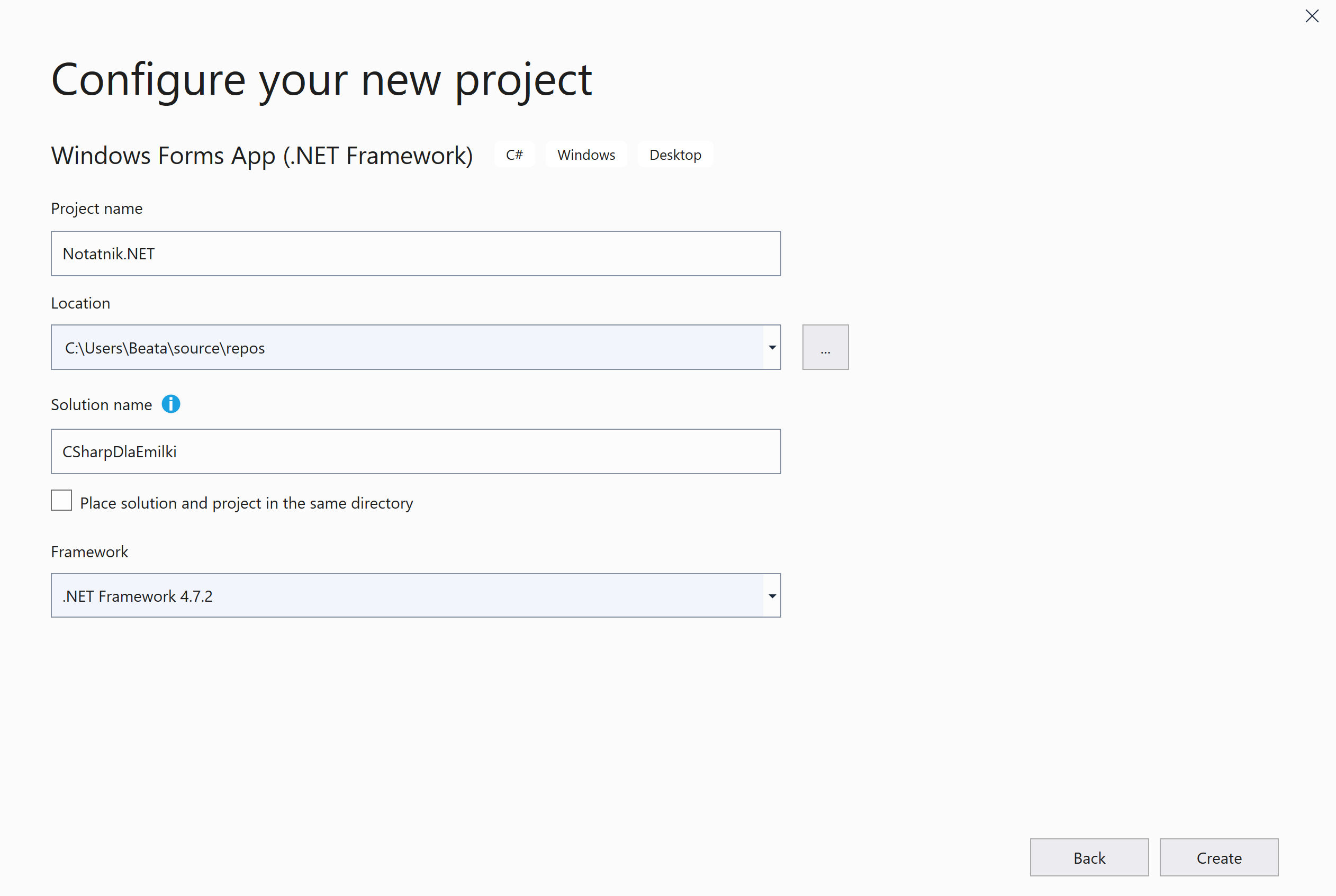
1. W środowisku Visual Studio 2019 wybierz opcję

*Create a New Project*.

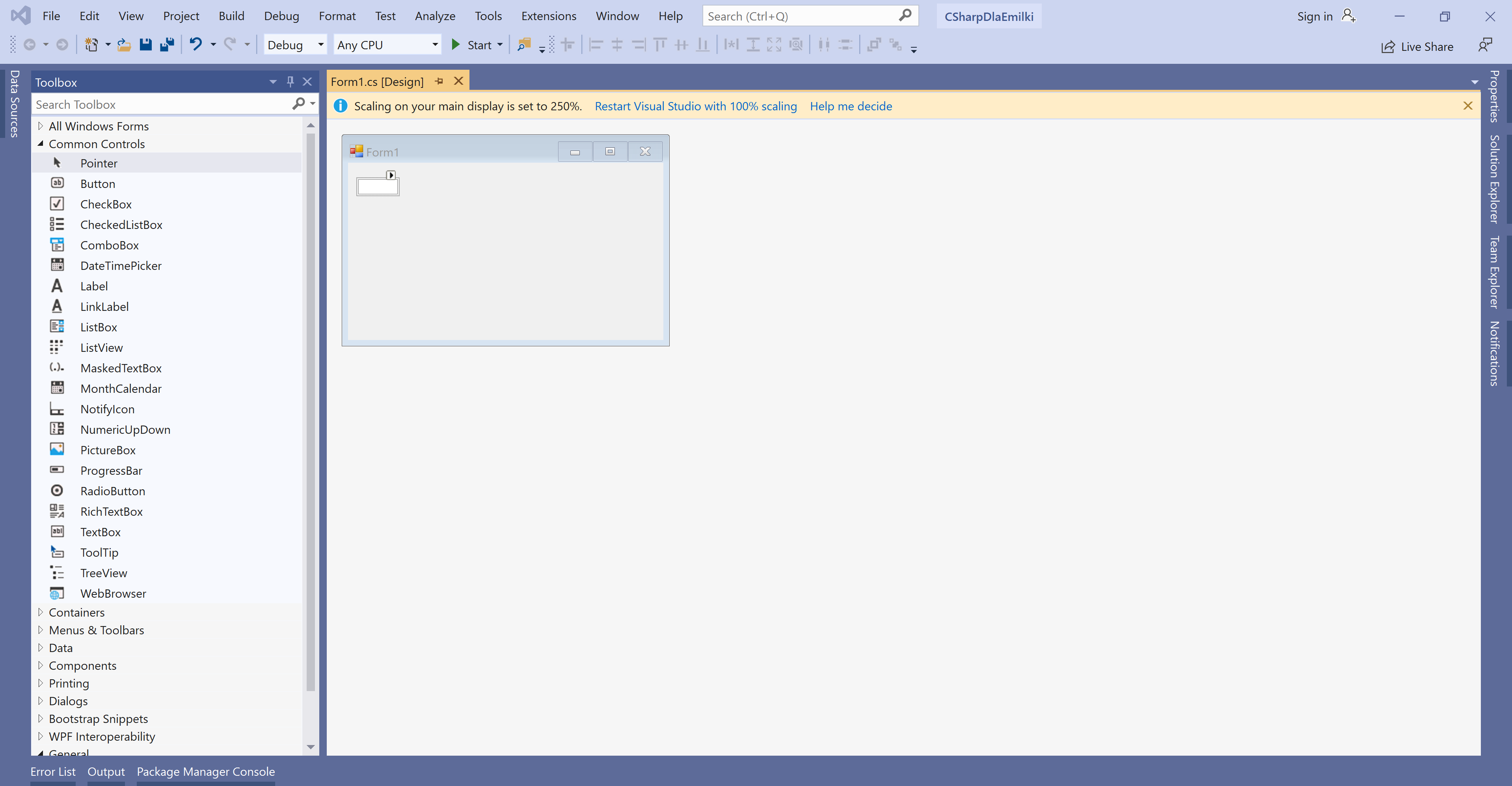
1. W oknie *New Project*:
   1. zaznacz pozycję *Windows Forms App (.Net Framework)*;
   2. kliknij *Next*;



* 1. w polu *Name* wpisz nazwę aplikacji Notatnik.NET;
  2. kliknij *Create*.



1. Następnie w widoku projektowania (zakładka *Form1.cs [Design]*), do którego zostałeś automatycznie przeniesiony po utworzeniu projektu, w palecie komponentów (**Toolbox** -podokno *Toolbox*), w kategorii *Common Controls*, odnajdź komponent TextBox i umieść go na powierzchni formularza.



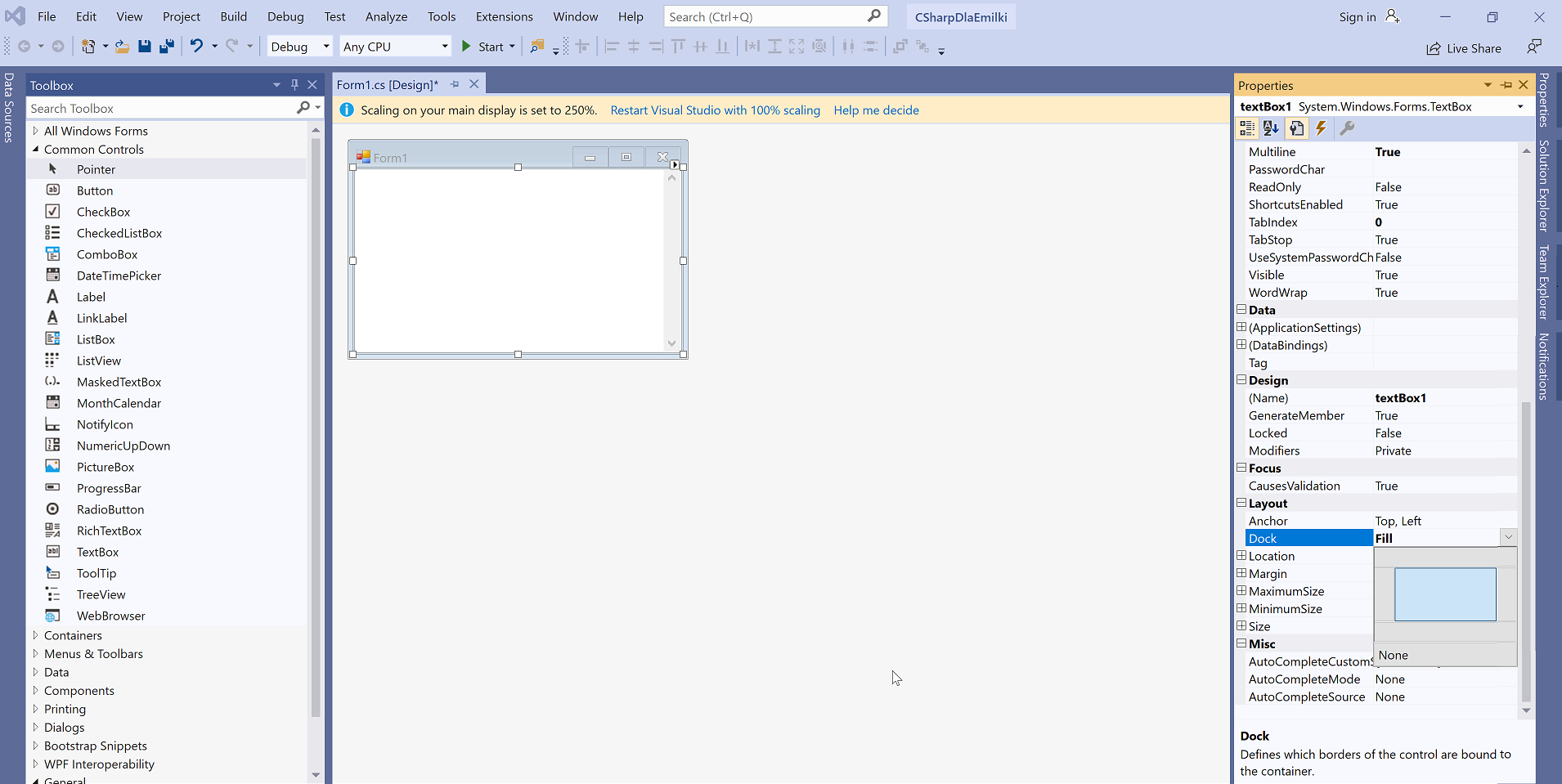
1. Zaznacz dodany do formy komponent, kliknij na nim prawym

przyciskiem myszy i wybierz opcję *Properties*. W oknie własności,

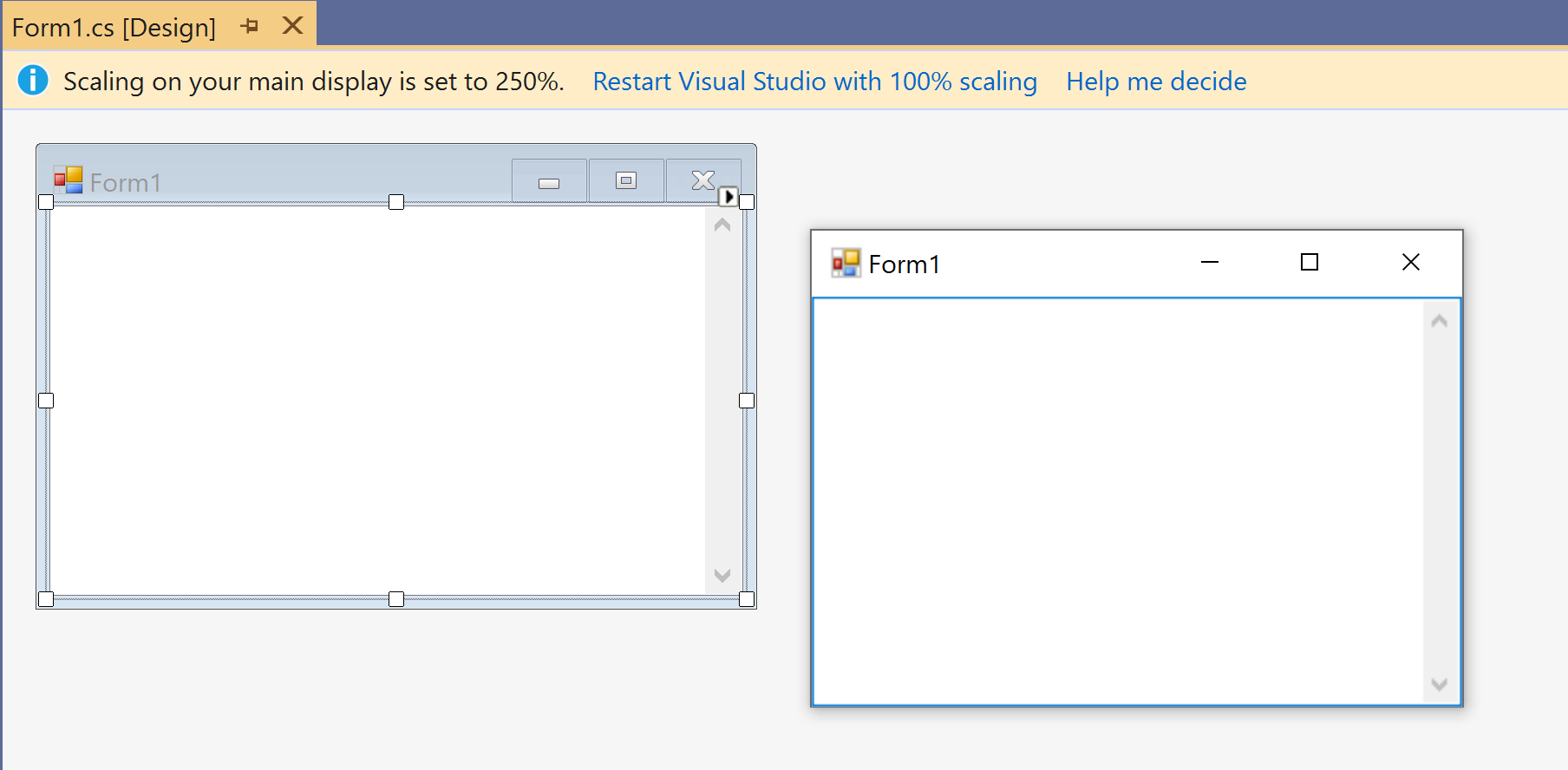
które się pojawiło:

* 1. przełącz okno własności do widoku w kategoriach (przycisk *Categorized *);
  2. w grupie *Behavior* zmień własność Multiline na True (to ustawienie dostępne jest też w liście bocznej tej kontrolki);
  3. w grupie *Layout* odnajdź własność Dock; z własnością tą związane jest proste okno dialogowe, które pozwala na wybór położenia komponentu na formie; wybierz Fill — pole tekstowe wypełni całą dostępną przestrzeń okna;
  4. zmień także widoczną w grupie *Appearance* własność ScrollBars

(paski przewijania) na Both.



Dzięki ustawieniu własności Dock nie musimy się martwić o zachowanie pola tekstowego w razie zmiany rozmiaru formy przez użytkownika — komponent powinien zmieniać swój rozmiar tak, żeby dopasować go do nowego rozmiaru okna. Warto uruchomić aplikację (*F5*) i sprawdzić, jak to działa.



##### Zmiana nazwy okna

Nazwa okna pozwala na identyfikację aplikacji. Warto zatem umieścić na pasku tytułu nazwę naszej aplikacji, tj. Notatnik.NET. Będzie ona również widoczna na pasku zadań.

1. W widoku projektowania zaznacz klasę okna Form1 (np. klikając pasek tytułu formy na podglądzie) i przejdź do listy własności na zakładce *Properties*.
2. Odnajdź własność Text formy i w związanym z nią polu edycyjnym

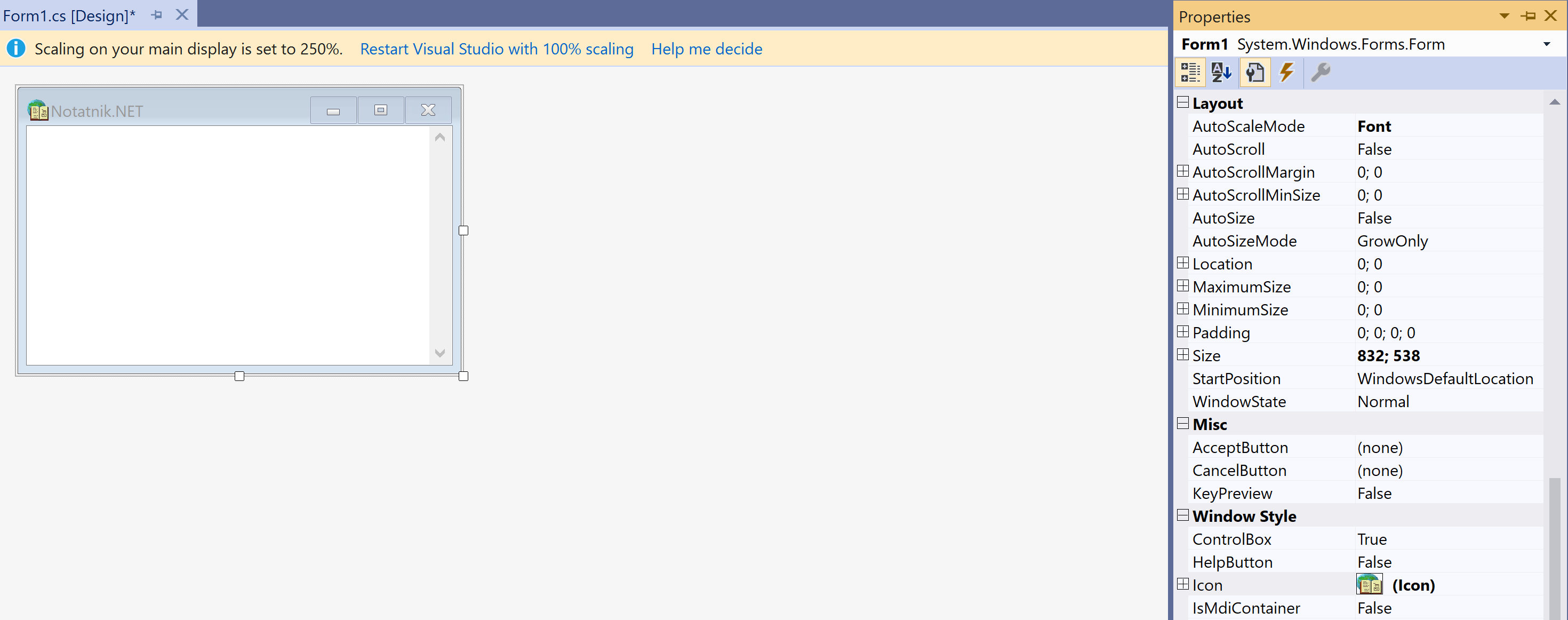
wpisz Notatnik.NET. Naciśnij *Enter*, aby potwierdzić zmianę.

##### 

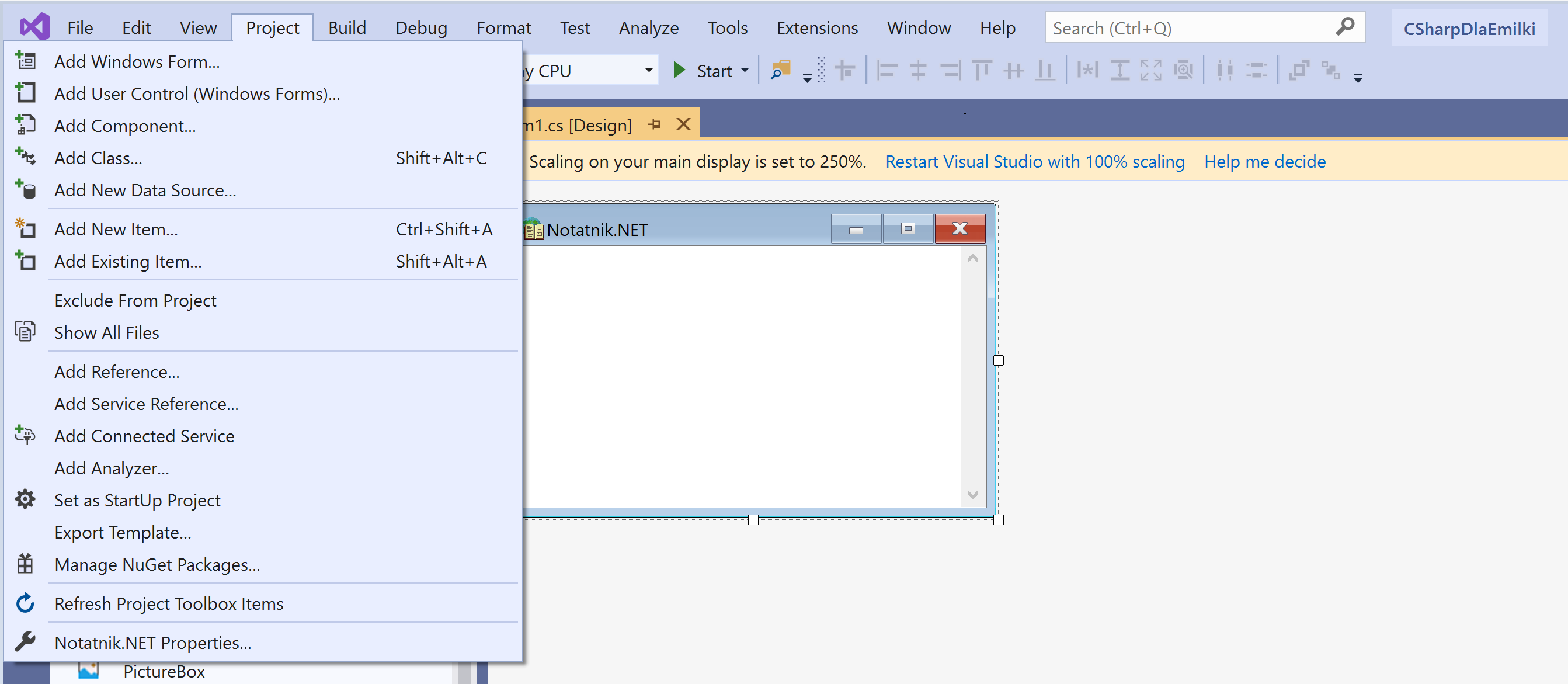
##### Zmiana ikony okna i aplikacji

Ustawmy ikonę formy i projektu. W tym celu:

