

Metodyka programowania

Edytor tekstu.
Komponenty edytora

Edytor tekstu

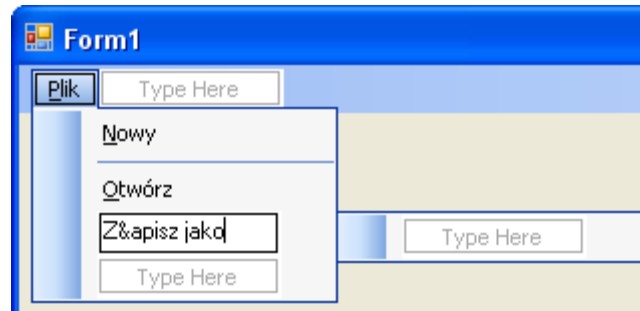
Komponenty edytora

- Menu
MenuStrip
- Pasek narzędzi
ToolStrip albo
ToolStripContainer i kilka pasków *ToolStrip*
- Edytor
TextBox albo
RichTextBox
- Okna dialogowe
FontDialog
ColorDialog
OpenFileDialog
SaveFileDialog

Komponenty

MenuStrip – menu główne

- Automatycznie lokuje się u góry okna (domyślnie właściwość *Dock = Top*)
- Wyposażone w wygodny edytor, umożliwiający tworzenie wielopoziomowego menu bezpośrednio na formularzu

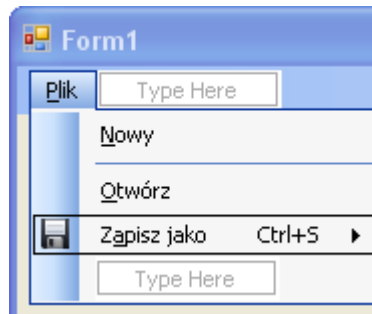


- Wpisanie opisu i potwierdzenie klawiszem <Enter> powoduje utworzenie komponentu *ToolStripMenuItem*;
Nietypowe nazwy: ***opisElementuToolStripMenuItem*** zamiast ***ToolStripMenuItem15***

Komponenty

ToolStripMenuItem – pozycja menu

- Dostępny wyłącznie jako element menu (*MenuStrip* lub *ContextMenuStrip*) – nie ma w Toolboxie Visual Studio



- Właściwości:
 - Name – nazwa w komponencie w programie; niektórzy wolą *saveToolStripMenuItem* zamiast *zapiszToolStripMenuItem*
 - Image – obrazek wyświetlany obok opisu; powinien być, jeżeli polecenie menu dostępne również przez pasek narzędzi, oczywiście wszędzie taki sam