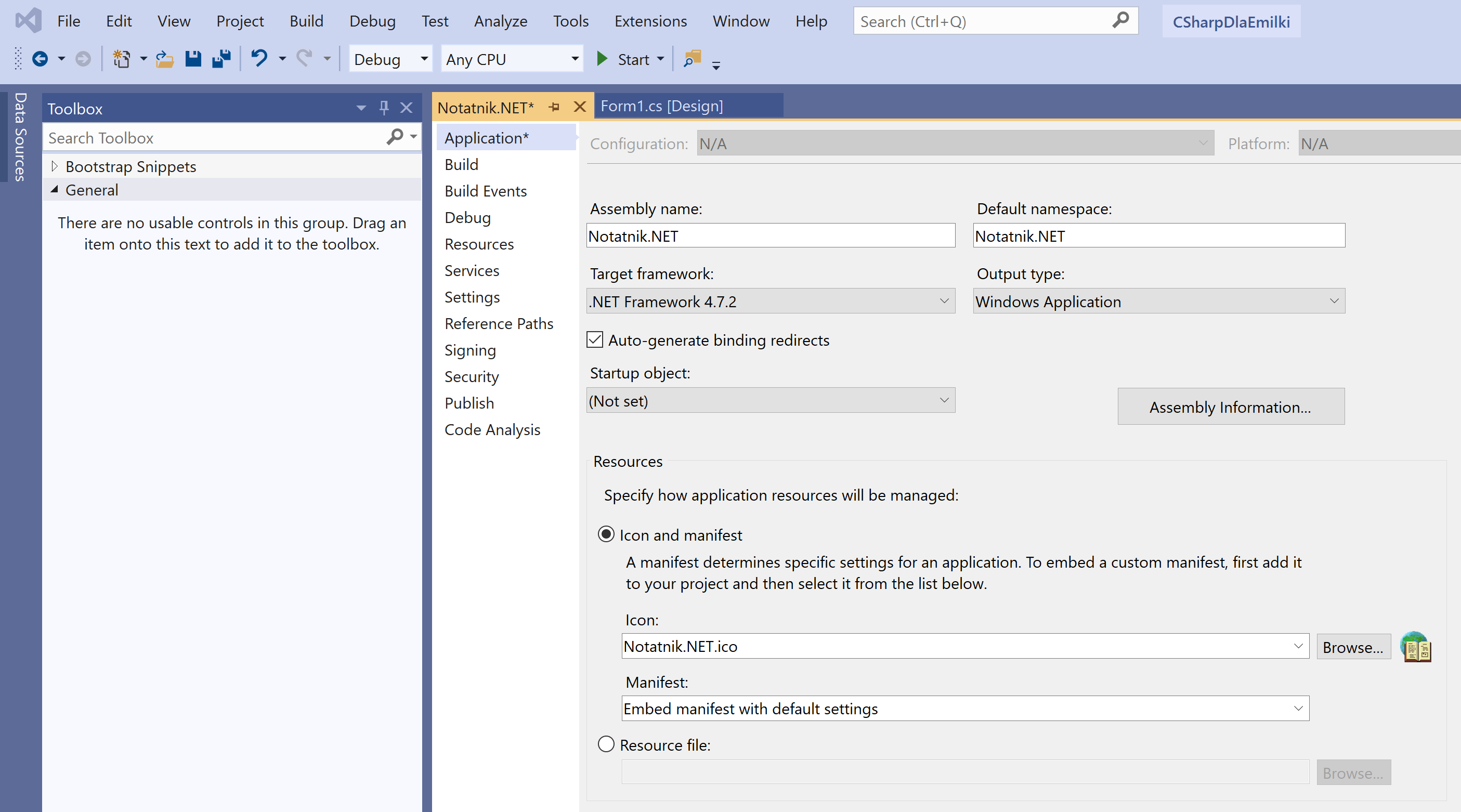
1. Ponownie zaznacz obiekt okna *Form1*, klikając jego pasek tytułu w podglądzie.
2. Odnajdź jego własność Icon w grupie *Window Style* i kliknij przycisk z trzema kropkami.
3. Otworzy się standardowe okno dialogowe, w którym możesz wskazać plik o rozszerzeniu *.ico*. Zostanie z niego pobrana ikona wyświetlana na pasku tytułu formy. Po kliknięciu *Otwórz* wybierz jakiś plik *.ico* i zamknij okno dialogowe.



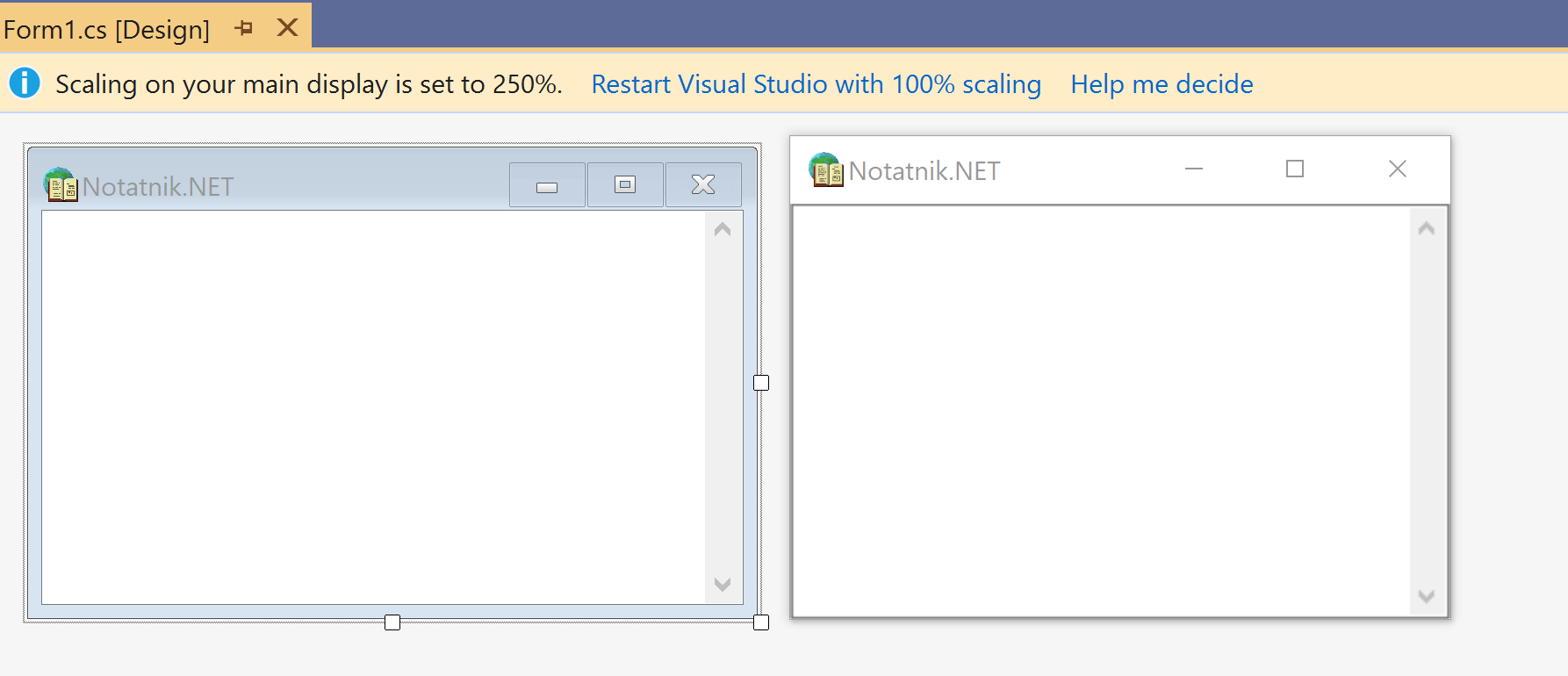
1. Następnie z menu *Project* wybierz pozycję *Notatnik.NET Properties…*.



1. Pojawi się zakładka *Notatnik.NET*, na której kliknij przycisk z trzema kropkami przy polu *Icon* w grupie *Resources*.
2. Ponownie zobaczysz okno dialogowe wyboru pliku. Wybierz w nim plik zawierający ikonę aplikacji1, włączoną do skompilowanego pliku *.exe*. Nic nie stoi na przeszkodzie, a nawet wskazane jest, aby użyć tej samej ikony, którą wcześniej umieściłeś na pasku tytułu. Kliknij *Otwórz*, aby zamknąć okno dialogowe.



1. Teraz zamknij zakładkę z własnościami projektu i naciśnij klawisz *F6*, aby zbudować cały projekt, a następnie *F5*, aby go uruchomić.



W obu przypadkach ikona zostanie umieszczona w zasobach aplikacji, a więc oryginalny plik *.ico* nie będzie więcej potrzebny. Nie trzeba go więc dołączać do rozpowszechnianej aplikacji.

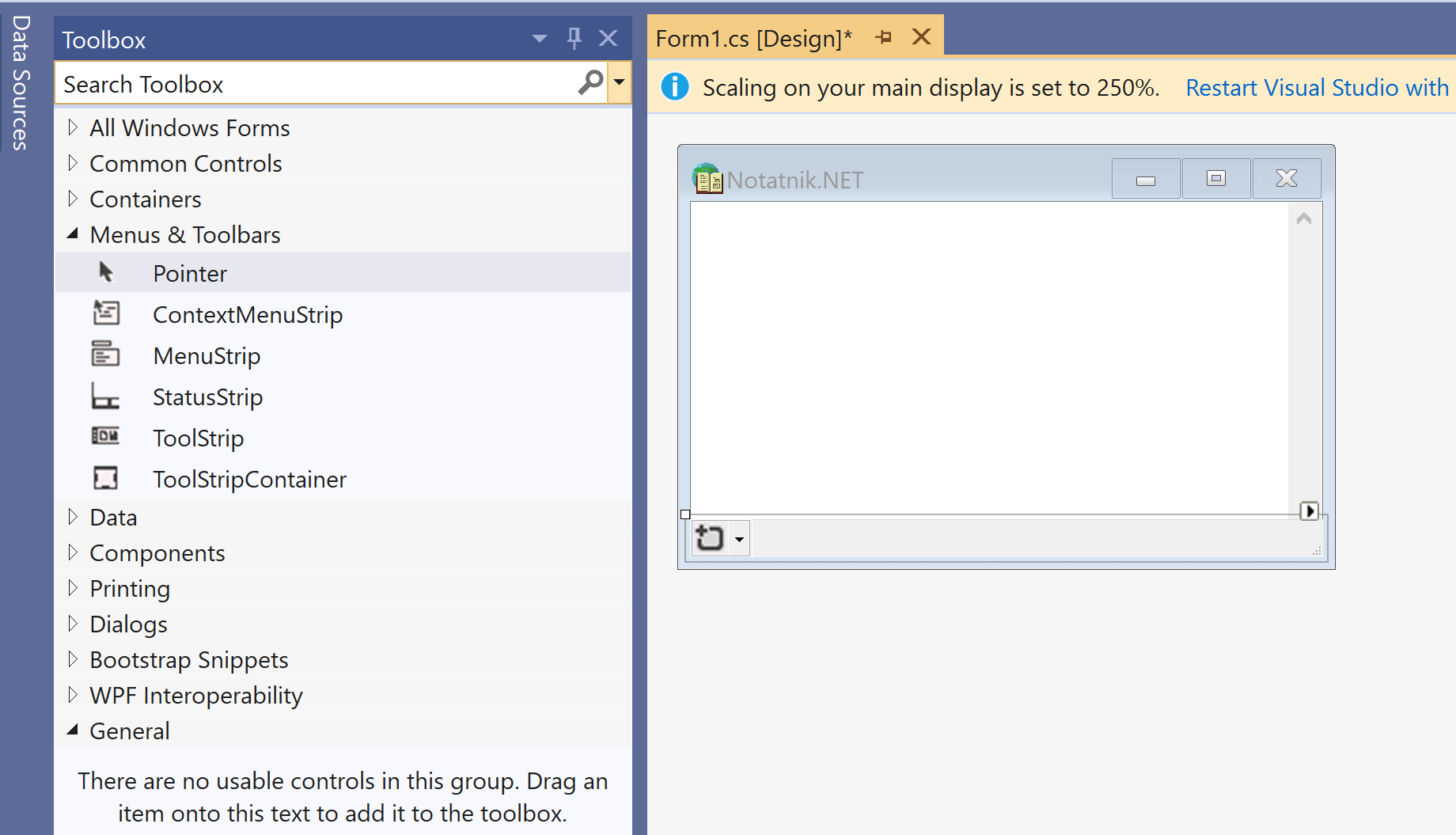
1 Chodzi o ikonę, którą można zobaczyć w Eksploratorze Windows przy przeglądaniu plików.

W ikonie może znajdować się kilka obrazów o różnych rozmiarach i głębi kolorów. Windows sam wybierze najwłaściwszy, najlepiej pasujący do miejsca, w którym ma być wyświetlony. Jeżeli nie znajdzie idealnie dopasowanego, wybierze najbardziej zbliżony i go dostosuje. Przykładowo plik *.ico* z ikoną dla aplikacji i panelu sterowania w nowych wersjach systemu Windows (od wersji Vista) powinien zawierać pełny zestaw obrazów 1616, 3232, 4848, 256256 pikseli, ponieważ użytkownik może dowolnie zmieniać wielkość ikon oglądanych w oknach eksploratora. Ikony paska narzędzi powinny z kolei zawierać obrazy o rozmiarach 1616, 2424 i 3232. Najlepszym rozmiarem dla ikon paska szybkiego uruchamiania jest 4040. Ten rozmiar powinien też znaleźć się w ikonach wykorzystywanych w *dymkach* (ang. *balloon*).

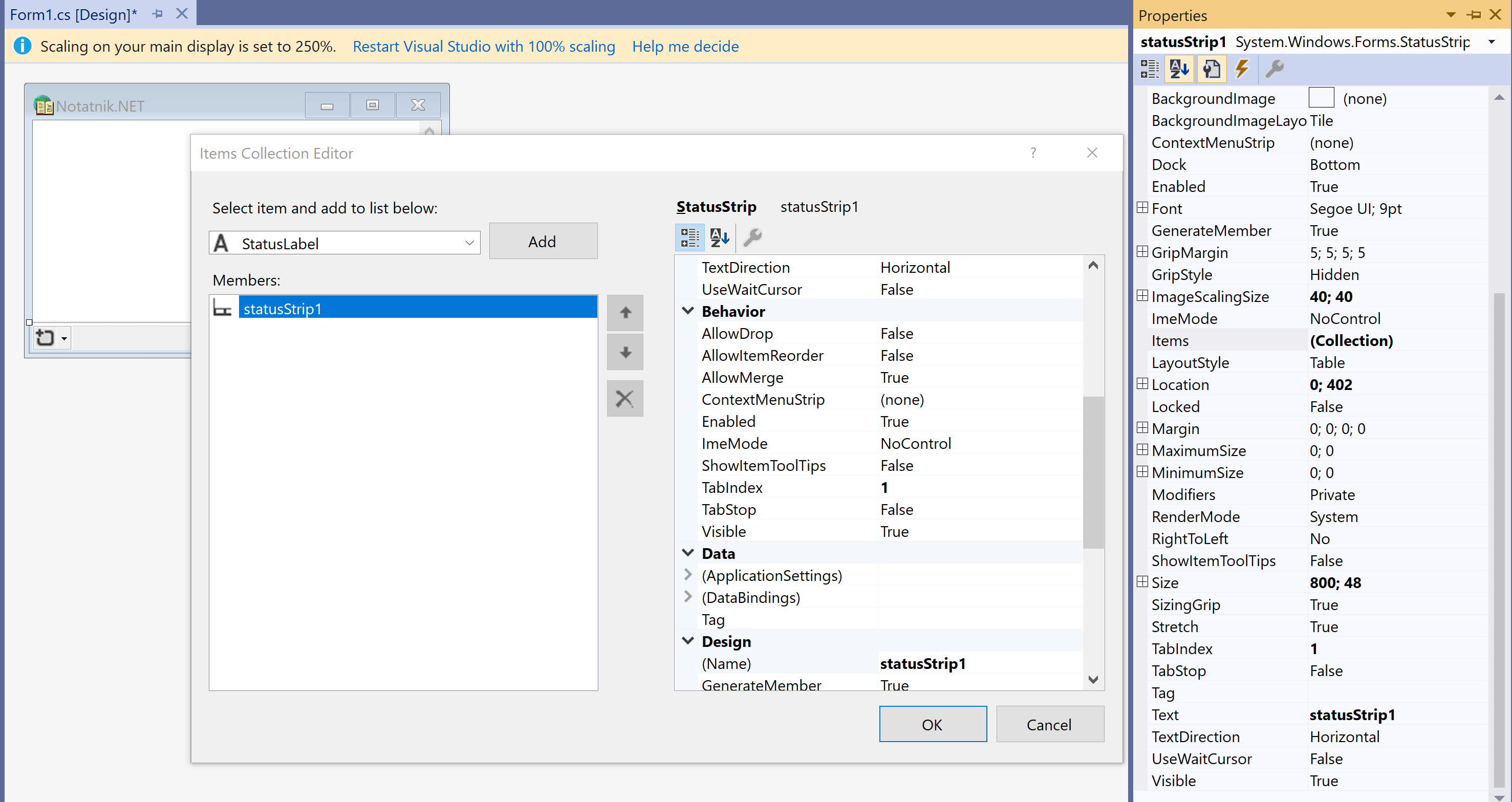
##### Pasek stanu

Biblioteka Windows Forms udostępnia komponent StatusStrip implementujący pasek stanu. Umieśćmy go na formie i przygotujmy do wyświetlania nazwy pliku wczytanego do notatnika. W tym celu:

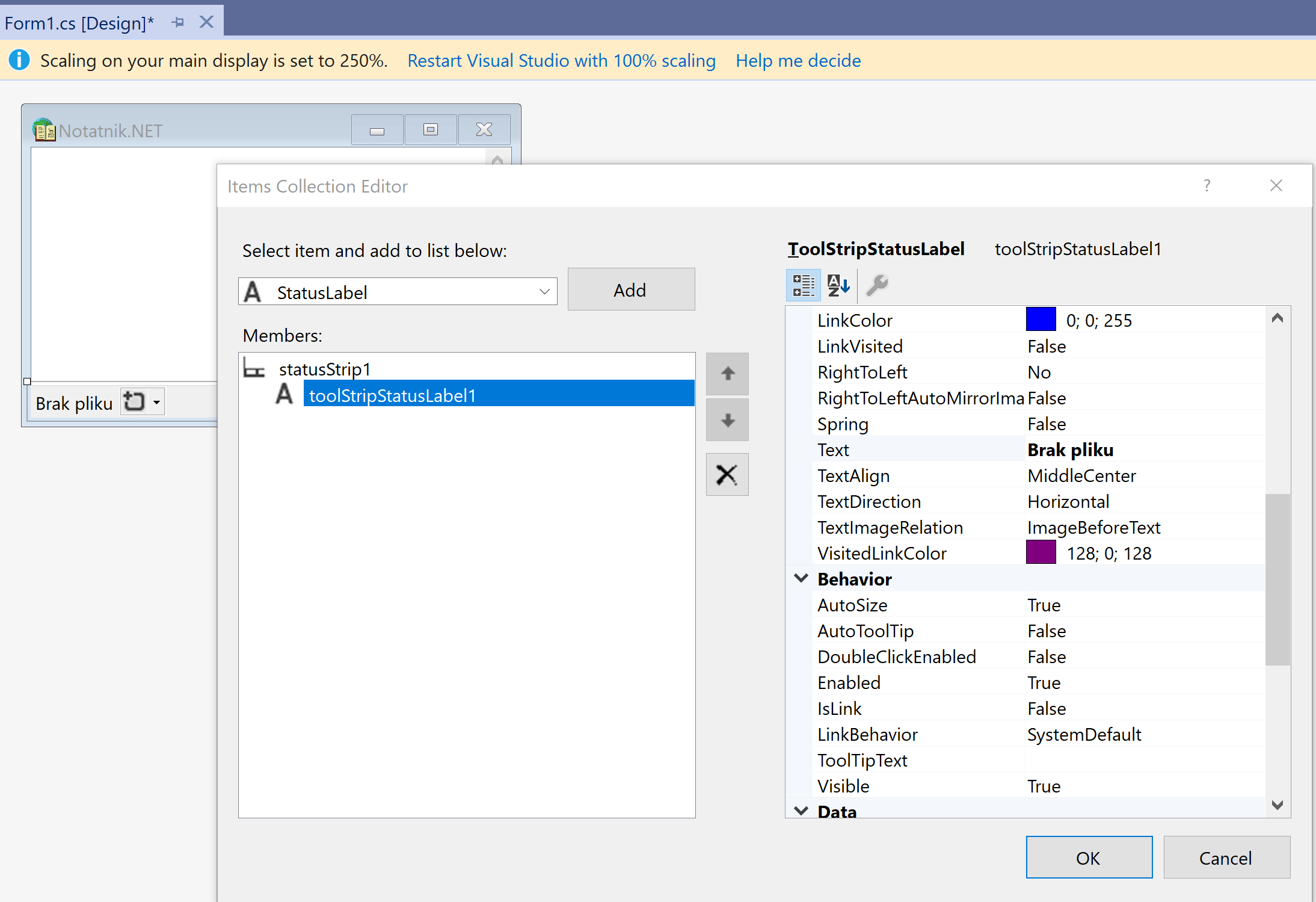
1. Dodaj do okna pasek stanu, czyli komponent StatusStrip z grupy *Menus & Toolbars* (wystarczy kliknąć go dwukrotnie w podoknie *Toolbox*). Jego własność Dock jest automatycznie ustawiona na Bottom, co powoduje, że jest przyklejony do dolnej krawędzi formy.



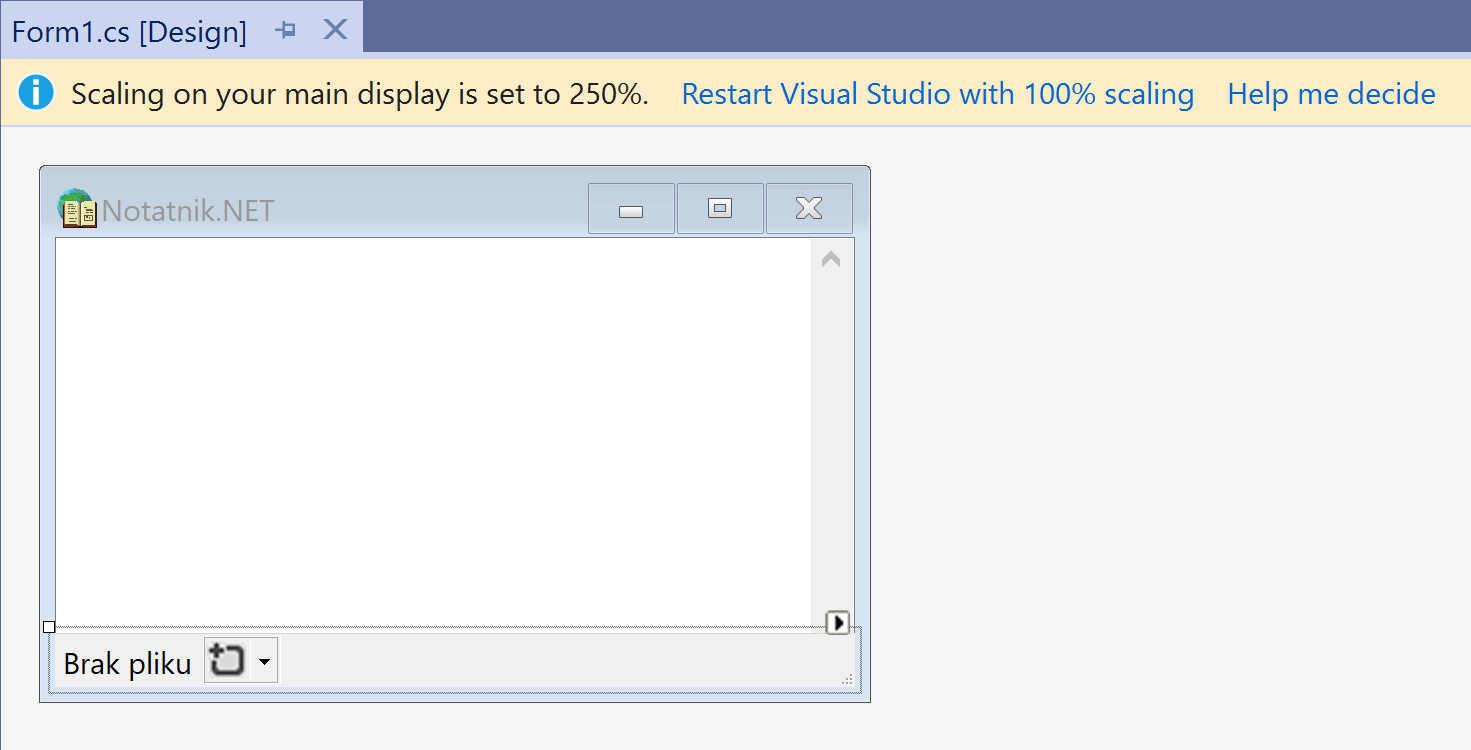
1. Powstanie obiekt statusStrip1. Zaznacz go. W oknie własności odszukaj własność Items i kliknij przycisk z wielokropkiem, aby uruchomić edytor kolekcji.

**

1. W oknie *Items Collection Editor* z rozwijanego menu z lewej strony wybierz *Status Label* (w zasadzie jest on wybrany domyślnie) i kliknij przycisk *Add*. W ten sposób dodasz do paska stanu pole tekstowe.
2. Nie zamykając okna edytora, zaznacz dodany przed chwilą element toolStripStatusLabel1 na lewej liście i za pomocą lokalnego okna własności zmień jego własność Text na Brak pliku.
3. Kliknij przycisk *OK*, aby zamknąć edytor.



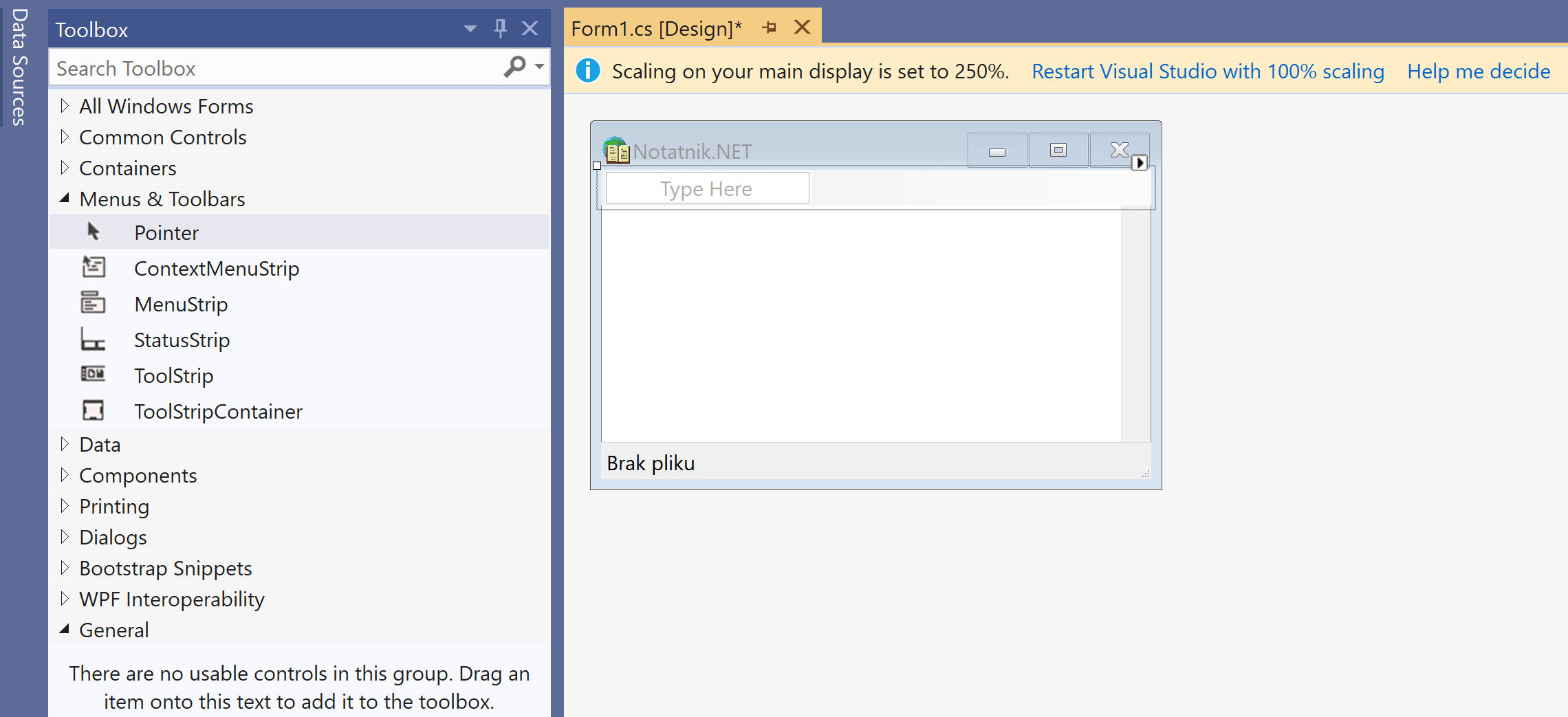
Z powyższego ćwiczenia wynika, że komponent paska stanu jest pojemnikiem dla innnych elementów, np. użytego przez nas ToolStripStatusLabel. Dopiero ten element pozwala na umieszczenie napisów widocznych na pasku stanu.



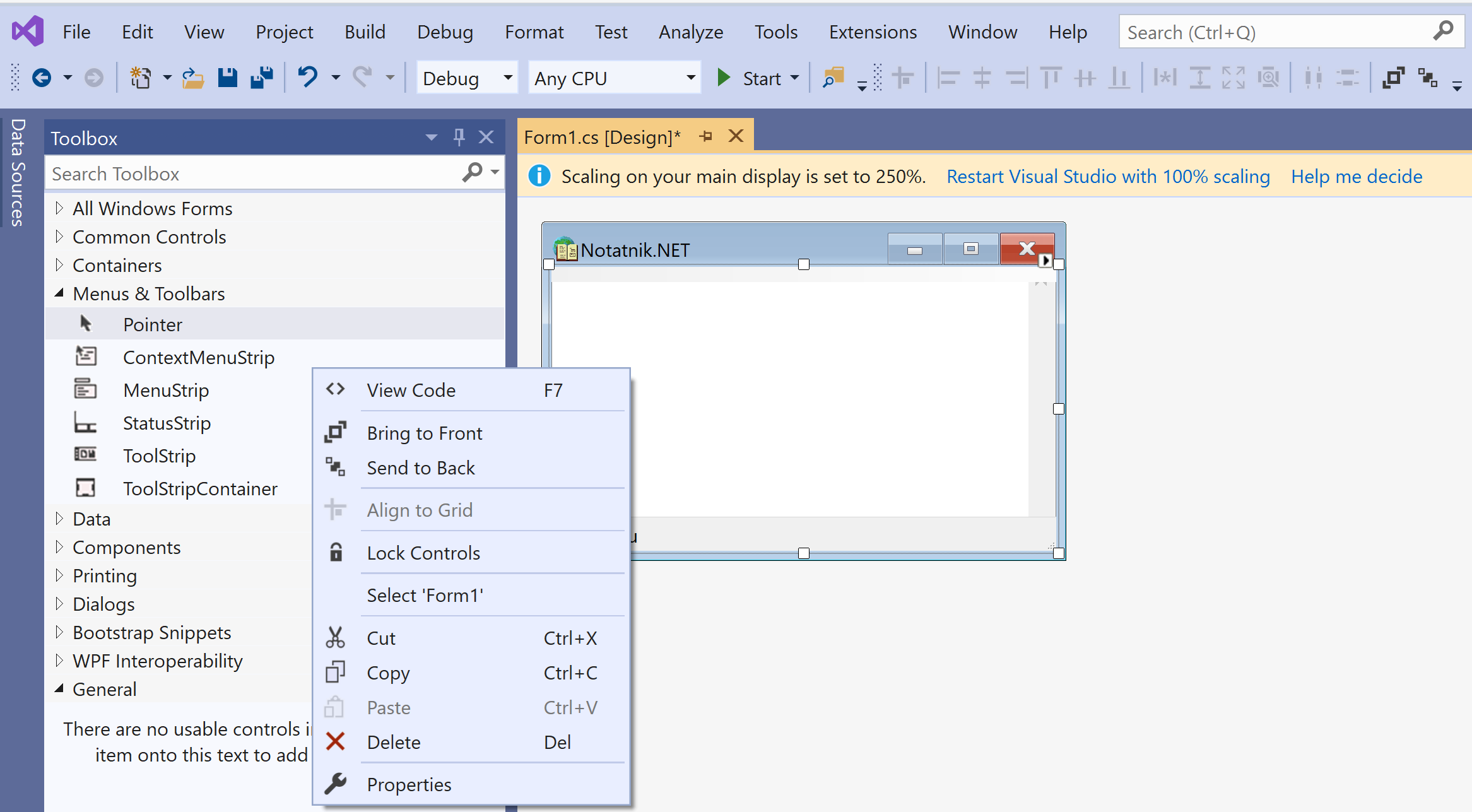
##### Menu główne aplikacji

Kolejnym bardzo typowym elementem, który można dodać do aplikacji, jest menu główne. Umieśćmy je na formie i wyposażmy w pozycje, jakie znajdują się w standardowym notatniku systemu Windows.

1. W widoku projektowania, w podoknie *Toolbox*, w grupie *Menus & Toolbars*, zaznacz komponent MenuStrip i umieść go na formie. Podobnie jak komponent paska stanu, także i ten komponent znajdzie się na dodatkowym pasku pod podglądem formy. Ten komponent ma jednak także reprezentację na podglądzie formy — jest to zarazem edytor menu.



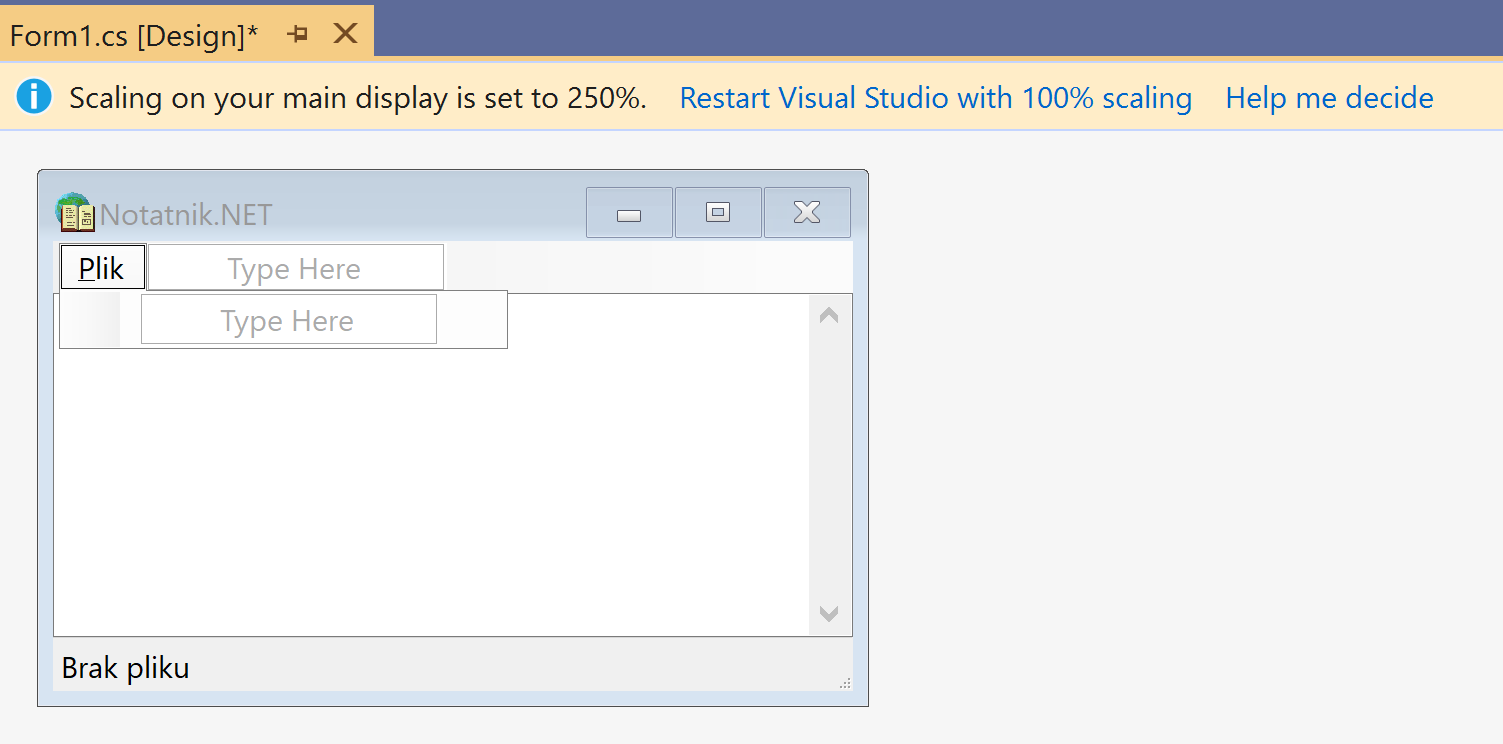
1. Po wstawieniu menu sposób ułożenia pola edycyjnego textBox1 względem okna (własność Dock ustawiona na Fill) może nie być prawidłowy, ponieważ menu ułożone jest niejako nad polem edycyjnym i przesłania jego górną część. W takiej sytuacji zaznacz pole edycyjne i z jego menu podręcznego wybierz polecenie *Bring to Front*.



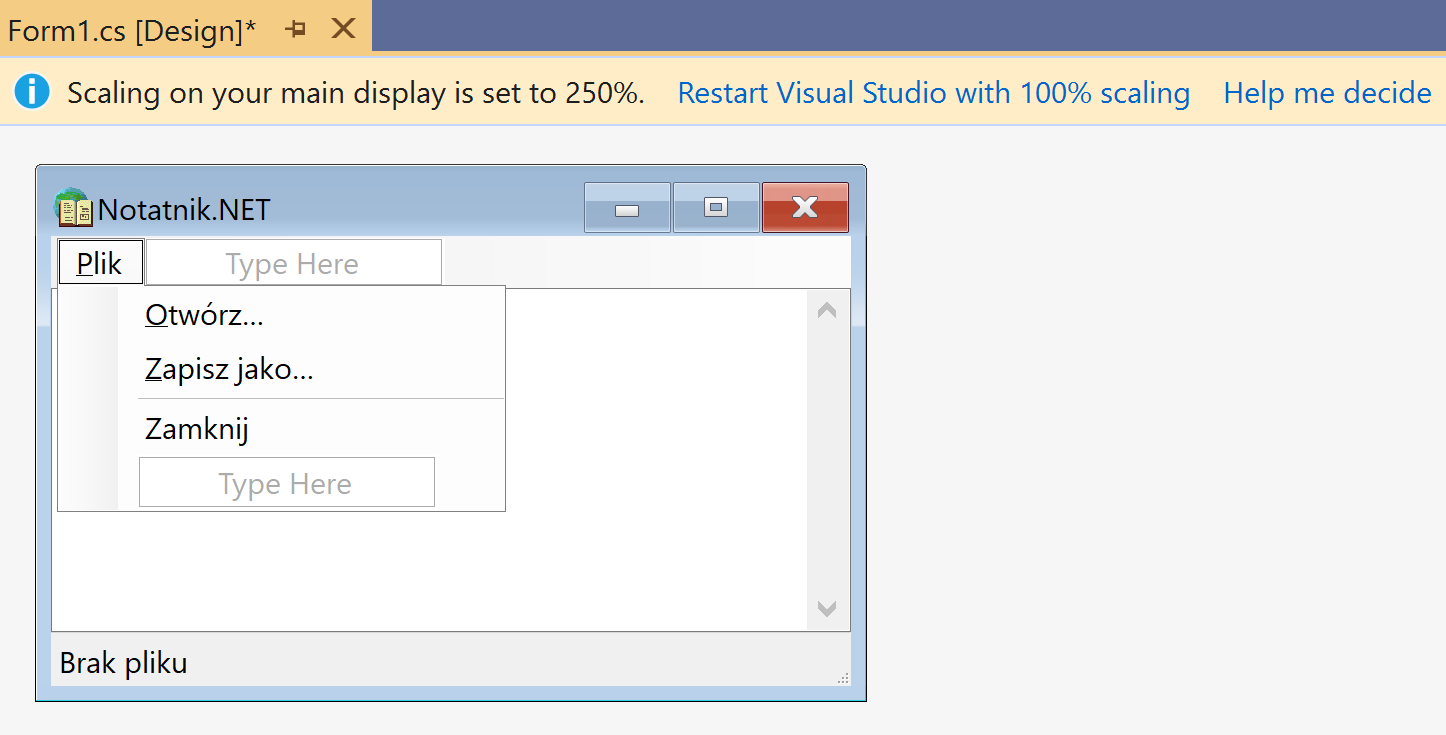
1. Jeżeli komponent menuStrip1 jest zaznaczony, w podglądzie okna staje się widoczny całkiem wygodny edytor menu. Miejsce, w którym możesz wpisać nazwę podmenu, jest oznaczone napisem *Type Here* (z ang. *pisz tutaj*).



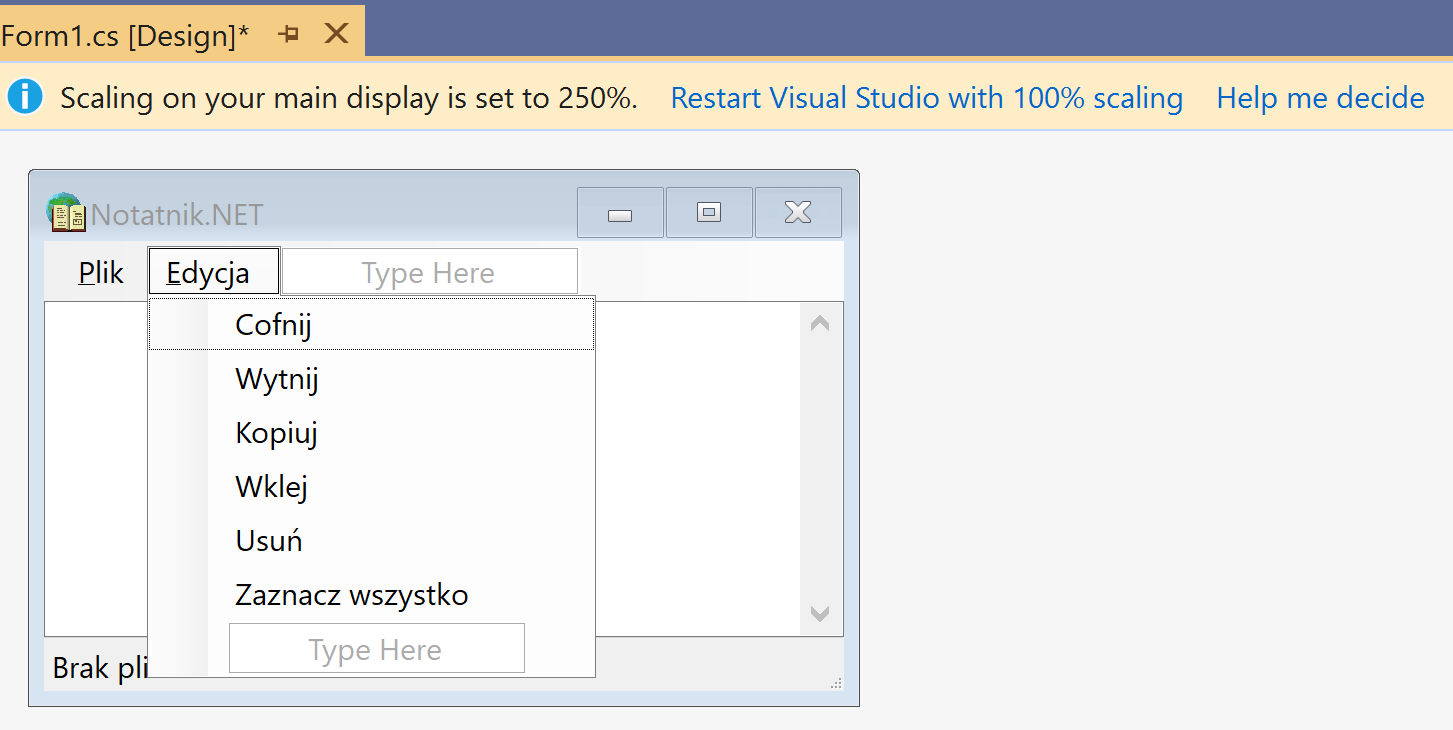
1. Wprowadź nazwę pierwszego podmenu: &Plik. Po rozpoczęciu pisania natychmiast pojawią się dodatkowe miejsca pozwalające na utworzenie kolejnego podmenu oraz pierwszej pozycji w podmenu *Plik*. Oba oznaczone napisami *Type Here*.



1. W podmenu *Plik* umieść cztery pozycje: *&Otwórz…*, *&Zapisz jako…*, separator (należy wpisać pojedynczy znak myślnika) i *Zamknij*.

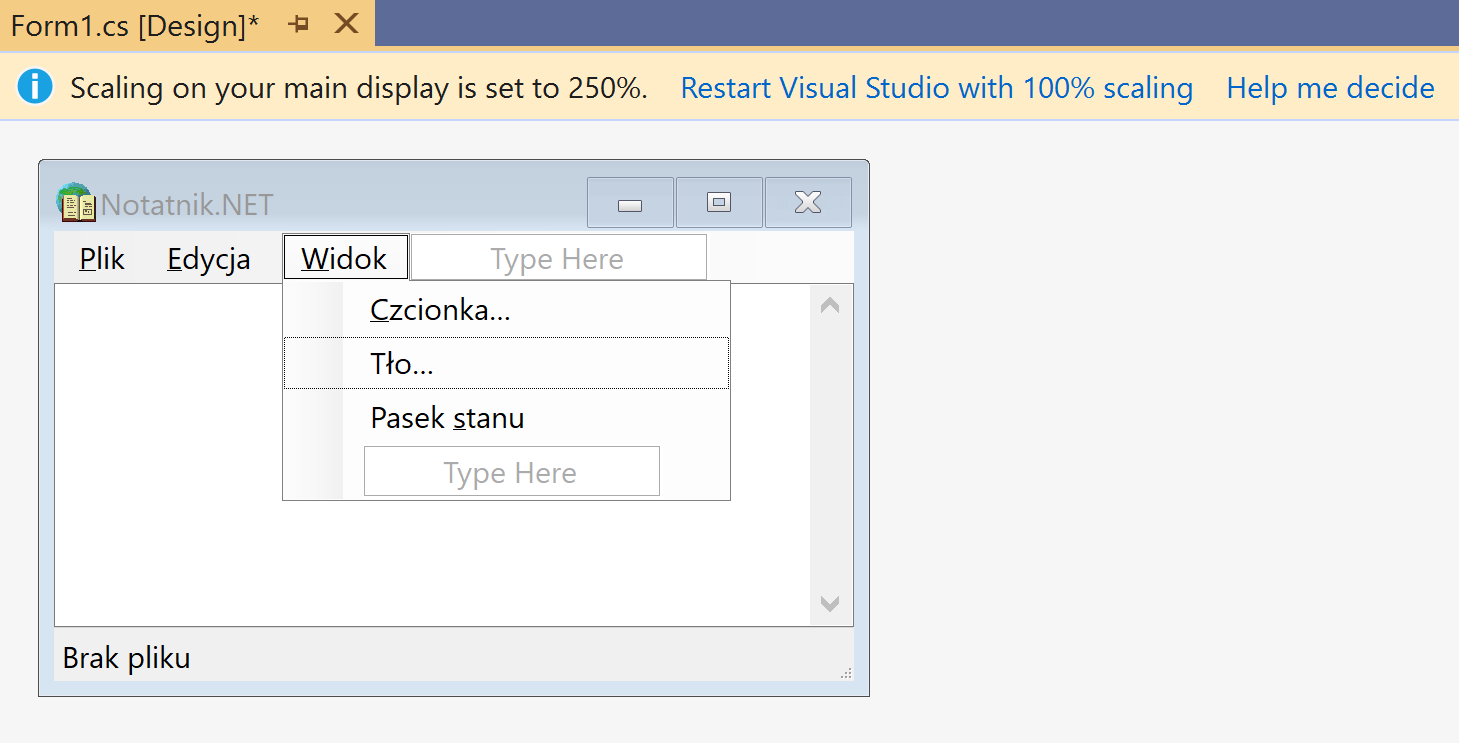


1. Następnie dodaj podmenu *&Edycja* z pozycjami *Cofnij*, *Wytnij*, *Kopiuj*, *Wklej*, *Usuń* oraz *Zaznacz wszystko*. W razie kłopotów z edycją nowego podmenu możesz użyć klawisza *Tab* do przejścia do odpowiedniej pozycji w edytorze menu.