Komponenty

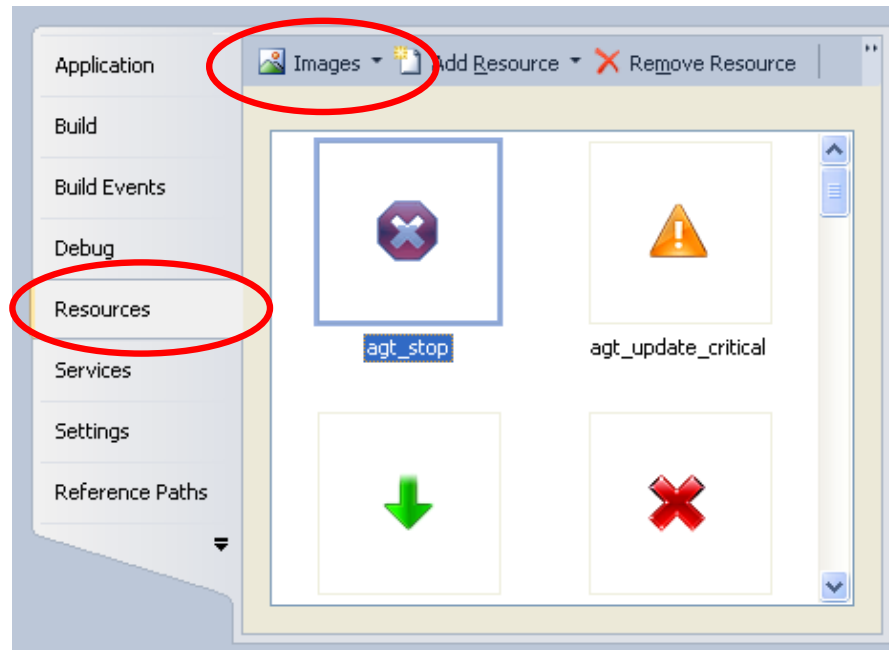
ToolStripMenuItem – pozycja menu

- Właściwości (c.d.)
 - ShortcutKeys – skrót klawiaturowy
(nie mylić z ShortcutKeyDisplayString – opis skrótu)
 - ShortcutKeyDisplayString – właściwy opis pojawia się automatycznie (np. "Ctrl + O"), nie ma potrzeby zmieniać
 - Checked – symbol zaznaczenia
 - CheckOnClick – automatyczne włączanie i wyłączanie symbolu zaznaczenia – przekształca element menu w rodzaj CheckBoxa
(nie należy używać, jeżeli dana opcja jest dostępna również inną drogą, np. przez pasek narzędzi)

Komponenty

Zasoby aplikacji

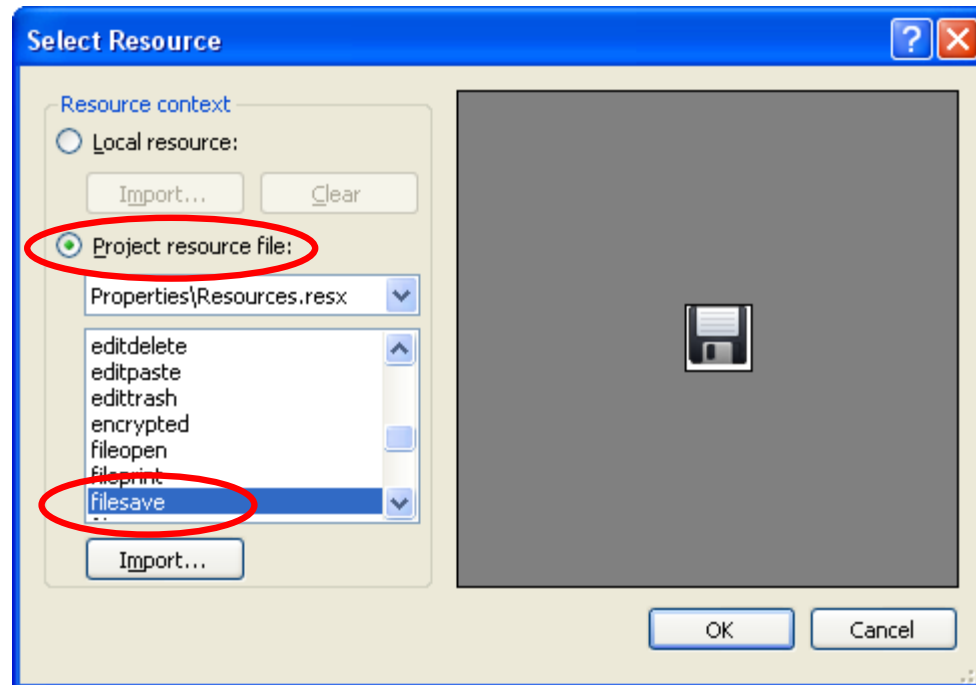
- Dwukrotne kliknięcie folderu *Properties* (właściwości), w oknie *Solution Explorer*, otwiera okno właściwości aplikacji
- Zakładka *Resources* (tj. zasoby) zawiera zasoby dostępne w całej aplikacji; Po zmianie rodzaju zasobów ze *String* na *Images* można dodać obrazki