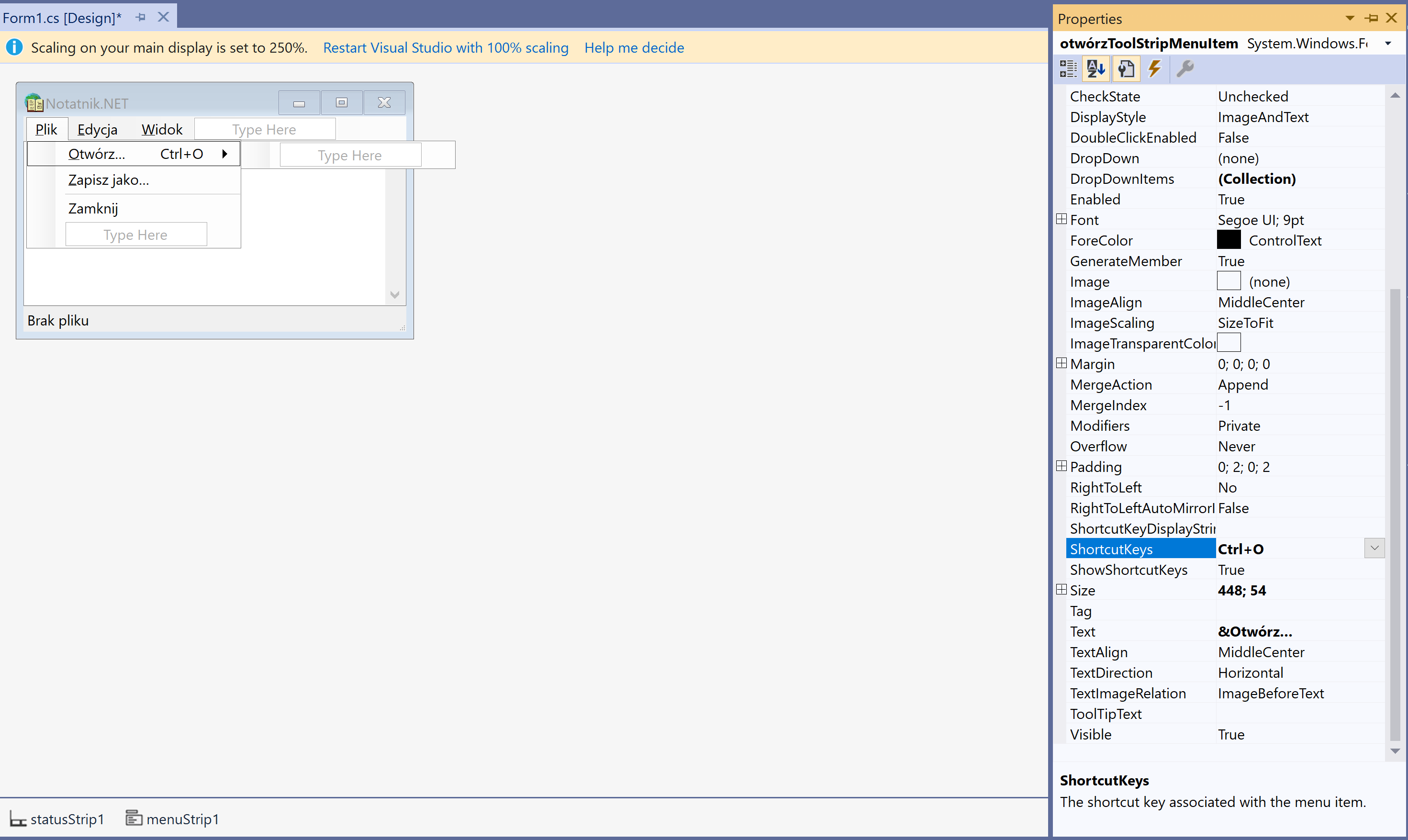


1. Dodaj także podmenu *&Widok*, w którym możesz umieścić pozycje

*&Czcionka…*, *&Tło…* i *Pasek &stanu*.



1. Następnie zaznacz w edytorze pozycję *Otwórz…* i w oknie własności odnajdź ShortcutKeys. Po uruchomieniu edytora tej własności zaznacz w nim opcję *Ctrl*, a z rozwijanego menu wybierz *O* (rysunek 9.5). Analogicznie powiąż z pozycją *Zapisz jako…* kombinację klawiszy *Ctrl+S*.



Umieszczony w nazwie pozycji menu znak *&* wskazuje na aktywny klawisz (oznaczona w ten sposób litera będzie podkreślona po naciśnięciu lewego klawisza *Alt* i uaktywnieniu menu). Klawisze aktywne pozwalają na nawigację w menu bez użycia myszy — wystarczy nacisnąć klawisz odpowiadający podkreślonej literze (*O* w przypadku *Otwórz…*), żeby uruchomić związaną z nią metodę zdarzeniową. To nie koniec ułatwień dla osób nielubiących odrywać dłoni od klawiatury: w punkcie 8. zdefiniowaliśmy dwa klawisze skrótu związane z operacjami na plikach. Naciśnięcie kombinacji *Ctrl+O* spowoduje (oczywiście dopiero po przygotowaniu w następnych ćwiczeniach odpowiednich metod) otwarcie okna dialogowego odpowiedzialnego za wskazanie pliku do wczytania, a *Ctrl+S* — okna dialogowego pozwalającego na wybranie nazwy pliku do zapisu zawartości notatnika.

### Okna dialogowe i pliki tekstowe

Zakończyliśmy przygotowywanie interfejsu. Możemy zatem przejść do programowania poleceń, które umieściliśmy w menu. Zajmiemy się najpierw operacjami na plikach, a więc wczytywaniem tekstu z pliku do edytora i zapisywaniem do pliku ewentualnych zmian.

Zgodnie z filozofią programowania zdarzeniowego skupimy się na obsłudze zdarzeń, jakie może wygenerować użytkownik — w naszym przypadku będzie to reakcja na wybór którejś z trzech pozycji menu: *Otwórz…*, która pozwoli na wczytanie obrazu ze wskazanego przez użytkownika pliku tekstowego, *Zapisz jako…*, która zapisze go do wskazanego pliku, oraz *Zamknij*, której wybranie zamknie naszą aplikację. Ponieważ obsługa ostatniej jest najprostsza, zaczniemy właśnie od niej.

##### Metoda zdarzeniowa zamykająca aplikację

Z pozycją *Zamknij* w menu *Plik* wiążemy metodę zdarzeniową zamykającą okno aplikacji.

1. W widoku projektowania na podglądzie formy kliknij dwukrotnie pozycję menu *Zamknij*.
2. W powstałej w ten sposób metodzie zdarzeniowej (zostaniesz automatycznie przeniesiony do edytora kodu) wpisz polecenie Close(); .

private void zamknijToolStripMenuItem\_Click(object sender,

EventArgs e)

{

Close();

}

Polecenie Close(); umieszczone w metodzie zamknijToolStripMenuItem\_Click jest wywołaniem metody formy, a więc metody instancji klasy Form1. Metoda ta została zdefiniowana w jej klasie bazowej, tj. w System.Windows.Forms.Form. Skutkiem jej działania jest zamknięcie okna aplikacji. Zgodnie z filozofią systemu Windows zamknięcie głównego okna (w tym przypadku jedynego) oznacza także zamknięcie samej aplikacji. W ten sposób nasz cel zostaje osiągnięty w najprostszy z możliwych sposobów. Innym rozwiązaniem byłoby wykorzystanie polecenia Application.Exit();, które informuje bezpośrednio aplikację, że ma zamknąć wszystkie swoje okna, a następnie zakończyć działanie. Klasa Application jest zbiorem metod statycznych, które pozwalają zarządzać działaniem bieżącej aplikacji. Poza Exit są tam metody pozwalające kontrolować działanie wątków aplikacji, pętli obsługi komunikatów, a także metody pozwalające na pobranie informacji o aplikacji zapisanych w pliku *.exe*, ścieżki dostępu do tego pliku itp.

**Potwierdzenie zamknięcia aplikacji**

Zamykanie wszelkiego typu edytorów wiąże się z ryzykiem niezamierzonej utraty niezapisanych zmian w edytowanym dokumencie. Dlatego warto przed zamknięciem aplikacji poprosić użytkownika o potwierdzenie. Pozwala na to zdarzenie FormClosing, które wywoływane jest w przypadku próby zamknięcia formy, ale jeszcze przed zda- rzeniem FormClose, i które w odróżnieniu od tego drugiego pozwala na anulowanie procesu zamykania formy. Zatem przed zakończeniem działania aplikacji wyświetlmy komunikat z pytaniem, czy zapisać tekst lub może anulować zamknięcie.

1. Przejdź do widoku projektowania i utwórz metodę zdarzeniową do zdarzenia FormClosing.
2. Umieść w niej polecenie wyświetlające komunikat z odpowiednim pytaniem. Nie dysponujesz jeszcze metodą zapisującą tekst do pliku, więc w odpowiednim miejscu metody Form1\_FormClosing wstaw tymczasowe polecenie wyświetlające komunikat, który ma przypominać o uzupełnieniu kodu, ewentualnie zgłoszenie wyjątku NotImplementedException.

*Zapisać czy zapomnieć?*

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

DialogResult dr = MessageBox.Show("Czy zapisać zmiany w edytowanym dokumencie?", this.Text, MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question, MessageBoxDefaultButton.Button3);

switch (dr)

{

case DialogResult.Yes:

MessageBox.Show("Wstawić wywołanie metody zapisującej zawartość notatnika do pliku"); break;

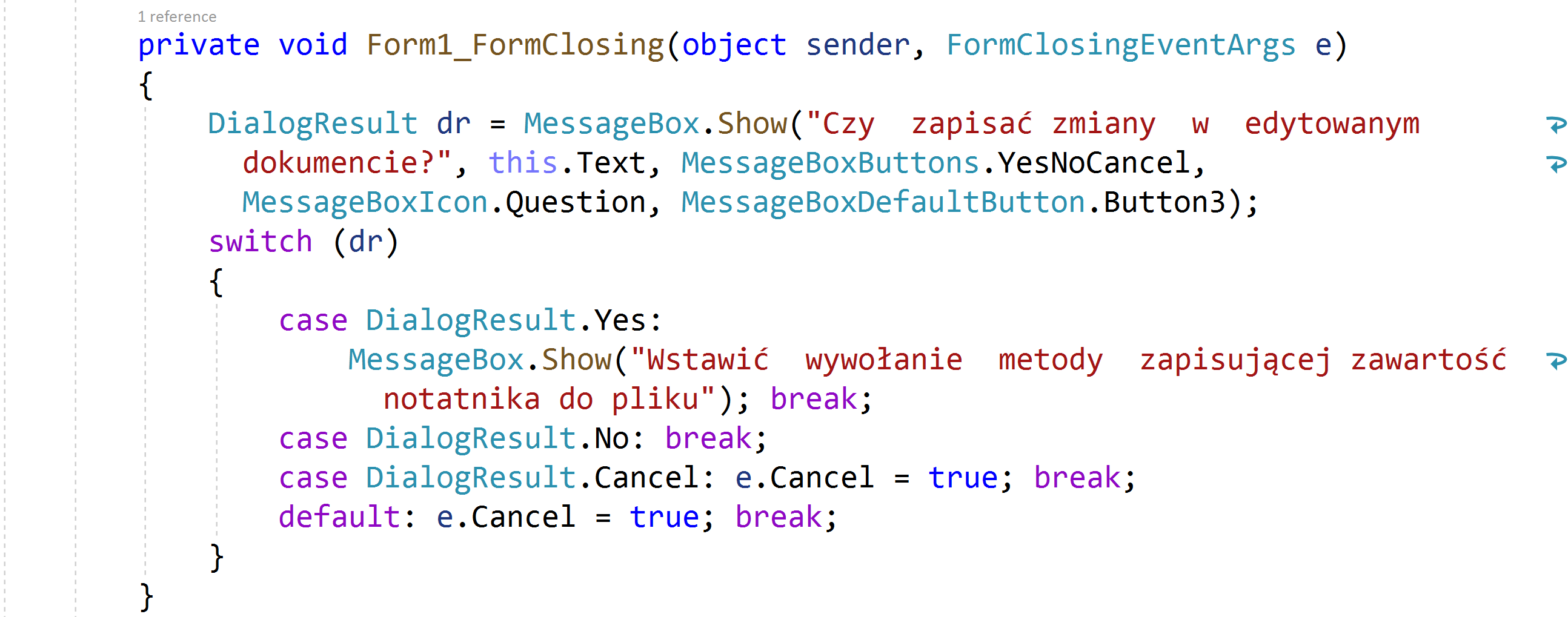
case DialogResult.No: break;

case DialogResult.Cancel: e.Cancel = true; break;

default: e.Cancel = true; break;

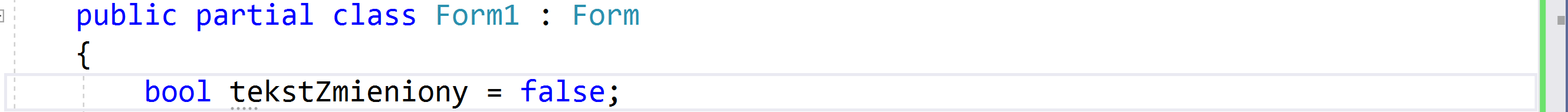
}

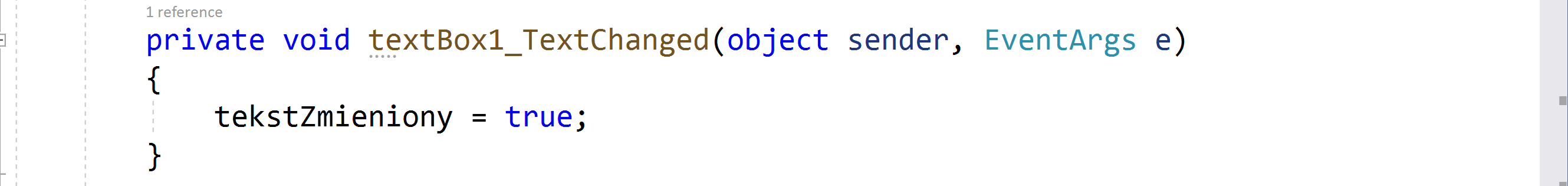
}



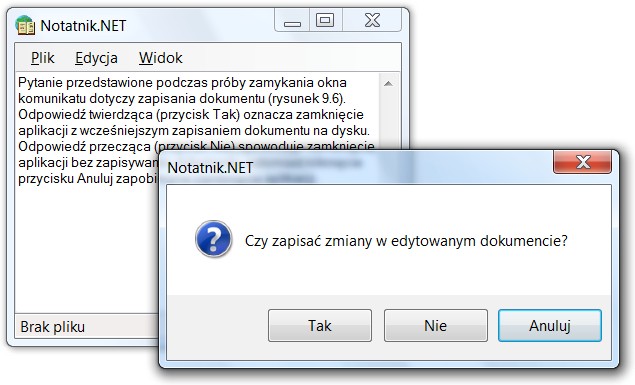
1. Pytanie o zachowanie dokumentu nie ma sensu, jeżeli nie został on zmieniony po otwarciu aplikacji lub po ostatnim zapisie na dysku. Dlatego warto w klasie Form1 zdefiniować pole tekstZmieniony typu bool, którego wartość będzie ustalana na true w przypadku modyfikacji zawartości notatnika (pomocne będzie zdarzenie TextChanged komponentu TextBox). Jeżeli podczas zamykania formy jest ono równe false, komunikat z pytaniem nie zostanie pokazany.

bool tekstZmieniony = false;





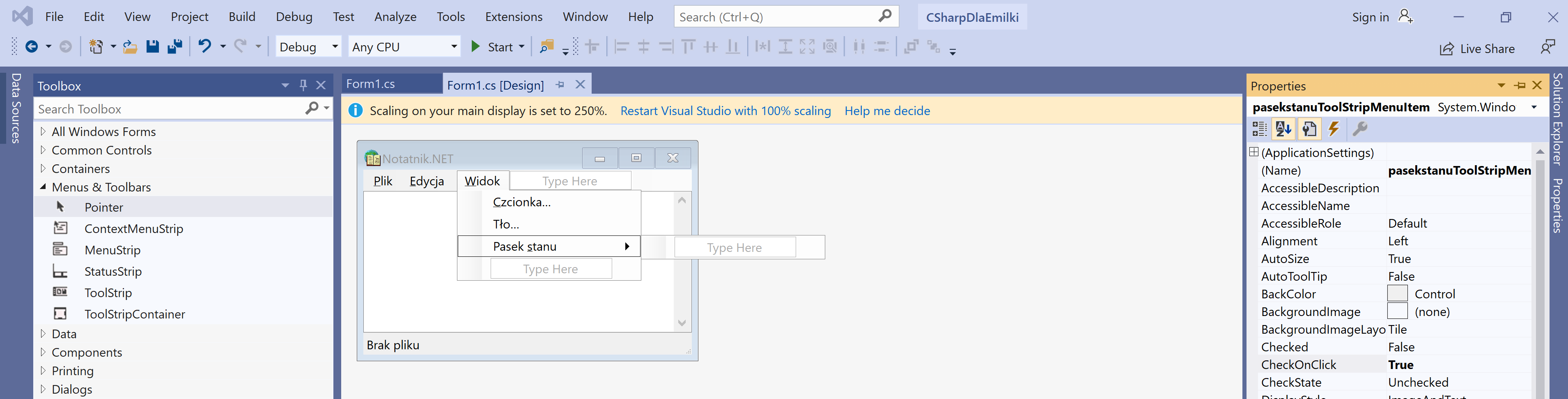
Pytanie przedstawione podczas próby zamykania okna komunikatu dotyczy zapisania dokumentu. Odpowiedź twierdząca (przycisk *Tak*) oznacza zamknięcie aplikacji z uprzednim zapisaniem dokumentu na dysku. Odpowiedź przecząca (przycisk *Nie*) spowoduje zamknięcie aplikacji bez zapisywania dokumentu, natomiast kliknięcie przycisku *Anuluj* zapobiegnie zamknięciu aplikacji.



##### Ukrywanie i pokazywanie paska stanu (polecenie w menu)

Teraz zajmiemy się pozycją *Pasek stanu* z menu *Widok*. Wykorzystamy jej własność Checked w taki sposób, żeby użytkownik mógł zmieniać stan pozycji (oznaczony „ptaszkiem” dodawanym z lewej strony pozycji w menu) i w ten sposób uzyskał możliwość kontrolowania tego, czy pasek stanu na dole formy jest widoczny. Innymi słowy, przygotujemy metodę wiążącą stan własności Visible paska stanu z pozycją *Pasek stanu* w menu *Widok*.

1. Zamknij działającą aplikację i wróć do podglądu formy (klawisz *F7*, jeżeli aktualnie jesteś w edytorze kodu).
2. W podglądzie formy zaznacz pozycję *Pasek stanu* z menu *Widok*. W oknie własności znajdź jej własność CheckOnClick i ustaw ją na True.



1. Następnie kliknij dwukrotnie tę pozycję w edytorze menu. Utworzysz w ten sposób metodę zdarzeniową, w której należy umieścić poniższy kod.

*Zmiana stanu pozycji menu wymaga zmiany wartości jej własności Checked*

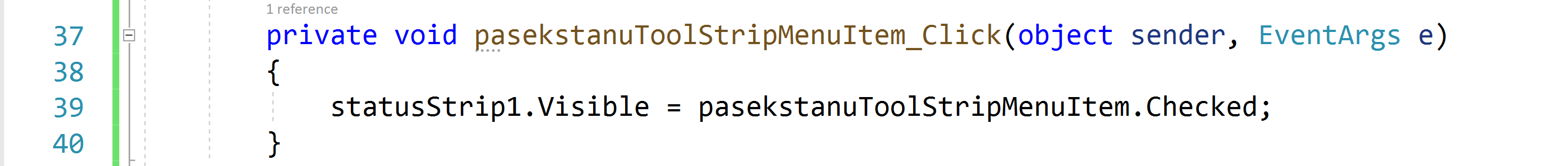
private void pasekstanuToolStripMenuItem\_Click(object sender,

EventArgs e)

{

statusStrip1.Visible = pasekstanuToolStripMenuItem.Checked;

}



Ustawiona na True własność CheckOnClick komponentu ToolStripMenuItem powoduje, że stan zaznaczenia pozycji w menu jest automatycznie przełączany po jej wybraniu. Zamiast tego zmianę zaznaczenia można wykonywać ręcznie w metodzie zdarzeniowej, korzystając z instrukcji postaci pasekstanuToolStripMenuItem.Checked

= !pasekstanuToolStripMenuItem.Checked;.

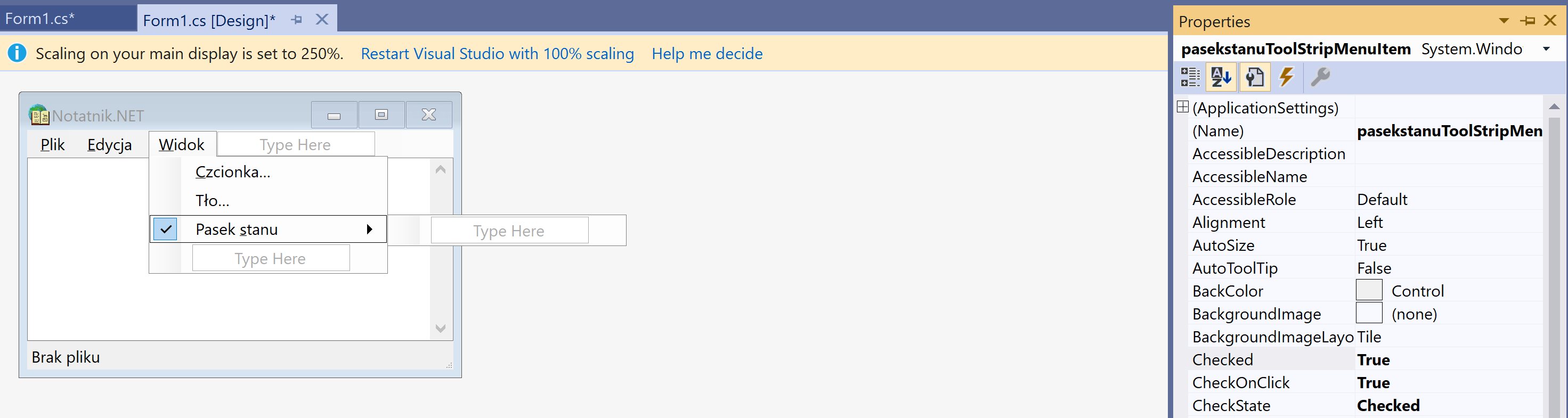
1. Jeszcze raz wróć do podglądu formy i ponownie zaznacz pozycję

*Pasek stanu* w menu *Widok*.

1. Za pomocą okna własności ustal wartość początkową własności

Checked na True (lub równoważnie zmień wartość własności

CheckState na Checked).



Tym razem sposób wykorzystania pozycji menu różni się nieco od poprzedniego przypadku, w którym służyła ona po prostu do

uruchomienia metody. Teraz wykorzystaliśmy własność Checked,

która odpowiada za umieszczenie z lewej strony menu symbolu zaznaczenia. Dzięki niemu pozycja *Pasek stanu* działa jak pole opcji,

które może mieć jeden z dwóch stanów — zaznaczony lub nie. Od tego stanu zależy to, czy w oknie aplikacji widoczny jest pasek stanu.

##### Odczytywanie tekstu z pliku

Użyty w projekcie notatnika komponent TextBox nie posiada żadnych metod pozwalających na łatwe czytanie lub zapisywanie zawartości do plików tekstowych. Musimy zatem sami przygotować odpowiednie metody. Szczególnie potrzebujemy metody wczytującej plik tekstowy do tablicy łańcuchów (każdy łańcuch będzie osobnym akapitem tekstu widocznego w edytorze).