Komponenty

Zasoby aplikacji

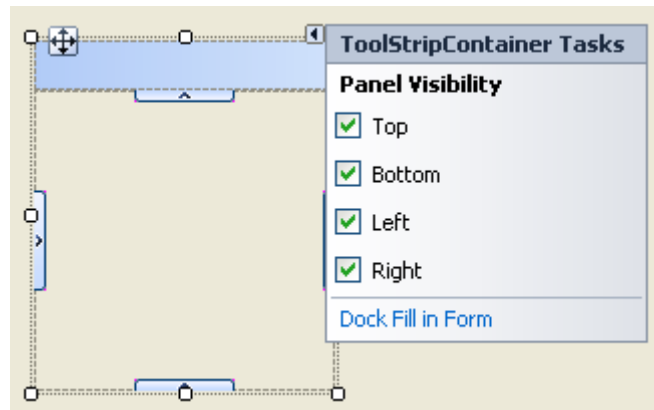
- Umieszczone w zasobach obrazki można umieszczać na wielu komponentach, np. w menu i na pasku narzędzi;
Do wyboru obrazka (z zasobów aplikacji) lub wskazania pliku obrazka służy specjalne okno dialogowe:



Komponenty

ToolStripContainer – kontener na paski narzędzi

- Zwykle zajmuje całe okno (*Dock = Fill*)
- Składa się z czterech paneli ulokowanych przy krawędziach (na paski narzędzi i menu) oraz centralnego panelu

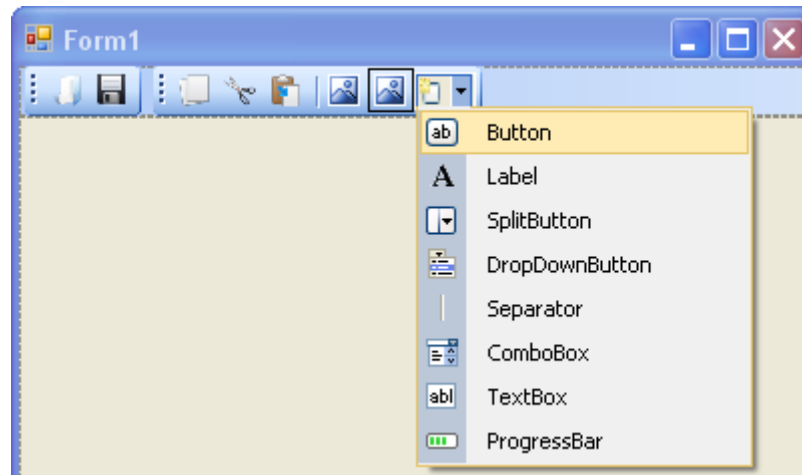


- Menu i paski narzędzi można swobodnie przemieszczać między czynnymi krawędziami kontenera, o ile właściwość *LayoutStyle* jest typu *StackWithOverflow*

Komponenty

ToolStrip – pasek narzędzi

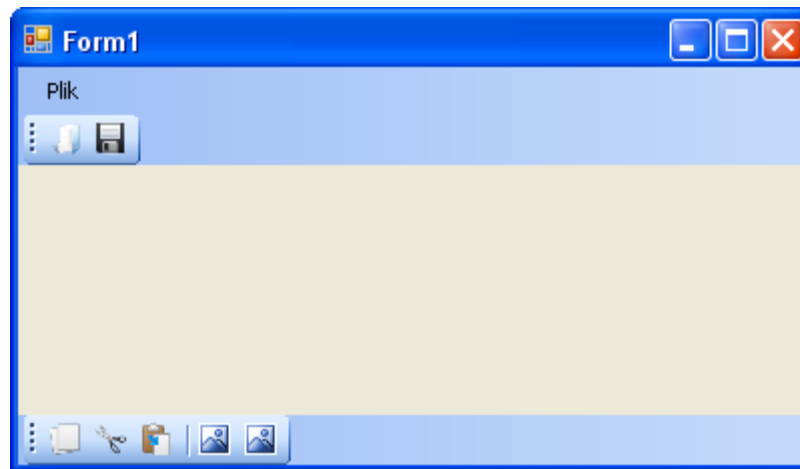
- Automatycznie lokuje się u góry okna (*Dock = Top*), o ile nie jest umieszczony na *ToolStripContainer*
- Wyposażone w wygodny edytor, umożliwiający dodawanie elementów (najczęściej przycisków i separatorów) bezpośrednio na formularzu