1. Przejdź do edytora kodu (*F7*).
2. Do zbioru deklaracji przestrzeni nazw na początku pliku dodaj:

using System.IO;

1. W klasie Form1 umieść kod metody widocznej poniżej.

*Metoda kopiująca zawartość pliku tekstowego do tablicy łańcuchów*

public static string[] CzytajPlikTekstowy(string nazwaPliku)

{

List<string> tekst = new List<string>();

try

{

using (StreamReader sr = new StreamReader(nazwaPliku))

{

string wiersz;

while ((wiersz = sr.ReadLine()) != null) tekst.Add(wiersz);

}

return tekst.ToArray();

}

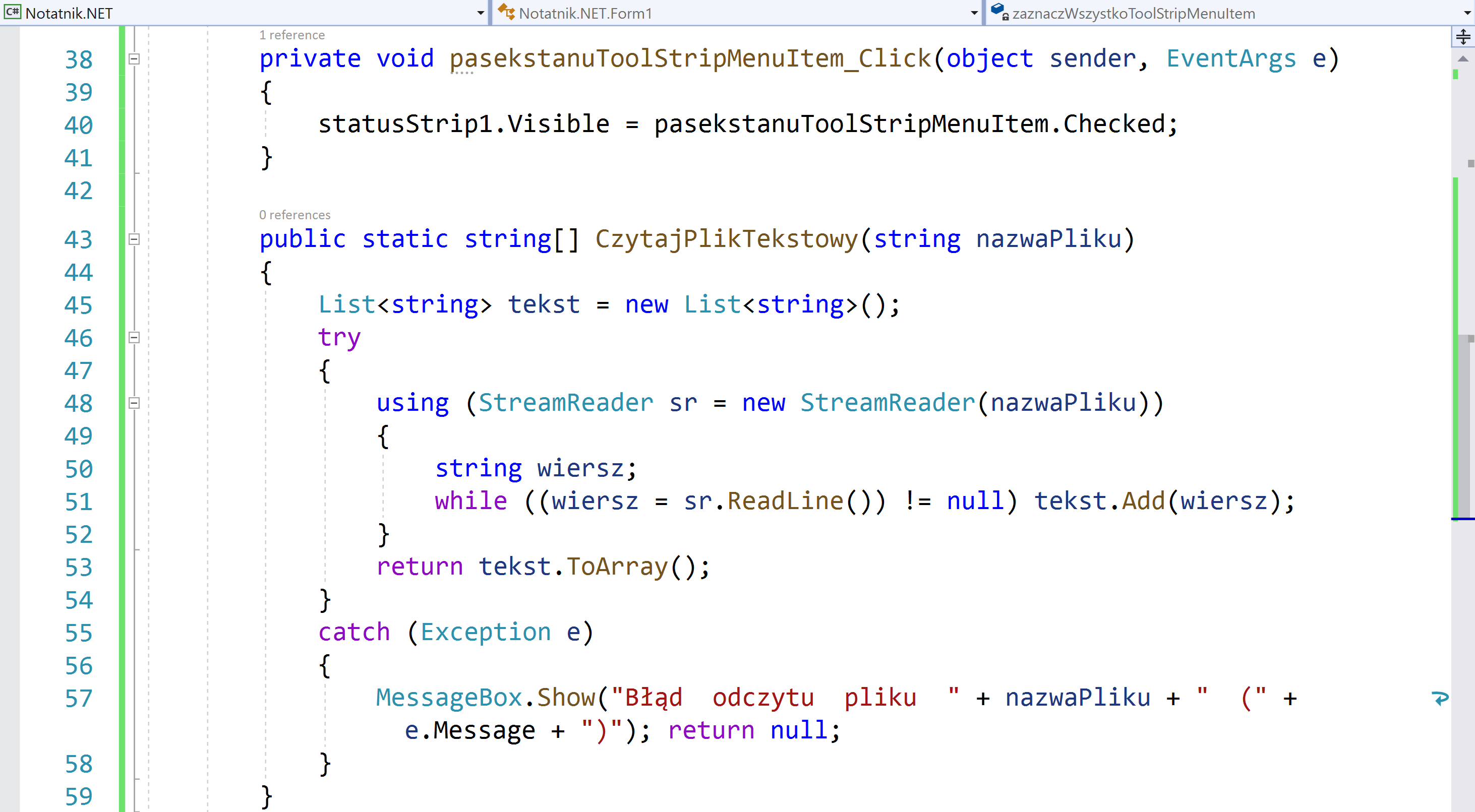
catch (Exception e)

{

MessageBox.Show("Błąd odczytu pliku " + nazwaPliku + " (" + e.Message + ")"); return null;

}

}



Do gromadzenia i przechowywania tekstu wewnątrz metody korzystamy z typu ogólnego (ang. *generics*) List sparametryzowanego typem string. Każdy łańcuch przechowuje osobny wiersz (akapit) odczytanego z pliku tekstu. W przypadku powodzenia (tj. braku wyjątków) metoda konwertuje ową listę na tablicę łańcuchów. Dzięki temu wartość zwracaną przez metodę będzie można wprost przypisać do własności Lines komponentu TextBox, która jest właśnie tablicą łańcuchów. Jeżeli jednak nie intere- sowałby nas podział na akapity (wiersze) wczytywanego tekstu, moglibyśmy go jednym poleceniem odczytać w całości, korzystając z metody ReadToEnd klasy StreamReader.

Należy zwrócić uwagę na użycie słowa kluczowego using w kodzie powyżej. Konstrukcja, w której po słowie using w nawiasach znajduje się polecenie tworzące instancję klasy, zapewnia, że utworzony obiekt zostanie usunięty z pamięci po opuszczeniu zakresu wyznaczonego przez nawiasy klamrowe (operator zakresu), pomimo że jest typu re- ferencyjnego.

Warto także zwrócić uwagę na konstrukcję try..catch. Ograniczyliśmy się w niej tylko do jednej sekcji catch wyłapującej wszystkie wyjątki, wszystkie bowiem dziedziczą po klasie Exception. Moglibyśmy przechwycone wyjątki filtrować, umieszczając po sekcji try kolejne sekcje catch z np. wyjątkami FileNotFoundException, Unauthorized

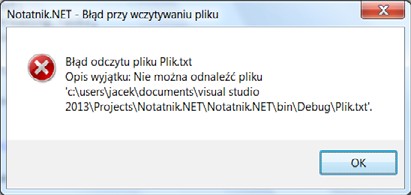
AccessException itd. Pamiętać tylko należy, aby wyjątki najbardziej szczegółowe (najniższe w hierarchii dziedziczenia) umieszczać przed wyjątkami bardziej ogólnymi. Zauważmy także, że linia wyświetlająca nazwę pliku na pasku stanu została włączona do sekcji try. Unikamy w ten sposób sytuacji, w której byłaby ona wykonywana, gdy wcześniej zgłoszony został wyjątek. Należy bowiem pamiętać, że po zgłoszeniu wyjątku i wykonaniu poleceń z sekcji catch wątek nie wraca już do sekcji try, a wykonuje polecenia znajdujące się za sekcją catch, czyli w naszym przypadku metoda button1\_ Click kończy działanie.

Przygotowując tekst informacji wyświetlanej za pomocą metody MessageBox.Show, można korzystać ze znanych z C oraz C++ znaków formatujących, np. \n wstawia znak końca linii, a \t — znak tabulatora. Okno komunikatu może zawierać również tytuł oraz ikonę informującą o charakterze komunikatu (o tym, czy jest to błąd, ostrzeżenie, informacja). Aby tak było, należy podać dodatkowe argumenty metody MessageBox.Show, np.:

MessageBox.Show("Błąd odczytu pliku " + nazwaPliku + "\nOpis wyjątku: " + e.Message, "Notatnik.NET - Błąd przy wczytywaniu pliku", MessageBoxButtons.OK,

MessageBoxIcon.Error);

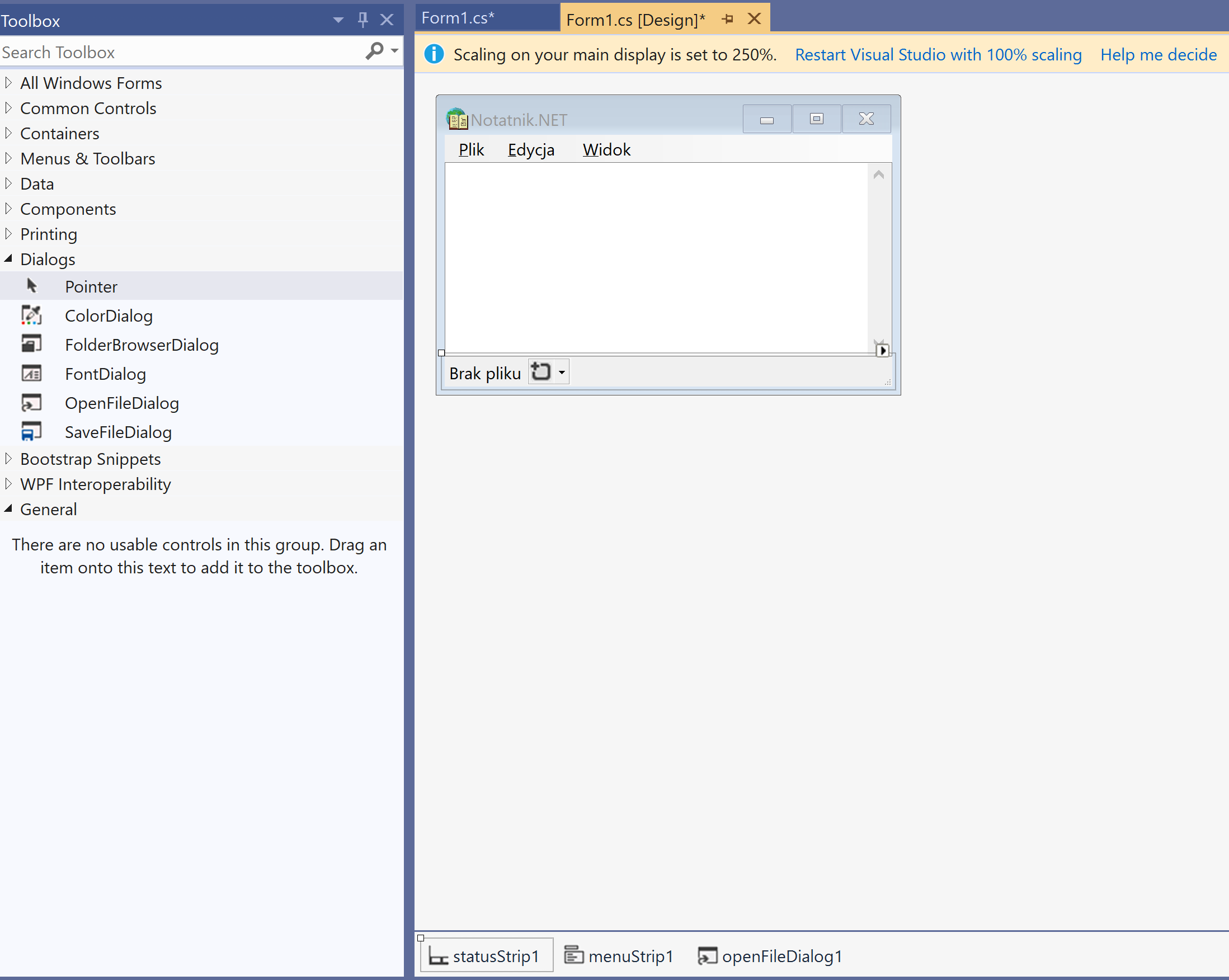
Dodatkowe argumenty to kolejno: tytuł okna, rodzaj umieszczonych na oknie przycisków i wygląd ikony widocznej w oknie wiadomości. Po tych zmianach okno powinno wyglądać tak jak poniżej.



##### Wybór pliku za pomocą okna dialogowego

Przygotujemy teraz metodę pozwalającą na wybór pliku za pomocą standardowego okna dialogowego, a następnie przy użyciu przygotowanej w poprzednim ćwiczeniu metody pomocniczej wczytującą zawartość wybranego pliku do pola tekstowego.

1. Wróć do widoku projektowania, przełączając zakładkę edytora na *Form1.cs [Design]* lub naciskając klawisz *F7*.
2. W podoknie *Toolbox*, w grupie *Dialogs*, zaznacz komponent *OpenFileDialogs* i umieść go na podglądzie formy. Nowy obiekt openFileDialog1 nie pojawi się jednak na samej formie, ale pod nią, na liście komponentów „niewizualnych”, razem z menuStrip1 i statusStrip1.



1. Dwukrotnie kliknij pozycję menu *Otwórz…* w podglądzie menu, tworząc w ten sposób metodę zdarzeniową uruchamianą w przypadku wybrania tej pozycji menu w działającej aplikacji.
2. Do nowej metody wpisz kod poniżej.

*Metoda związana z poleceniem Otwórz… w menu Plik*

private void otwórzToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

string nazwaPliku = openFileDialog1.FileName;

textBox1.Lines = CzytajPlikTekstowy(nazwaPliku);

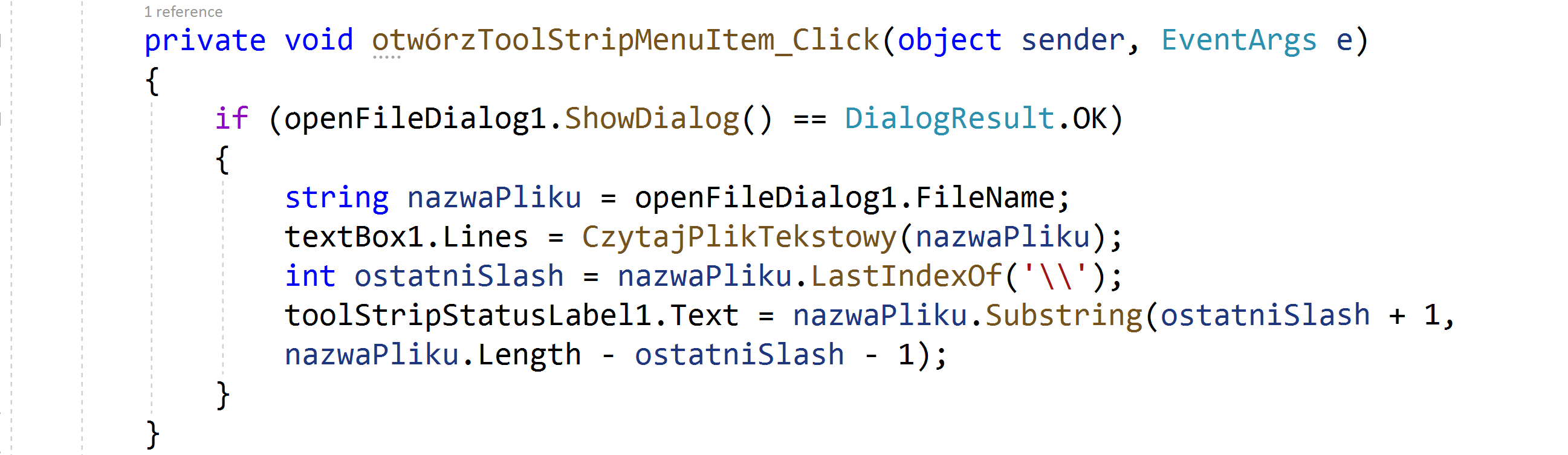
int ostatniSlash = nazwaPliku.LastIndexOf('\\');

toolStripStatusLabel1.Text = nazwaPliku.Substring(ostatniSlash + 1,

nazwaPliku.Length - ostatniSlash - 1);

}

}

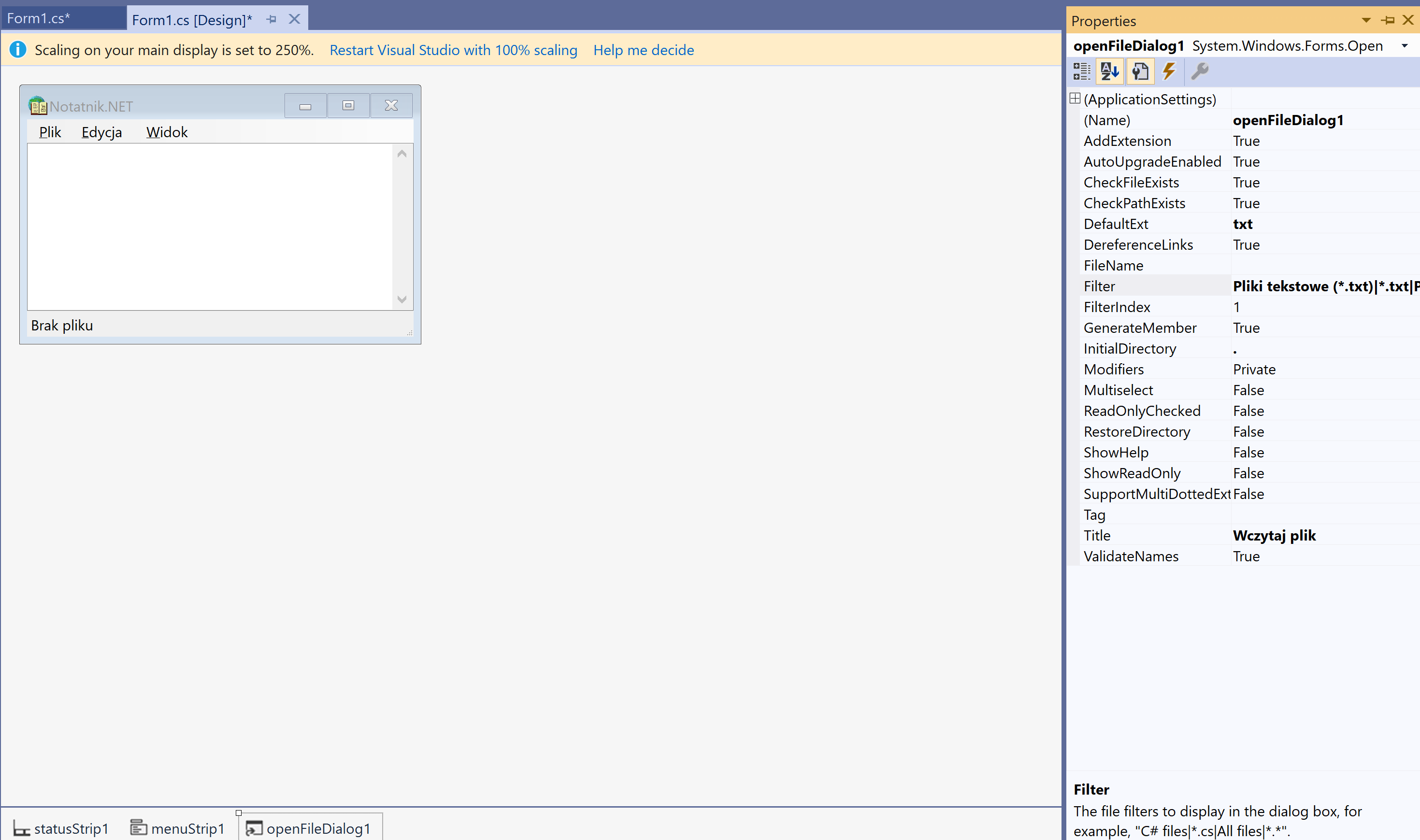


1. Ustaw także kilka opcji okna dialogowego. Aby to zrobić, wróć do widoku projektowania, zaznacz komponent okna dialogowego openFileDialog1 i korzystając z okna własności:
2. zmień tytuł okna, ustawiając własność Title np. na *Wczytaj plik*;
3. własność DefaultExt zmień na *txt* (jest to rozszerzenie dopisywane

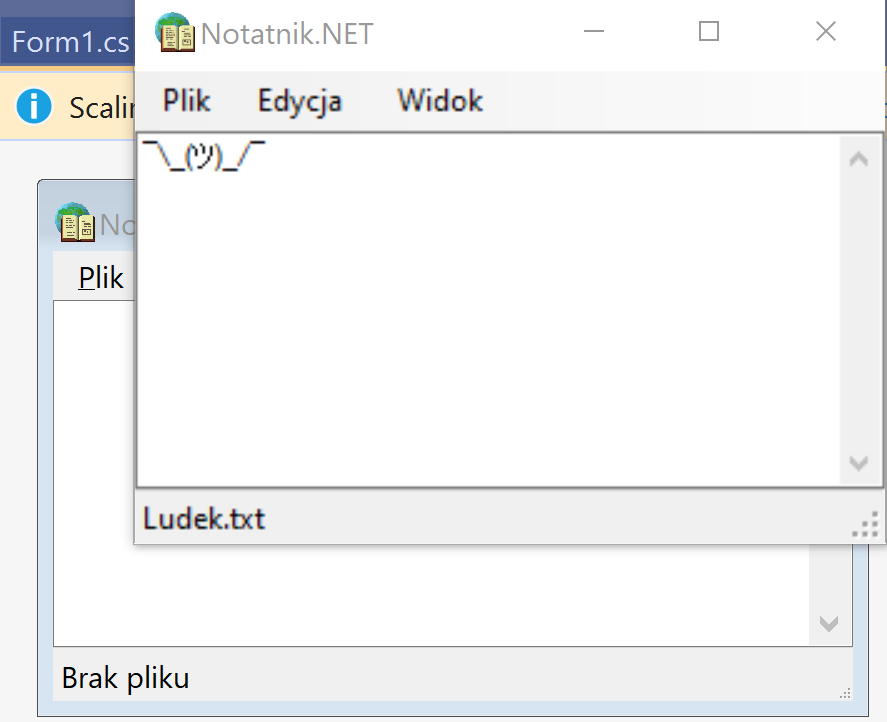
do nazwy pliku, jeżeli użytkownik żadnego rozszerzenia nie

doda sam);

1. wyczyść zawartość pola przy własności FileName;
2. edytuj pole przy własności InitialDirectory, podając katalog domyślny równy katalogowi bieżącemu, tj. po prostu kropkę;
3. wpisz łańcuch określający filtry (własność Filter), np.: Pliki tekstowe (\*.txt)|\*.txt|Pliki INI (\*.ini)|\*.ini|Pliki źródłowe (\*.cs)|\*.cs| Wszystkie pliki (\*.\*)|\*.\*. Uważaj, aby przy znakach | nie wstawić spacji.



Po wprowadzeniu powyższych zmian i kompilacji projektu warto sprawdzić, czy uda nam się wczytać jakiś plik tekstowy, choćby plik *Form1.cs* z kodem źródłowym C#.



W Visual Studio nie ma dobrych edytorów pozwalających na wygodne tworzenie filtrów okien dialogowych (punkt 5e). Do dyspozycji mamy tylko zwykłe pole edycyjne. Własność Filter jest bowiem zwykłym łańcuchem, czyli zmienną typu string.

Fragment tego łańcucha definiujący jeden filtr składa się z dwóch segmentów oddzielonych znakiem |. W pierwszym określamy opis wyświetlany w oknie dialogowym, a w drugim — maskę filtru. Trochę zamieszania powoduje to, że kolejne filtry również oddzielane są znakiem |, przez co cały łańcuch traci na czytelności. Tym bardziej dotkliwy staje się brak wygodnego edytora tej własności.

##### Zapisywanie tekstu do pliku

Przygotujmy teraz metodę pomocniczą ZapiszDoPlikuTekstowego zapisującą wskazaną w argumencie tablicę łańcuchów do pliku tekstowego. Następnie w metodzie zdarzeniowej związanej z pozycją *Zapisz jako...* z menu *Plik* wykorzystamy tę metodę do zapisania do pliku zawartości komponentu TextBox.