Komponenty

ToolStrip – właściwości

- ImageScalingSize – rozmiar obrazków na przyciskach (nie można zmieniać przez przycisk)
- LayoutStyle – jeżeli *StackWithOverflow*, to pasek można przenosić między aktywnymi krawędziami *ToolStripContainer*; Jeżeli *Flow*, to nie można przenieść (brak jest stosownego uchwytu); tak powinno się ustawiać paski umieszczone bezpośrednio na formularzu (bez *ToolStripContainer*)



Komponenty

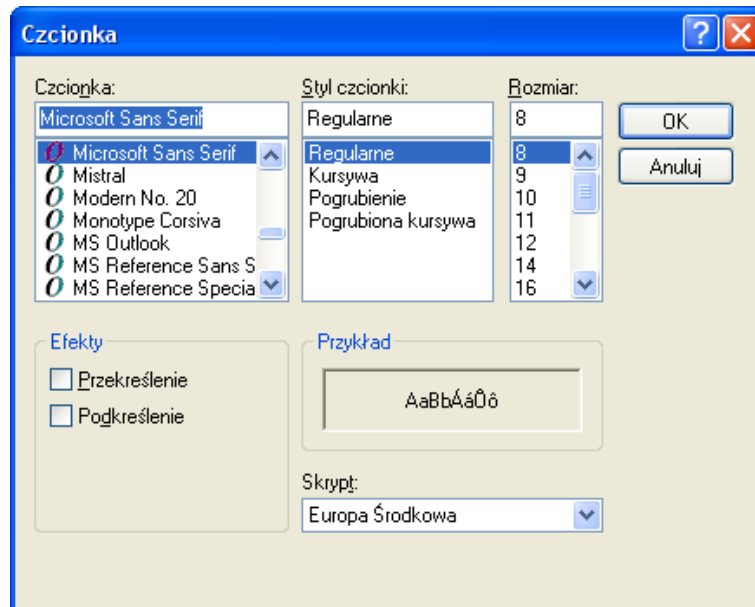
ToolStripButton – przycisk na pasku narzędzi

- Specjalny przycisk, niedostępny w Toolbox
 - Image – obrazek; można korzystać z zasobów aplikacji (jak w przypadku komponentu *ToolStripMenuItem*)
 - ImageTransparentColor – kolor tła na obrazku, który będzie zamieniony na przezroczystość po wyświetleniu
 - AutoToolTip – wyświetlanie podpowiedzi w "dymku" po wskazaniu myszą
 - ToolTipText – treść podpowiedzi w dymku
 - Checked – zaznaczony (tj. trwale wciśnięty) pozwala traktować ToolStripButton jak pole wyboru (Checkbox) lub przycisk radiowy (RadioButton)
 - CheckOnClick – automatycznie zmienia Checked po kliknięciu (nie wystarczy, jeżeli przycisk ma działać jak RadioButton!)

Komponenty

FontDialog – Okno dialogowe "Czcionka"

- Komponent nie ma widocznej reprezentacji na oknie (podobnie pozostałe dialogi)
- Metoda *ShowDialog* wyświetla standardowe okno dialogowe systemu Windows zmiany czcionki;
Właściwości komponentu określają jego wygląd i zachowanie



Komponenty

FontDialog – właściwości

- Różne opcje
 - ShowApply – wyświetla przycisk "Zastosuj";
trzeba obsługiwać zdarzenie OnApply
 - ShowEffects – wyświetla style (Pogrubienie, Pochylenie, ...)
 - ShowColor – wyświetla komponent do zmiany koloru czcionki
Domyślnie wyłączone, ponieważ właściwość Font nie obejmuje koloru; jeżeli włączone, to równolegle ze zmianą czcionki trzeba zmienić kolor
 - FontMustExist – pozwala wybrać tylko istniejącą czcionkę

Komponenty

FontDialog– właściwości

- **Font** – cechy czionki:
 - proponowane (przed wyświetleniem okna),
 - wybrane przez użytkownika (po zamknięciu okna)
- **Color** – kolor czionki:
 - proponowany (przed wyświetleniem okna),
 - wybrany przez użytkownika (po zamknięciu okna)

Komponenty

FontDialog – sposób użycia

- Najpierw należy przenieść cechy czcionki (i ew. koloru) z komponentu do okna dialogowego, następnie wyświetlić okno i koniecznie sprawdzić rezultat; Jeżeli jest DialogResult.OK, przenieść cechy czcionki z okna dialogowego do komponentu. (uwaga – zmieniona nazwa komponentu, na *fontDlg*)

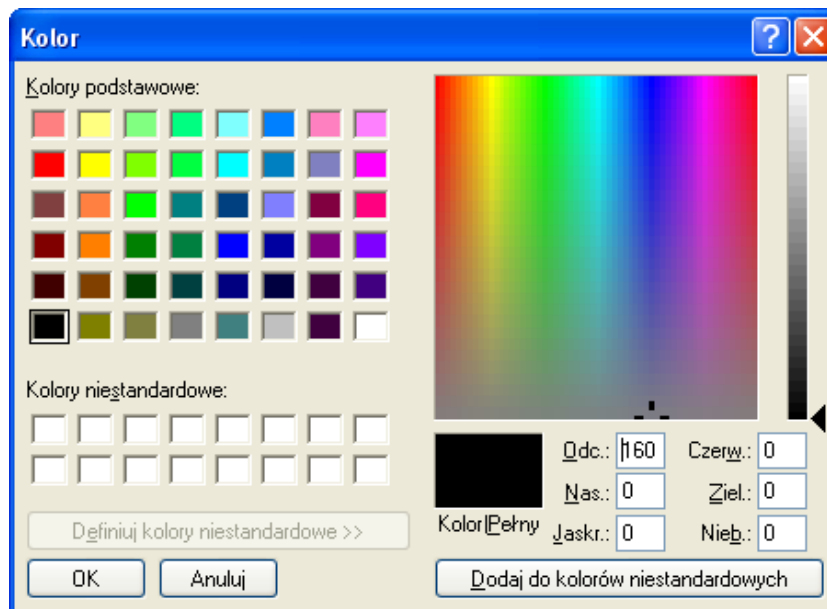
```
fontDlg.Font = textBox.Font;  
if (fontDlg.ShowDialog() == DialogResult.OK)  
{  
    textBox.Font = fontDlg.Font;  
}
```

Uwaga! W RichTextBox należy użyć właściwości SelectionFont oraz SelectionColor

Komponenty

ColorDialog – Okno dialogowe "Kolor"

- Komponent nie ma widocznej reprezentacji na oknie (jak pozostałe dialogi)
- Metoda *ShowDialog* wyświetla standardowe okno dialogowe systemu Windows zmiany koloru;
Właściwości komponentu określają jego wygląd i zachowanie



Komponenty

ColorDialog – właściwości

- Różne opcje
 - FullOpen – wyświetla całe okno
 - AllowFullOpen – zezwala na odsłonięcie całego okna
 - SolidColorOnly – pozwala wybrać tylko standardowe kolory (jeżeli *true*, to AllowFullOpen powinno być wyłączone)
- **Color** – kolor czionki:
 - proponowany (przed wyświetleniem okna),
 - wybrany przez użytkownika (po zamknięciu okna)

Komponenty

ColorDialog – sposób użycia

- Najpierw należy przenieść koloru z komponentu do okna dialogowego, następnie wyświetlić okno i koniecznie sprawdzić rezultat; Jeżeli jest DialogResult.OK, przenieść kolor z okna dialogowego do komponentu.