1. W obrębie klasy Form1 zdefiniuj metodę jak poniżej.

*Metoda zapisująca zawartość tabeli łańcuchów do pliku tekstowego*

public static void ZapiszDoPlikuTekstowego(string nazwaPliku, string[]

tekst)

{

using (StreamWriter sw = new StreamWriter(nazwaPliku))

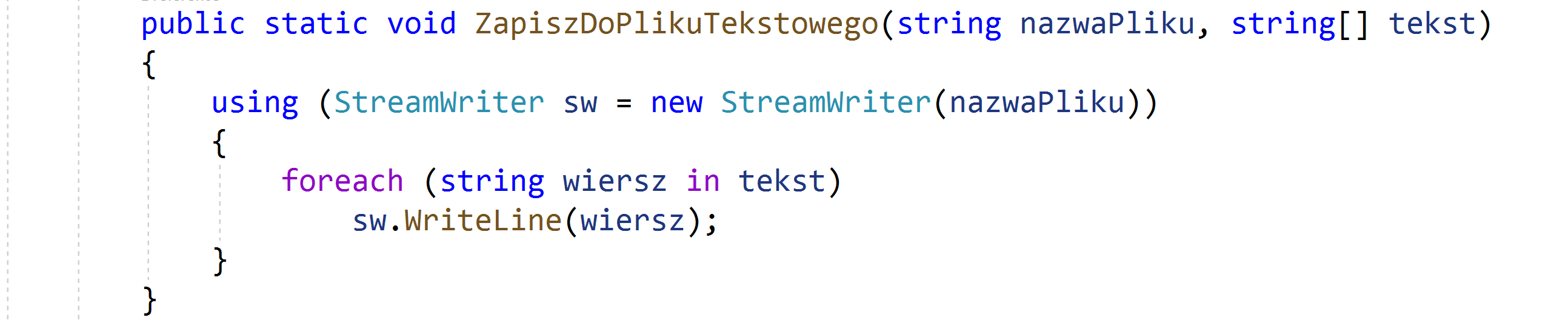
{

foreach (string wiersz in tekst)

sw.WriteLine(wiersz);

}

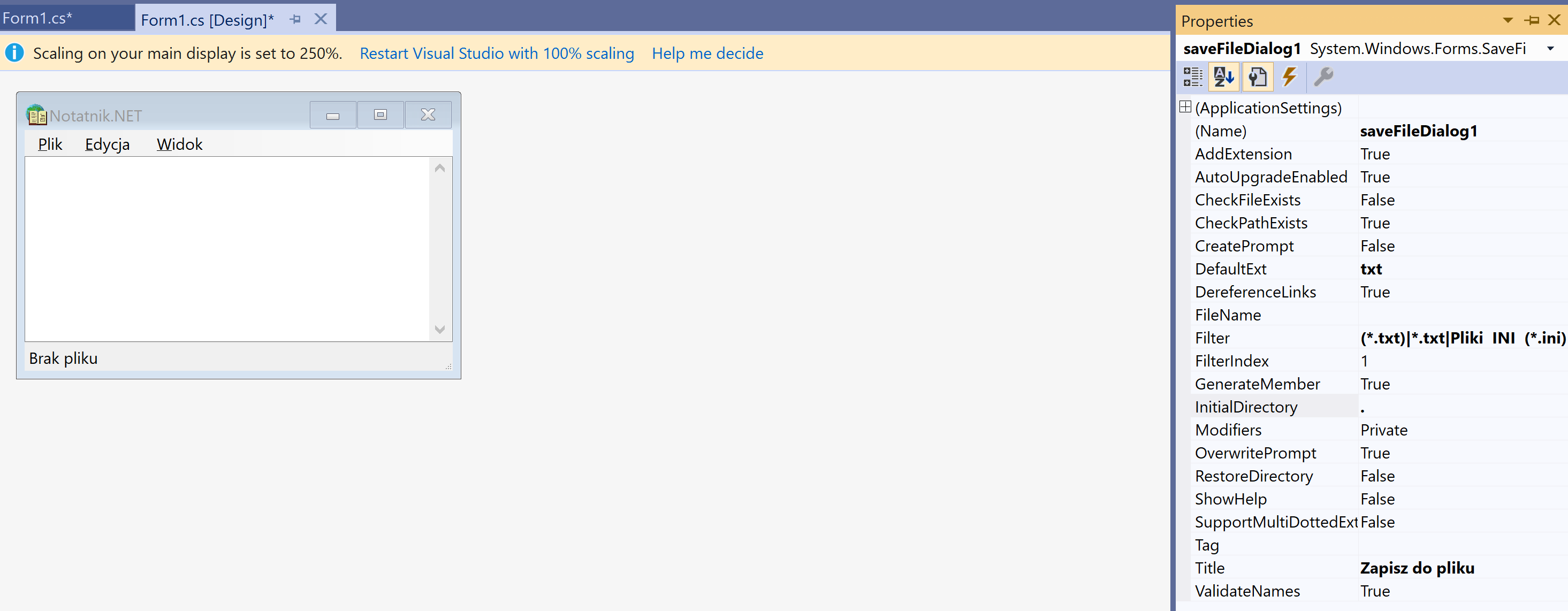
}



1. W widoku projektowania umieść na formie komponent SaveFileDialog

z zakładki *Dialogs* podokna *Toolbox*.

1. Zaznacz go i w oknie własności na zakładce *Properties*:
   1. zmień własność Title na *Zapisz do pliku*;
   2. DefaultExt na *txt*;
   3. odnajdź jego własność Filter i w związanym z nią polu wpisz łańcuch definiujący filtr: Pliki tekstowe (\*.txt)|\*.txt|Pliki INI (\*.ini)|\*.ini| Pliki źródłowe (\*.cs)|\*.cs| Wszystkie pliki (\*.\*)|\*.\*;
   4. zmień InitialDirectory na kropkę.



1. Utwórz metodę zdarzeniową do pozycji *Zapisz jako…* w menu *Plik* i umieść w niej poniższy kod.

*Metoda zdarzeniowa związana z poleceniem Zapisz jako z menu Plik*

private void zapiszJakoToolStripMenuItem\_Click(object sender, EventArgs e)

{

string nazwaPliku = openFileDialog1.FileName;

if (nazwaPliku.Length > 0) saveFileDialog1.FileName = nazwaPliku;

if (saveFileDialog1.ShowDialog() == DialogResult.OK)

{

nazwaPliku = saveFileDialog1.FileName;

ZapiszDoPlikuTekstowego(nazwaPliku, textBox1.Lines);

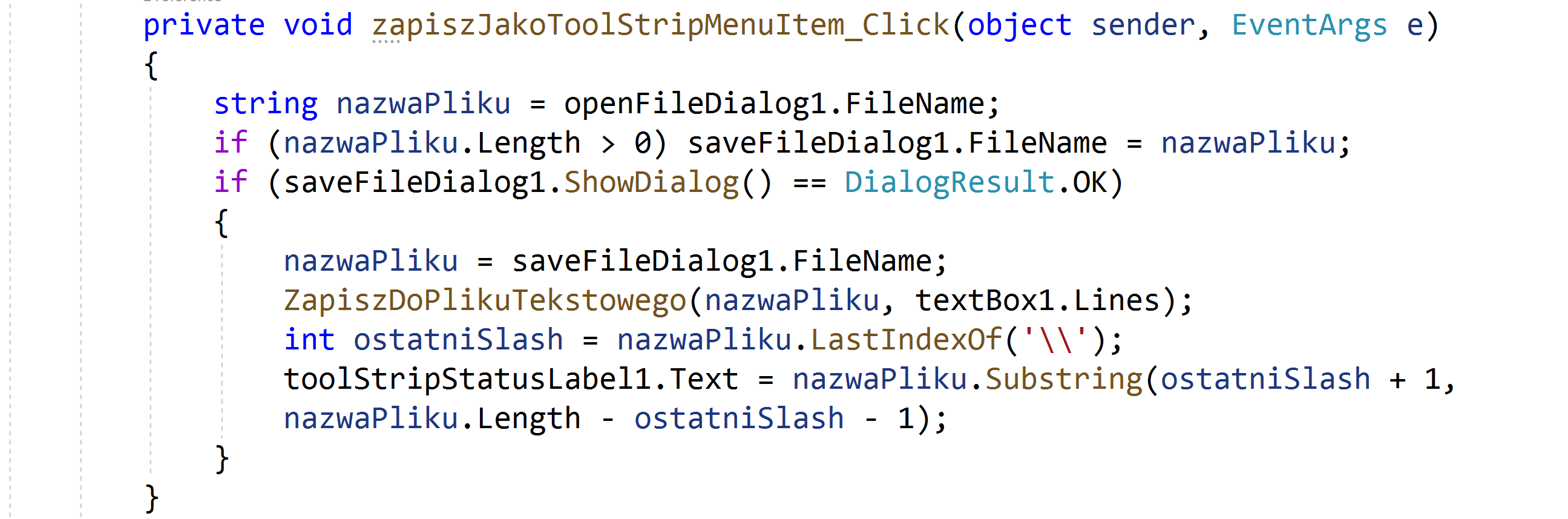
int ostatniSlash = nazwaPliku.LastIndexOf('\\');

toolStripStatusLabel1.Text = nazwaPliku.Substring(ostatniSlash + 1,

nazwaPliku.Length - ostatniSlash - 1);

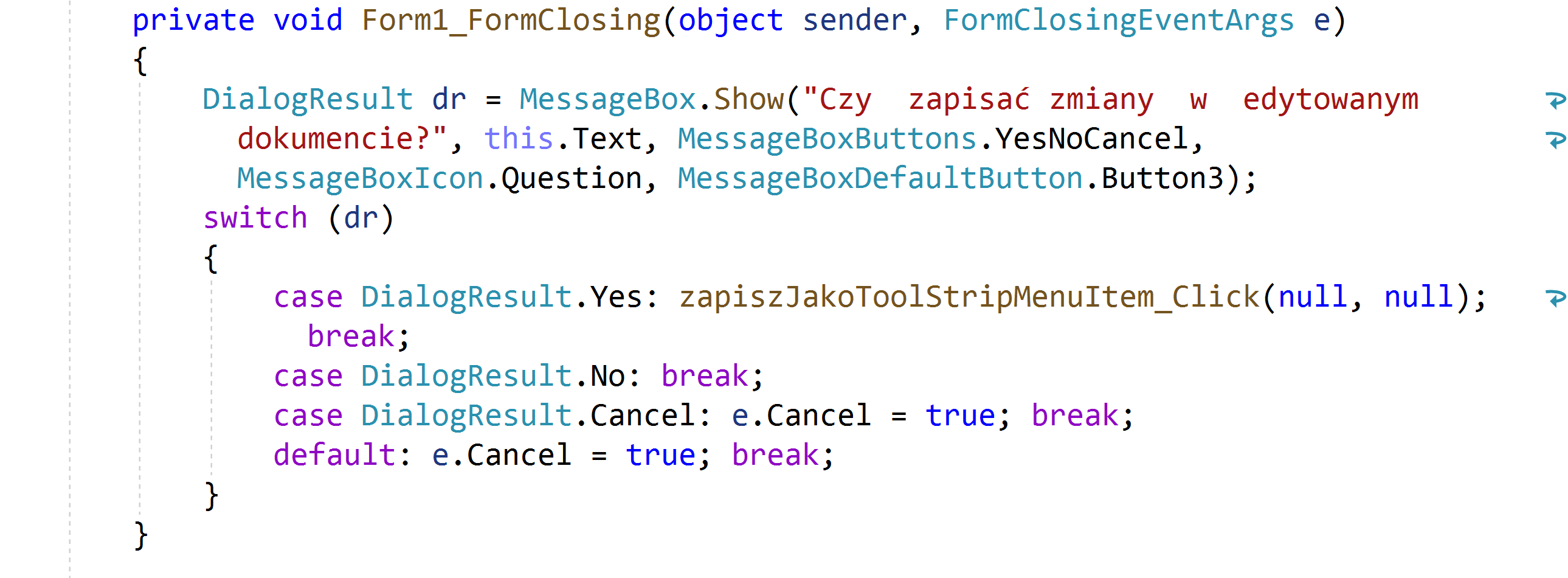
}

}



1. Teraz możesz uzupełnić metodę Form1\_FormClosing wywołaniem powyższej metody z argumentami równymi null (należy je wstawić zamiast tymczasowego komunikatu wyróżnionego w kodzie poniżej):

case DialogResult.Yes: zapiszJakoToolStripMenuItem\_Click(null, null); break;

Na początku metody zapiszJakoToolStripMenuItem\_Click podejmowana jest próba pobrania z własności FileName obiektu openFileDialog ścieżki dostępu do pliku, z które go ewentualnie wcześniej wczytany był tekst. Odczytanie łańcucha z tej własności może zwrócić łańcuch pusty jedynie wtedy, gdy wcześniej nie został wczytany żaden plik. Odczytana ścieżka pliku przypisywana jest własności saveFileDialog1.FileName. Ten komponent sam rozpoznaje w ścieżce pliku właściwy katalog i zastępuje nim katalog domyślny z własności InitialDirectory. W efekcie okno dialogowe pozwalające na wybór pliku do zapisania pokazuje katalog, z którego wybrany został plik do wczytania.

##### Okna dialogowe wyboru czcionki i koloru

Przygotujmy teraz metody związane z pozycjami menu *Widok*, które będą pozwalać na wybór koloru tła notatnika i użytej w nim czcionki. Ponownie użyjemy standardo- wych okien dialogowych. W przypadku czcionki umożliwimy także wybór jej koloru.

1. Na podglądzie formy umieść okno dialogowe ColorDialog z zakładki *Dialogs*.
2. Następnie dla pozycji *Tło…* w menu *Widok* utwórz metodę zdarzeniową z poleceniami widocznymi w kodzie poniżej.

*Metoda zdarzeniowa wywoływana po kliknięciu pozycji Czcionka w menu Widok*

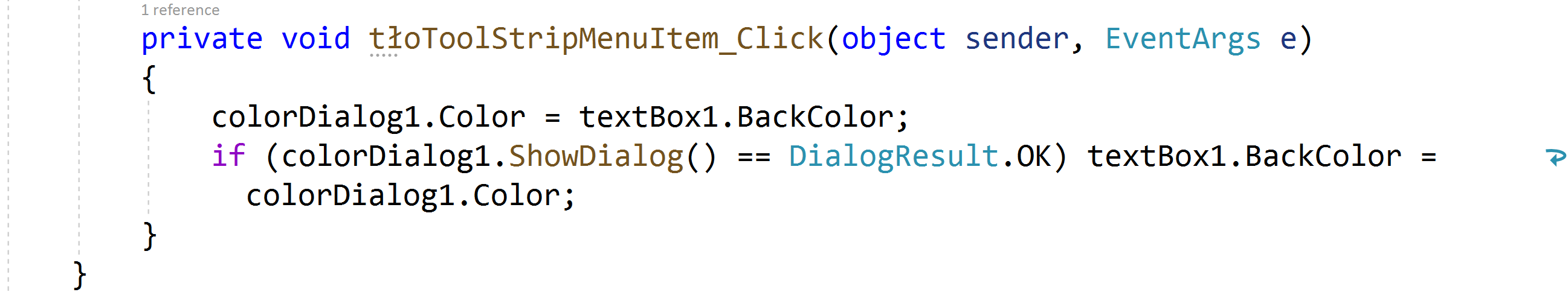
private void tłoToolStripMenuItem\_Click(object sender, EventArgs e)

{

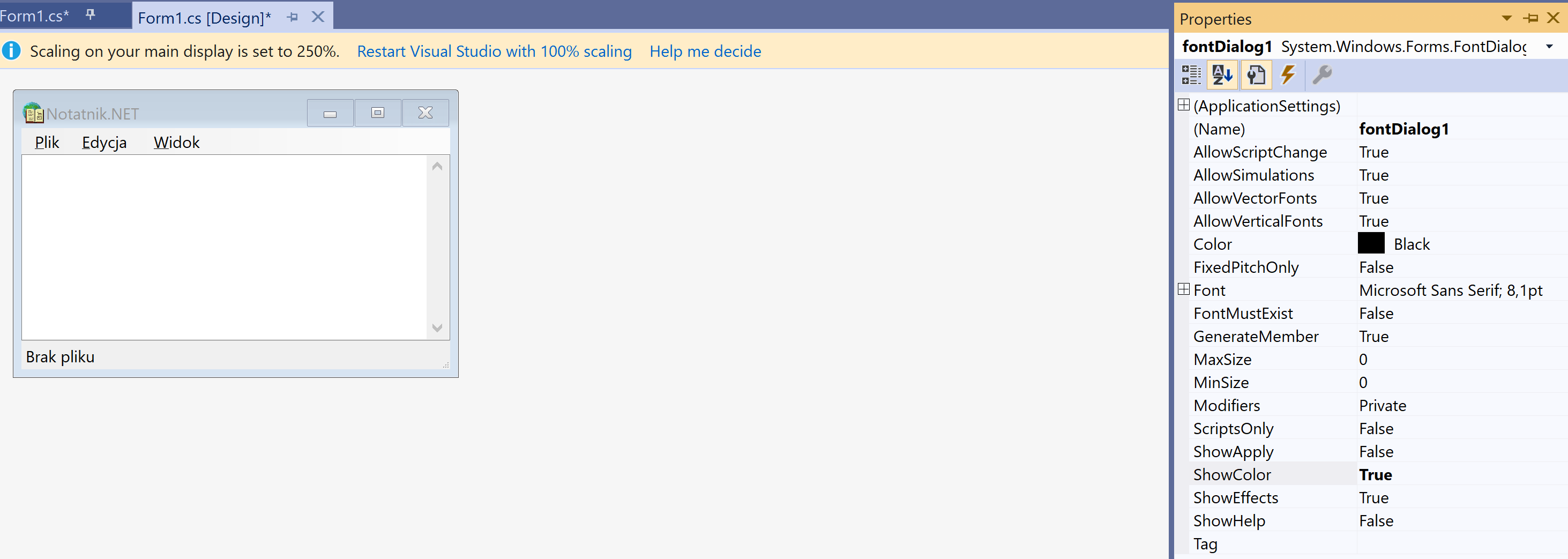
colorDialog1.Color = textBox1.BackColor;

if (colorDialog1.ShowDialog() == DialogResult.OK) textBox1.BackColor = colorDialog1.Color;

}



1. W podobny sposób możesz przygotować metodę pozwalającą na wybór czcionki i jej koloru (kod poniżej), korzystającą z komponentu FontDialog.
2. Po umieszczeniu tego komponentu na podglądzie okna ustaw jego własność ShowColor na True.



1. Następnie w metodzie zdarzeniowej związanej ze zdarzeniem Click pozycji *Czcionka...* z menu *Widok* umieść polecenia widoczne w kodzie poniżej.

*Metoda zdarzeniowa wywoływana po kliknięciu pozycji Czcionka w menu Widok*

private void czcionkaToolStripMenuItem\_Click(object sender, EventArgs e)

{

fontDialog1.Font = textBox1.Font; fontDialog1.Color = textBox1.ForeColor;

if (fontDialog1.ShowDialog() == DialogResult.OK)

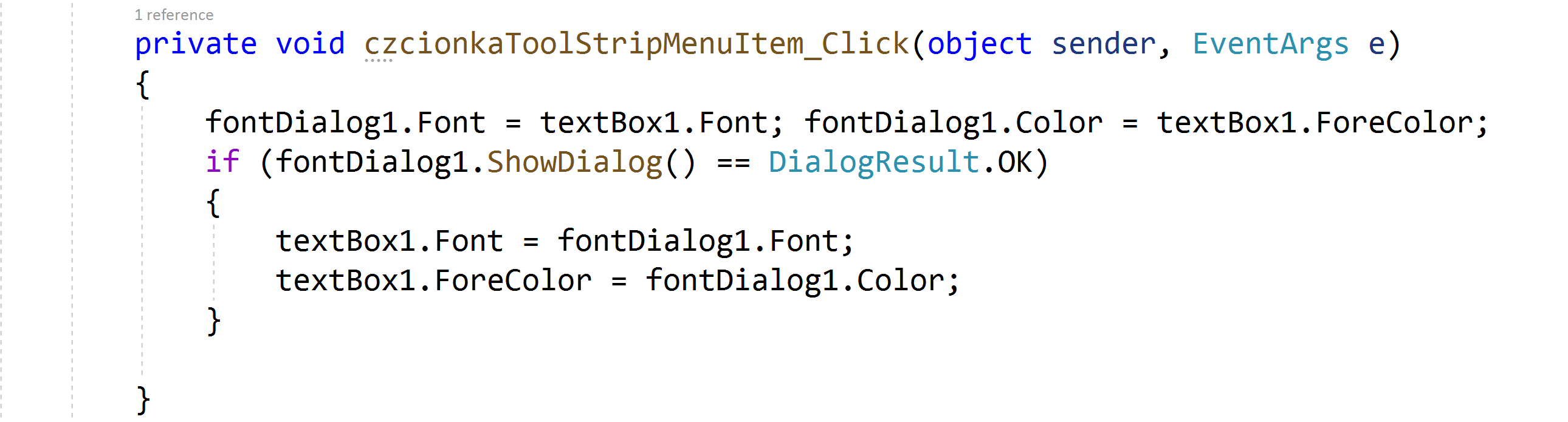
{

textBox1.Font = fontDialog1.Font;

textBox1.ForeColor = fontDialog1.Color;

}

}



W obu metodach polecenia sprzed instrukcji if uzgadniają początkową wartość pre- zentowaną w oknie dialogowym z bieżącą wartością odpowiednich własności komponentu textBox1. Dzięki temu okno dialogowe pozwala na modyfikowanie bieżącej wartości, zamiast wymuszać za każdym razem określanie jej od nowa.

### Edycja i korzystanie ze schowka

Kolejna grupa poleceń związana jest z menu *Edycja*. Ich obsługa okazuje się łatwa, ponieważ komponent TextBox posiada odpowiednie metody, co nasze zadanie sprowadza do ich właściwego wywołania. Tworzymy wobec tego metody zdarzeniowe do wszystkich pozycji z menu *Edycja* i odpowiednio wywołujemy w nich metody widoczne w kodzie poniżej.

*Zbiór metod związanych z pozycjami menu Edycja*

private void wytnijToolStripMenuItem\_Click(object sender, EventArgs e)

{

textBox1.Cut();

}

private void kopiujToolStripMenuItem\_Click(object sender, EventArgs e)

{

textBox1.Copy();

}

private void wklejToolStripMenuItem\_Click(object sender, EventArgs e)

{

textBox1.Paste();

}

private void usuńToolStripMenuItem\_Click(object sender, EventArgs e)

{

textBox1.SelectedText = "";

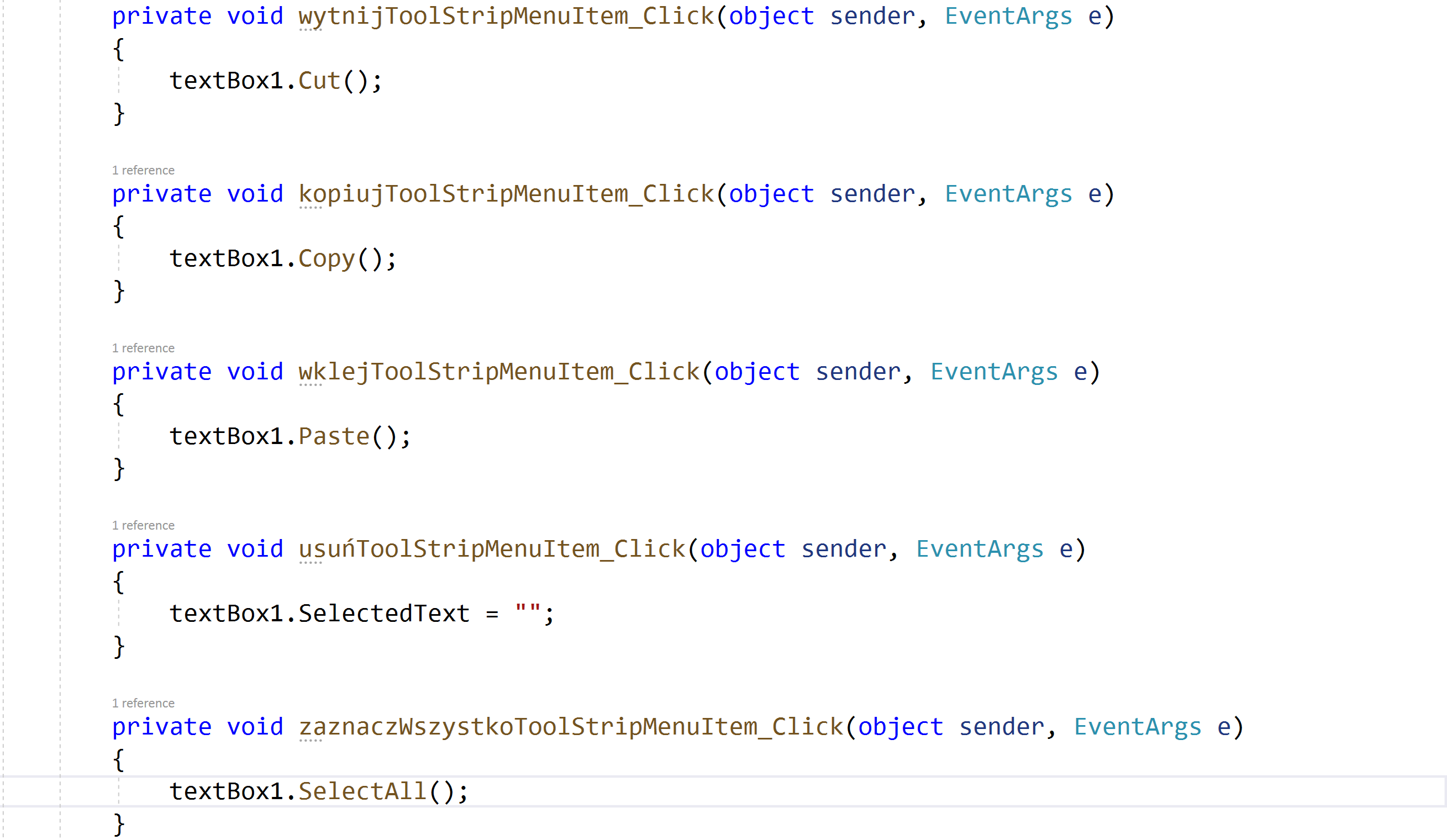
}

private void zaznaczWszystkoToolStripMenuItem\_Click(object sender, EventArgs e)

{

textBox1.SelectAll();

}



Zwróćmy uwagę na to, że zawartość utworzonego podmenu *Edycja* pokrywa się przynajmniej częściowo z zawartością menu kontekstowego domyślnie przypisanego do pola tekstowego. Wywoływane z obu menu metody obiektu textBox1 też są tożsame.

### Drukowanie

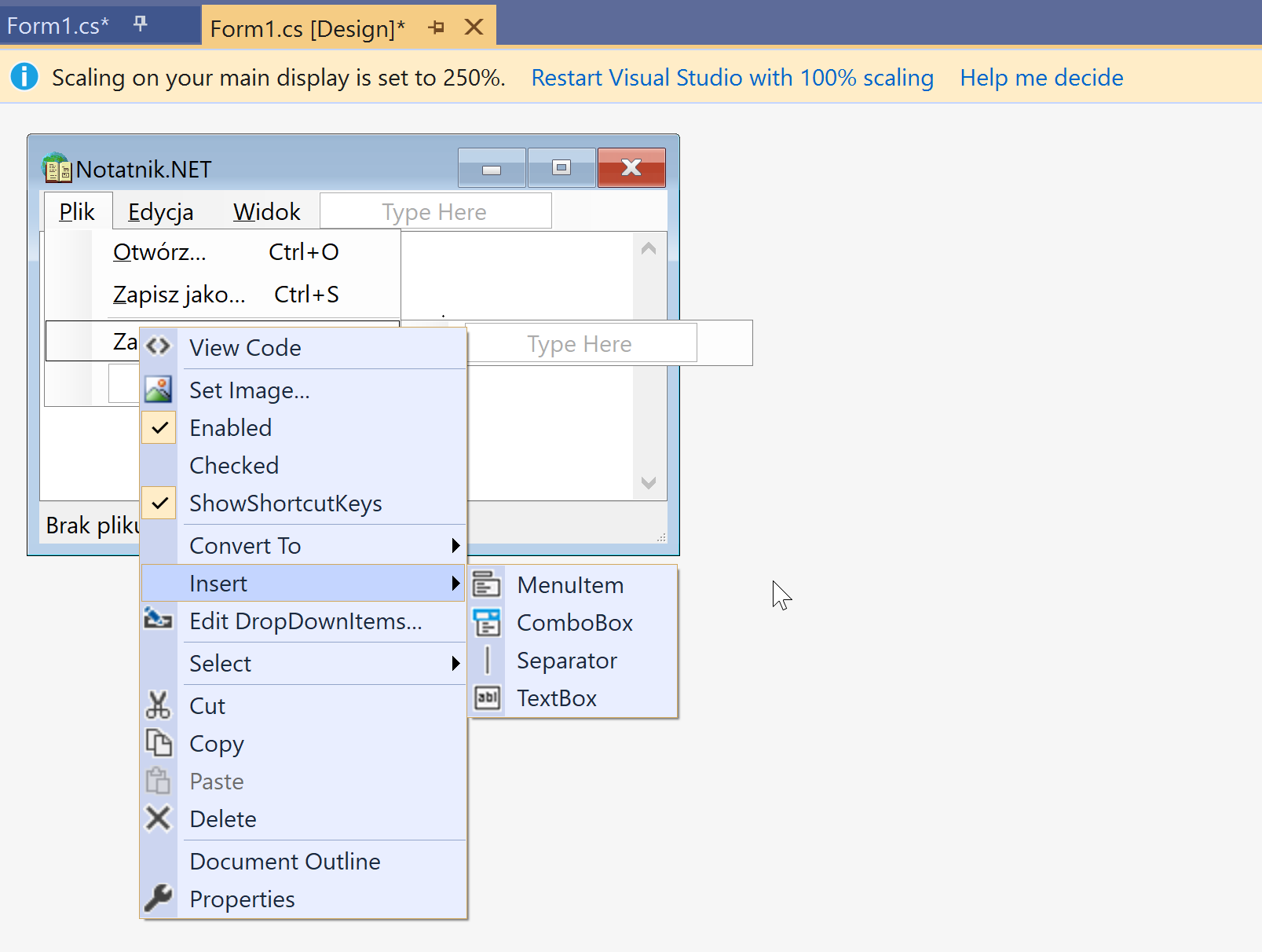
Ostatnią rzeczą, jaką zaimplementujemy w naszym notatniku, jest drukowanie edytowanego tekstu. Podobnie jak w przypadku operacji na plikach, tak i tu nie możemy liczyć na wygodną w użyciu metodę, która czarną robotę wykonałaby za nas. To oznacza, że i tym razem odpowiedni kod musimy napisać samodzielnie. Za to po przygotowaniu metody organizującej wydruk stron niemal za darmo uzyskamy podgląd wydruku.

Do drukowania w trybie graficznym należy użyć klasy Graphics — jej instancja będzie reprezentowała powierzchnię drukowanej kartki. Zrobimy użytek z metody DrawString, która pozwala na umieszczenie tekstu w dowolnym miejscu reprezentowanej przez klasę powierzchni (kartki). W przypadku drukowania inne metody (DrawText i TextRender) — niestety — się do tego nie nadają.

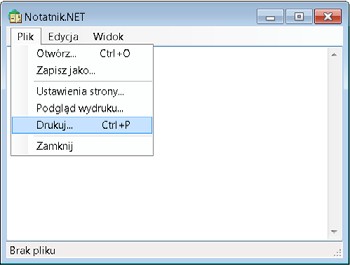
##### Polecenia związane z drukowaniem w menu Plik

Zacznijmy jednak od uzupełnienia naszego menu *Plik* o pozycje związane z drukowaniem.

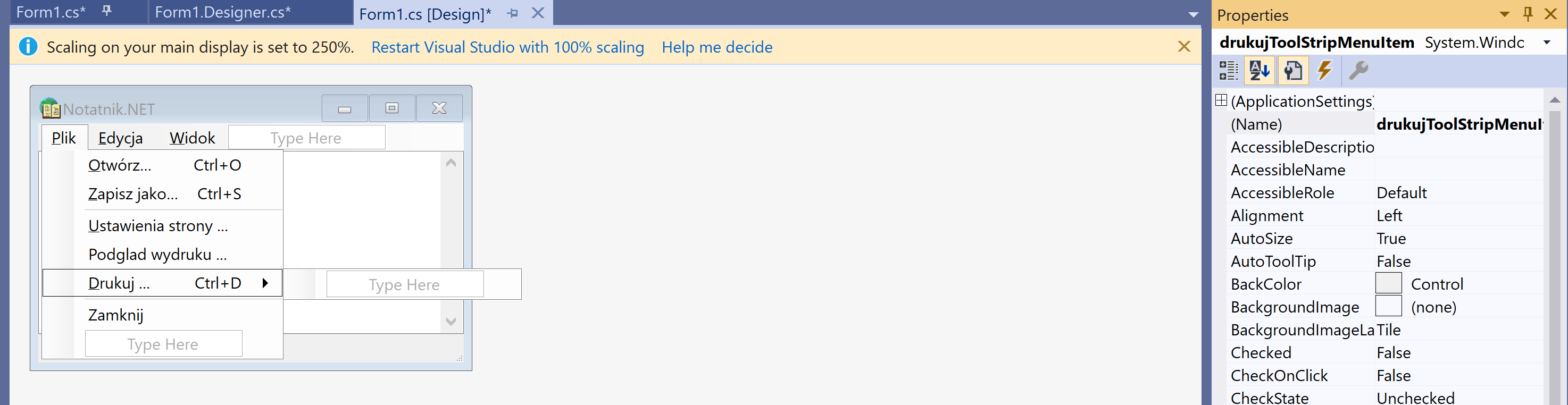
1. Przejdź do widoku projektowania.
2. Zaznacz komponent menuStrip1.
3. W widocznym wówczas na podglądzie formy edytorze menu zaznacz pozycję *Zamknij* w podmenu *Plik* i z menu kontekstowego wybierz *Insert/Menu Item*.



1. Kliknij dodany element i zmień jego etykietę na *&Ustawienia strony…*.



1. Powtórz czynności z punktów 3. i 4., aby dodać polecenia *Podgląd wydruku…* oraz *&Drukuj…*.
2. Ustaw klawisz skrótu tego ostatniego na *Ctrl+P*.
3. Za poleceniem *Drukuj* wstaw separator.
4. Tak dodane komponenty mają nazwy toolStripMenuItem*n*, gdzie *n* to kolejna liczba naturalna. Jeżeli chcesz nadać im bardziej przyjazne nazwy, zaznacz je w widoku projektowania, przejdź do okna własności i zmień ich nazwy, korzystając z własności (Name). Użyj nazw ustawieniaStronyToolStripMenuItem, podgladWydrukuToolStripMenuItem i drukujToolStripMenuItem.

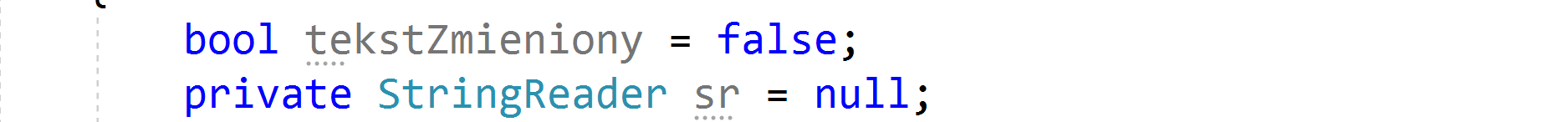
 Warto zwrócić uwagę, że wywołana w punkcie 3. gałąź *Insert* menu kontekstowego zawiera nie tylko pozycję *Menu Item*, ale również rozwijaną listę i pole tekstowe. Takie komponenty możemy również umieścić w menu głównym aplikacji.

##### Metoda zarządzająca drukowaniem tekstu

Kolejną czynnością będzie dodanie do aplikacji komponentu PrintDocument reprezentującego drukowany dokument i wykorzystanie jego zdarzenia PrintPage do wydrukowania zawartości pola tekstowego. Do inicjacji wydruku wykorzystamy okno dialogowe udostępniane przez kontrolkę PrintDialog.

1. Na podglądzie formy umieść komponent PrintDocument z zakładki *Printing*.
2. Zdefiniuj prywatne pole:

private StringReader sr = null;



1. W widoku projektowania interfejsu zaznacz obiekt printDocument1, w oknie własności przejdź na zakładkę zdarzeń i utwórz metodę związaną ze zdarzeniem PrintPage. Umieść w niej pętlę przenoszącą zawartość pola tekstowego linia po linii na powierzchnię drukowanej strony (kod poniżej).

*Metoda odpowiedzialna za przygotowanie wydruku poszczególnych stron tekstu*

private void printDocument1\_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)

{

Font czcionka = textBox1.Font;

int wysokoscWiersza = (int)czcionka.GetHeight(e.Graphics);

int iloscLinii = e.MarginBounds.Height / wysokoscWiersza;

if (sr == null) sr = new StringReader(textBox1.Text); //

czy pierwsza strona?

e.HasMorePages = true;

for (int i = 0; i < iloscLinii; i++)

{

string wiersz = sr.ReadLine(); if (wiersz == null)

{

e.HasMorePages = false;

sr = null;

break;

}

e.Graphics.DrawString(wiersz,

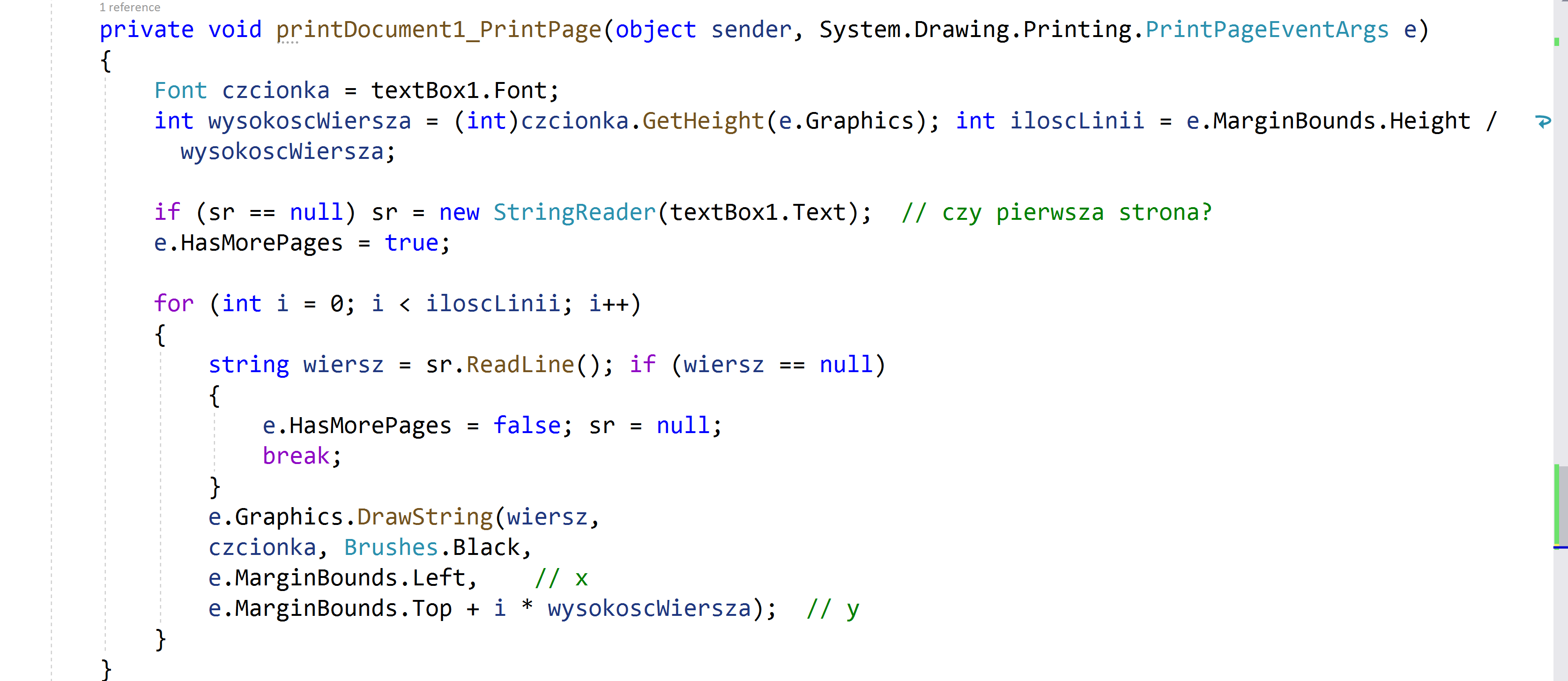
czcionka, Brushes.Black,

e.MarginBounds.Left, // x

e.MarginBounds.Top + i \* wysokoscWiersza); // y

}

}

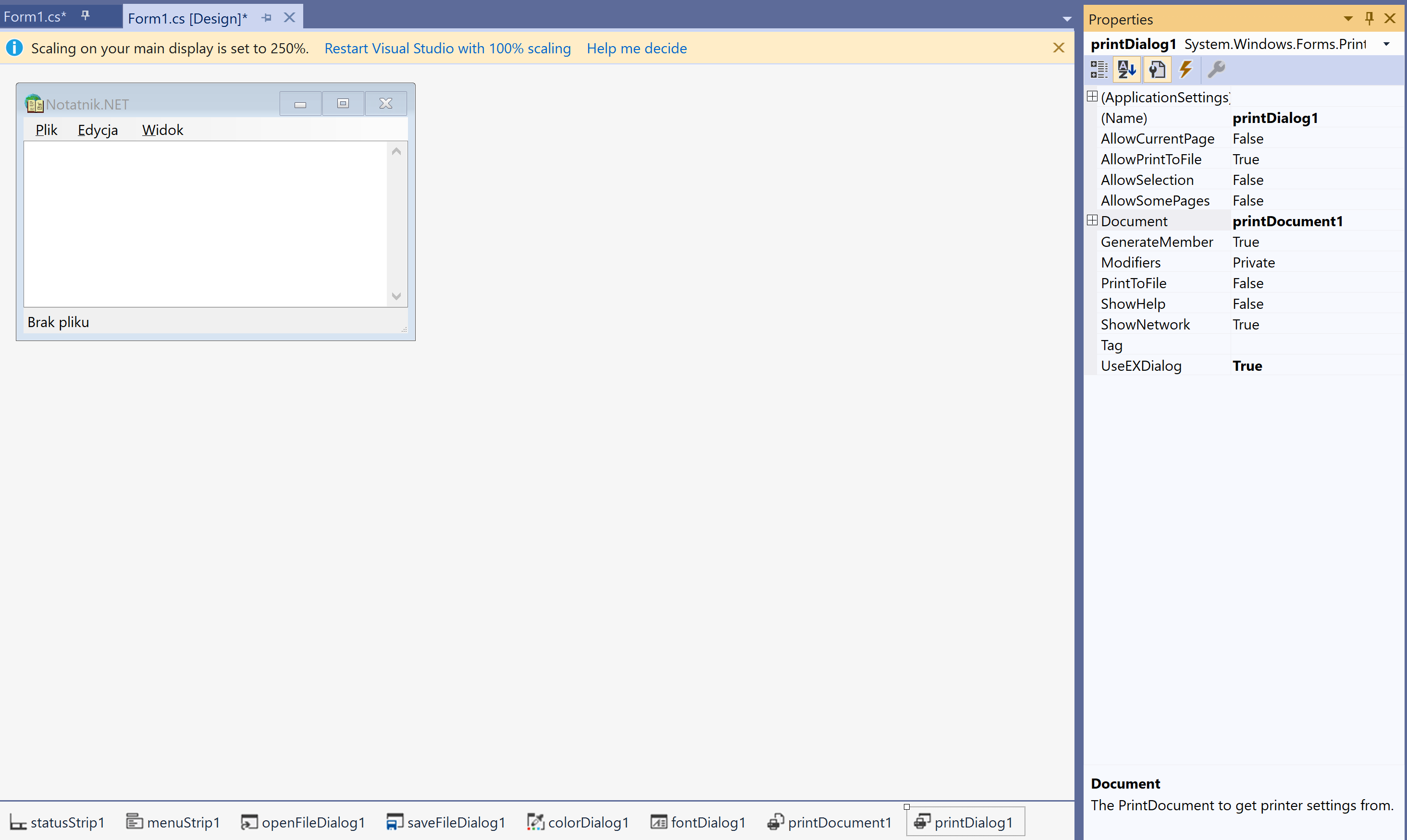


1. Dopuściliśmy się tu poważnego uproszczenia, zakładając, że

żadna linia tekstu nie jest dłuższa niż szerokość kartki. To założenie

jest zwykle nieprawdziwe i dlatego za chwilę do tego problemu wrócimy.

1. Teraz pozostaje uruchomić proces drukowania. W tym celu:
   1. na podglądzie formy umieść komponent PrintDialog z zakładki *Printing*;
   2. za pomocą okna *Properties* powiąż jego własność Document z obiektem printDocument1;



* 1. utwórz metodę zdarzeniową do pozycji *Drukuj…* w menu *Plik* i umieść w niej polecenia z kod poniżej.

*Metoda zdarzeniowa związana z poleceniem Drukuj z menu Plik*

private void drukujToolStripMenuItem\_Click(object sender, EventArgs e)

{

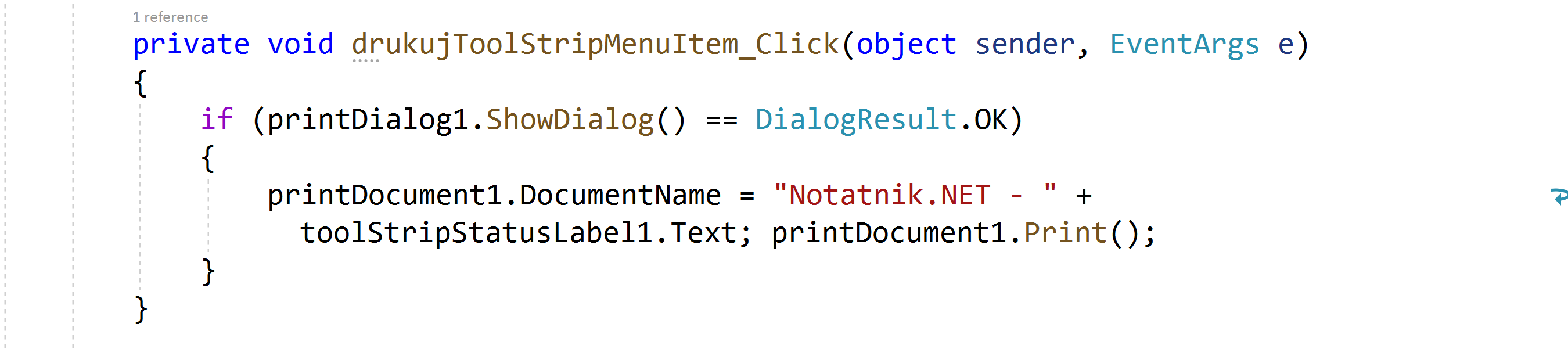
if (printDialog1.ShowDialog() == DialogResult.OK)

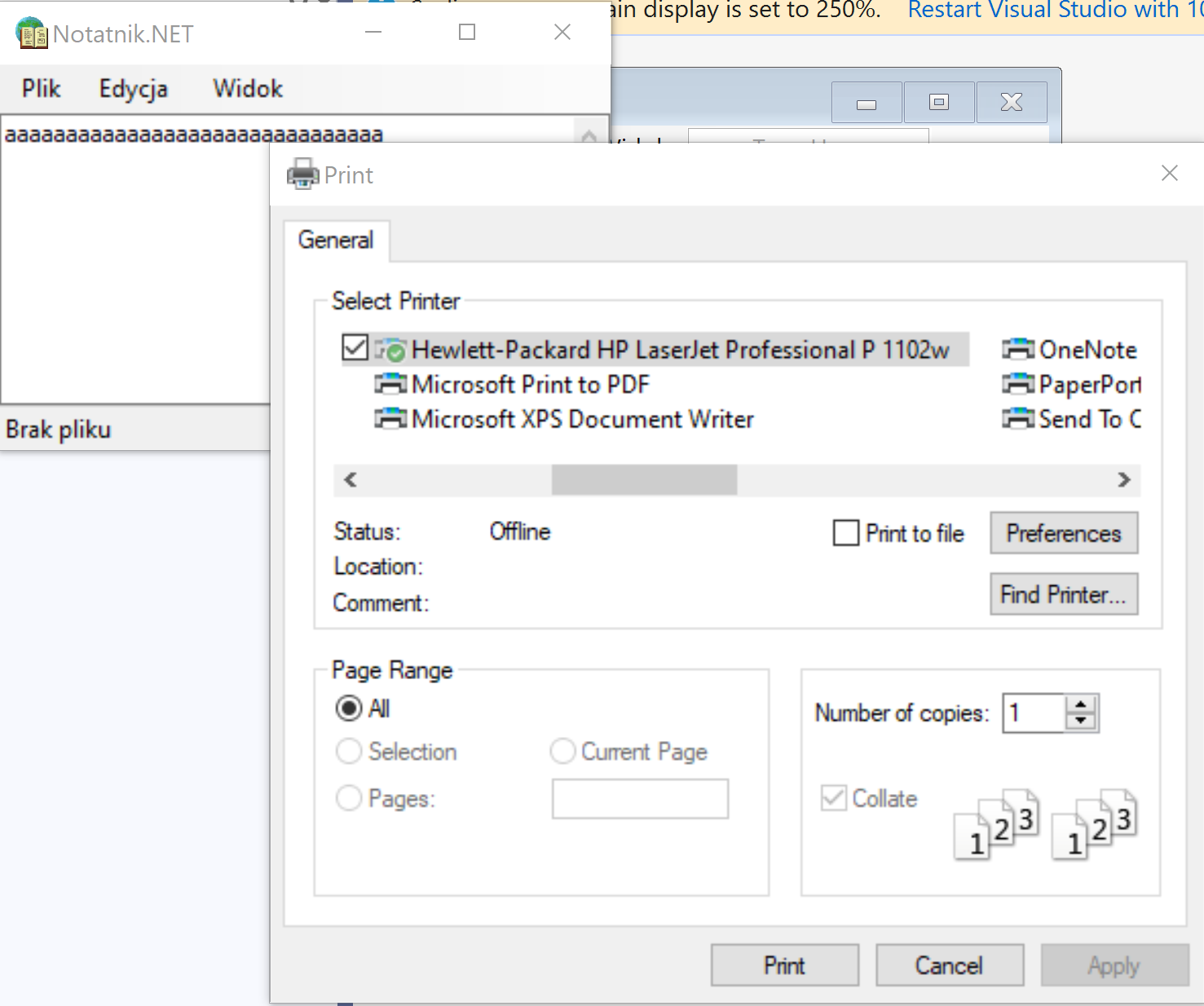
{

printDocument1.DocumentName = "Notatnik.NET - " + toolStripStatusLabel1.Text; printDocument1.Print();

}

}



Komentarza wymaga metoda zdefiniowana w punkcie 3. Przede wszystkim należy zauważyć, że zdarzenie, z którym jest związana, a więc printDocument1.PrintPage, jest wywoływane w momencie drukowania pojedynczej strony. Użytkownik sam musi zadbać o to, co będzie na tej stronie umieszczone. Co więcej, ustalając wartość e.HasMorePages, sam musi poinformować system, czy w wydruku będzie więcej stron (czy metoda zdarzeniowa ma być uruchomiona ponownie). Komponent printDocument1 nie udostępnia żadnego pola, które informowałoby o tym, ile stron zostało już wydrukowanych. To pozostawia programiście pełną swobodę korzystania z drukarki, ale jednocześnie zmusza go do wykonania samodzielnie całej pracy związanej z organizacją wydruku strony.

Jedno wywołanie metody zdarzeniowej printDocument1\_PrintPage związane jest z wydrukowaniem jednej strony.

Metoda widoczna w kodzie powyżej korzysta ze strumienia StringReader, co pozwala uniknąć kłopotliwego wyliczania zakresu linii, które mają być drukowane na konkret- nej stronie. Dzięki inicjacji strumienia przy drukowaniu pierwszej strony i zwalnianiu go dopiero przy ostatniej możemy pobierać z niego kolejne wiersze tekstu linia po linii bez obliczania, na której aktualnie stronie się znajdujemy.

Wszystkie informacje o ustawieniach strony pobierane są z obiektu e

typu PrintPageEventArgs. Zawiera obiekt Graphics, który

umożliwia umieszczanie na powierzchni strony kolejnych linii.

Korzystając z metod tego obiektu, bez trudu można tam umieścić także

obraz.

Przygotowanie metody związanej ze zdarzeniem PagePrint jest najtrudniejszym zadaniem związanym z drukowaniem. Tego zadania jeszcze nie zakończyliśmy. Nadal pozostaje do rozwiązania problem długich linii (akapitów). Jednak prowizoryczna wersja już działa. Aby wydrukować dokument, wystarczy teraz wywołać metodę Print komponentu printDocument1 (punkt 4.). Korzystamy przy tym z okna dialogowego PrintDialog. W tym oknie poza przyciskami *Drukuj* i *Anuluj*, które kończą działanie metody PrintDialog.ShowDialog, jest także przycisk *Preferencje* lub *Usta- wienia strony* (w zależności od wersji systemu operacyjnego). Kliknięcie tego przycisku powoduje uruchomienie okna dialogowego pozwalającego na ustawienie drukarki i własności drukowania. Możemy również wywołać to okno samodzielnie — do tego celu przygotowaliśmy w menu *Plik* pozycję o etykiecie *Ustawienia strony…*.

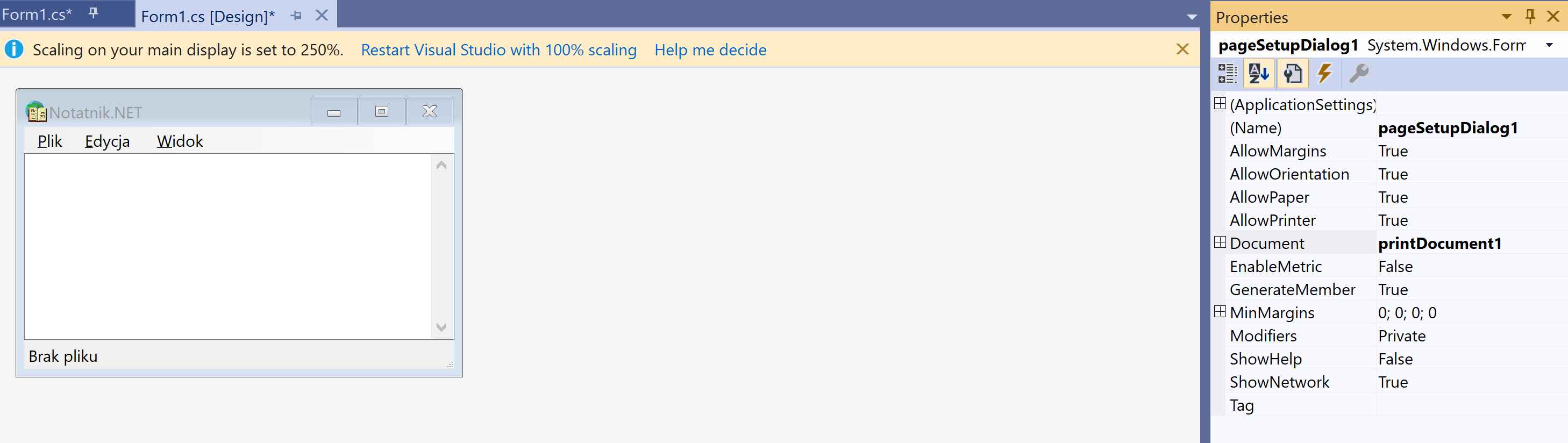
##### Okno dialogowe ustawień wydruku

Dodaj do formy komponent PageSetupDialog pozwalający na ustalenie przez użytkownika aplikacji sposobu, w jaki będą drukowane strony.

1. Na formie umieść komponent PageSetupDialog (powstanie obiekt

pageSetupDialog1).

1. Jego własność Document powiąż z printDocument1.

****

1. Utwórz metodę zdarzeniową do pozycji *Ustawienia strony…* z menu

*Plik* i umieść w niej wywołanie metody ShowDialog komponentu pageSetupDialog1 (kod poniżej).

*Metoda związana z pozycją Ustawienia strony menu Plik*

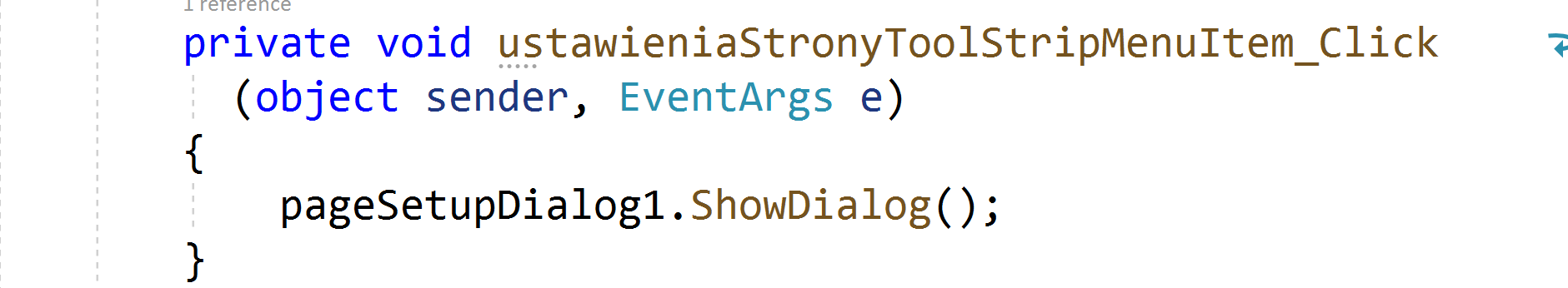
private void ustawieniaStronyToolStripMenuItem\_Click(object sender,

EventArgs e)

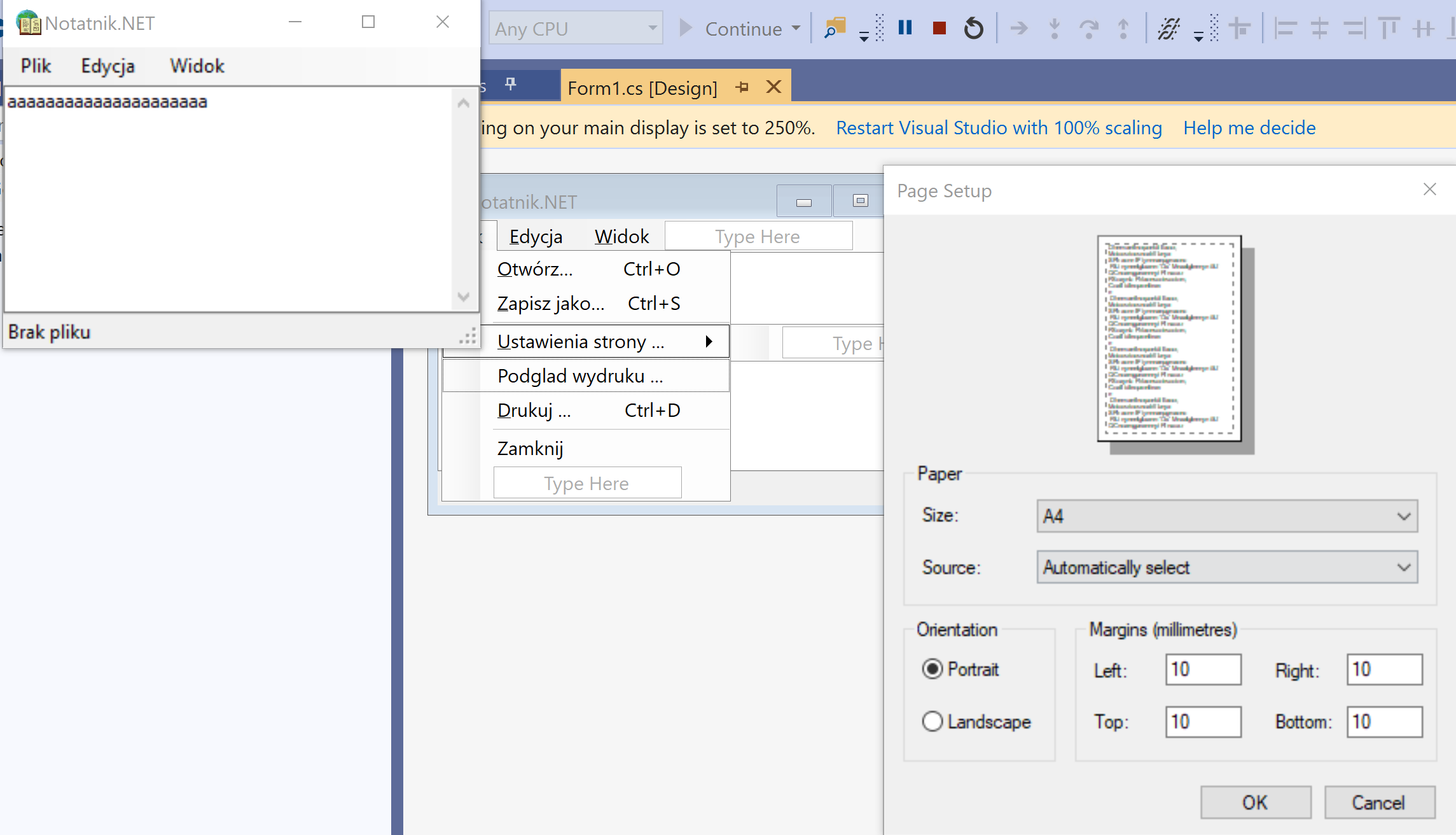
{

pageSetupDialog1.ShowDialog();

}



Nowe okno dialogowe umożliwia zmianę ustawień drukowania związanych z kom-ponentem printDocument1. Okno, które zobaczymy, może zależeć nie tylko od wersji systemu, ale również od wybranej drukarki.



##### Podgląd wydruku

Jeżeli podołaliśmy trudowi przygotowania metody związanej ze

zdarzeniem printDocument1.PagePrint, to niewielkim kosztem

możemy uruchomić podgląd wydruku. Służy do tego okno

dialogowe PrintPreviewDialog. Sam obszar podglądu, jeżeli chcemy

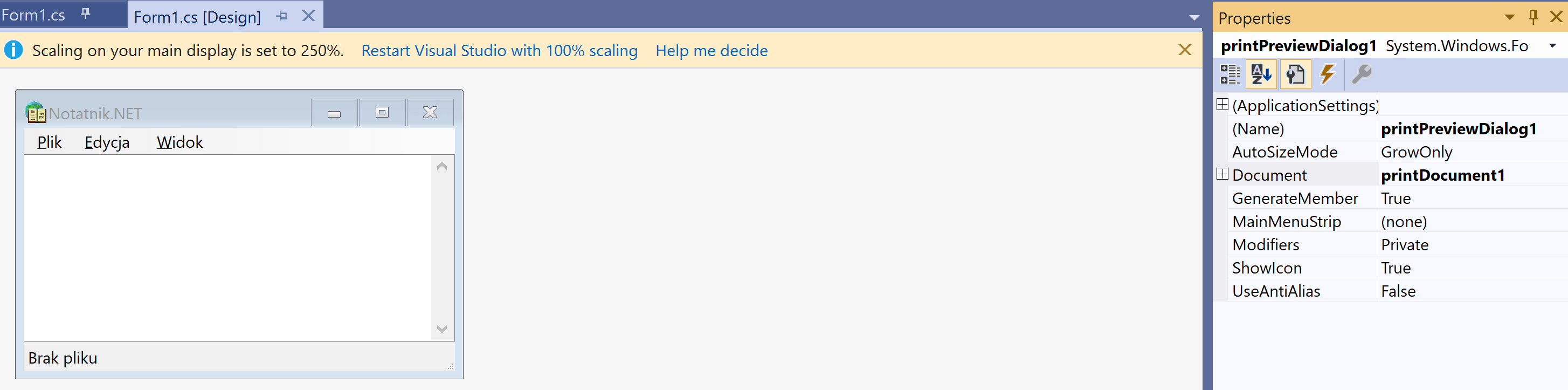
go wkomponować w główne okno aplikacji, jest również dostępny

dzięki komponentowi PrintPreviewControl. My jednak, głównie

aby oszczędzić sobie kłopotu, skorzystamy z wariantu podglądu w

osobnym oknie.

1. Na formie umieść komponent PrintPreviewDialog (nie należy mylić go z PrintPreviewControl).
2. Jego własność Document ustaw na printDocument1.



1. Utwórz metodę zdarzeniową do pozycji *Podgląd wydruku…* z menu

*Plik* i umieść w niej wywołanie metody ShowDialog komponentu printDocument1 (kod poniżej).

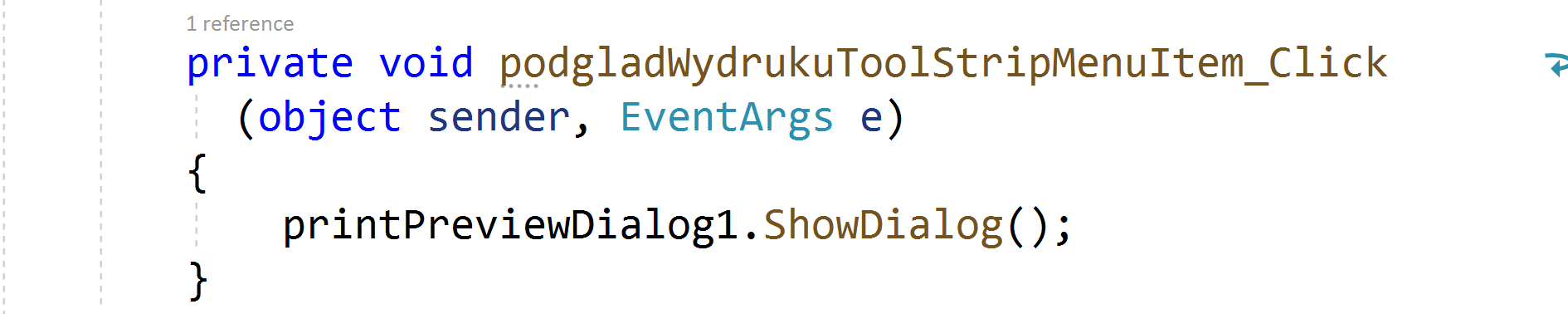
*Metoda związana z pozycją Podgląd wydruku menu Plik*

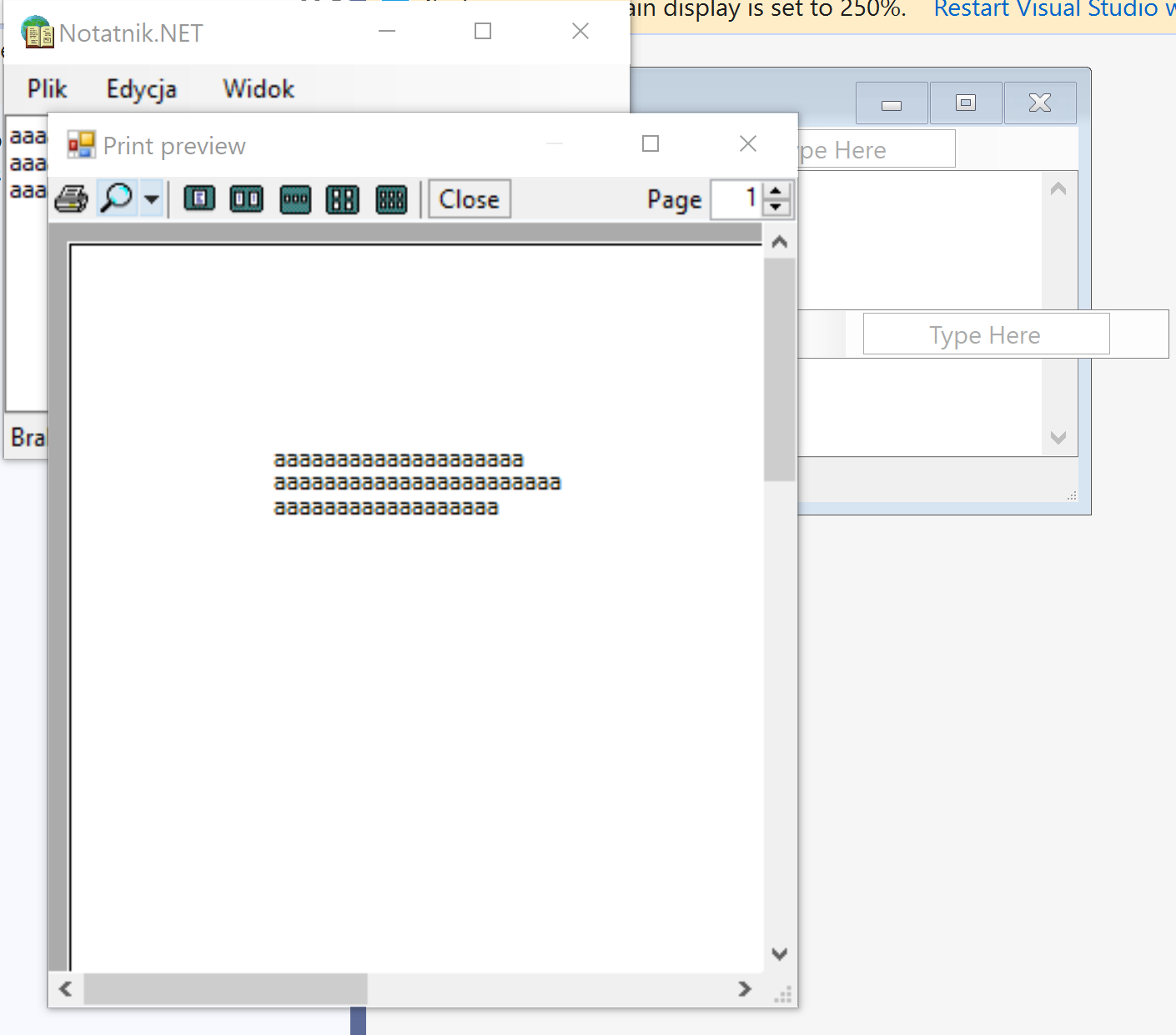
private void podgladWydrukuToolStripMenuItem\_Click(object sender, EventArgs e)

{

printPreviewDialog1.ShowDialog();

}

Po uruchomieniu notatnika i wybraniu pozycji *Podgląd wydruku…* powinniśmy zobaczyć okno podobne do widocznego powyżej.



##### Rozwiązanie problemu długich linii

Nie można już dłużej odkładać problemu długich linii. Co zrobić, aby pokazywane były w całości w wielu liniach? Bardzo kłopotliwe byłoby ich dzielenie w trakcie drukowania, szczególnie jeśli pojawią się na końcu strony. Najbardziej dogodnym momentem na podział linii wydaje się chwila, w której tworzony jest strumień repre- zentujący zawartość pola tekstowego, tj. linia if (sr == null) sr = new StringReader (textBox1.Text);. W tym celu musimy zastąpić argument konstruktora klasy StringReader, a więc łańcuch odczytany z własności textBox1.Text, łańcuchem, w którym długie linie będą podzielone. Modyfikujemy wobec tego metodę printDocument1\_PrintPage zgodnie z kodem poniżej.

*Metoda odpowiedzialna za wydruk po zmianach związanych z drukowaniem długich linii*

private void printDocument1\_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)

{

Font czcionka = textBox1.Font;

int wysokoscWiersza = (int)czcionka.GetHeight(e.Graphics);

int iloscLinii = e.MarginBounds.Height / wysokoscWiersza;

// przy dzieleniu linii nie dbamy o spacje

if (sr == null)

{

string tekst = "";

foreach (string wiersz in textBox1.Lines)

{

float szerokosc = e.Graphics.MeasureString(wiersz, czcionka).Width;

if (szerokosc < e.MarginBounds.Width)

{

tekst += wiersz + "\n";

}

else

{

float sredniaSzerokoscLitery = szerokosc / wiersz.Length;

int ileLiterWWierszu = (int)(e.MarginBounds.Width /

sredniaSzerokoscLitery);

int ileRazy = wiersz.Length / ileLiterWWierszu; // ile pełnych wierszy

for (int i = 0; i < ileRazy; i++)

{

tekst += wiersz.Substring(i \* ileLiterWWierszu,

ileLiterWWierszu) + "\n";

}

// kopiowanie reszty wiersza

tekst += wiersz.Substring(ileRazy \* ileLiterWWierszu) + "\n";

} // if-else

} // foreach

sr = new StringReader(tekst);

} // if (sr==null)

e.HasMorePages = true;

...*// dalszy ciąg metody*

W zmodyfikowanym fragmencie obliczamy szerokość odczytanego z pola tekstowego wiersza (akapitu). Jeżeli jest mniejsza od szerokości strony, kopiujemy ją do łańcucha tekst i przechodzimy do kolejnego. Natomiast jeśli jest większa, szacujemy liczbę liter w wierszu i dzielimy go na równe fragmenty. Następnie każdy z nich umieszczamy w zmiennej tekst jako osobną linię.