(uwaga – zmieniona nazwa komponentu, na *colorDlg*)

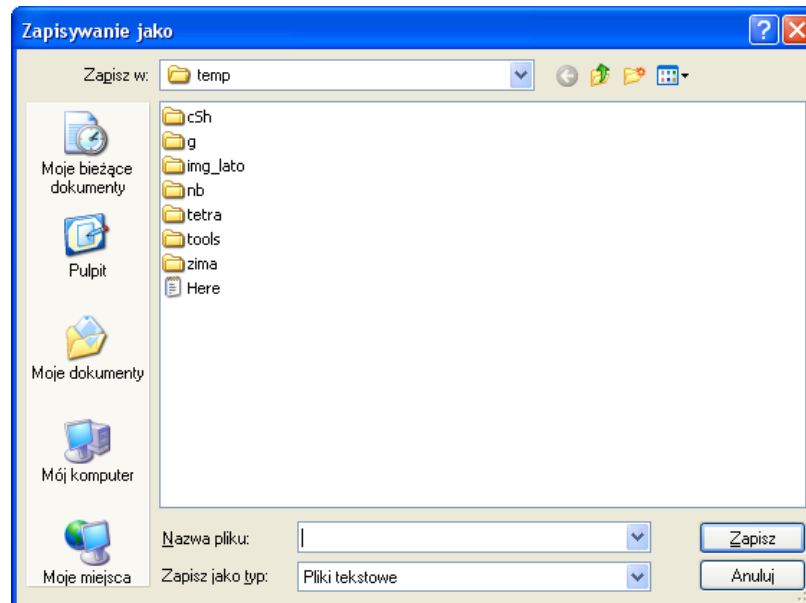
```
colorDlg.Color = textBox.ForeColor;  
if (fontDlg.ShowDialog() == DialogResult.OK)  
{  
    textBox.ForeColor = colorDlg.Color;  
}
```

Uwaga! W podobny sposób można zmienić kolor tła (BackColor)

Komponenty

SaveFileDialog – Okno dialogowe "Zapisywanie"

- Komponent nie ma widocznej reprezentacji na oknie (jak pozostałe dialogi)
- Metoda *ShowDialog* wyświetla standardowe okno dialogowe systemu Windows zapisywania pliku;
Właściwości komponentu określają jego wygląd i zachowanie



Komponenty

SaveFileDialog – właściwości

- Title – tekst wyświetlany na pasku tytułu okna dialogowego
- Różne opcje
 - OverwritePrompt – jeżeli plik istnieje, to wyświetla pytanie czy ma zostać nadpisany
 - CreatePrompt – jeżeli plik nie istnieje, pyta czy utworzyć
 - CheckFileExists – można wybrać tylko istniejący plik
 - CheckPathExists – można wybrać tylko istniejący folder
 - SupportMultiDottedExtensions – obsługa rozszerzeń w rodzaju .exe.manifest
 - ValidateNames – nie można podać nazwy pliku z zabronionymi znakami (np. ":")

Komponenty

SaveFileDialog – właściwości

- **Filter** – filtr wyświetlanych plików
Opis_1|Maska_1|Opis_2|Maska_2 ...
Pliki tekstowe|*.txt|Pliki źródłowe C#|*.cs|Wszystkie pliki|*.*
- **AddExtension** – czy do nazwy ma być dodawane rozszerzenie;
jeżeli tak, to dodawane jest rozszerzenie zależne od filtru
- **DefaultExt** – rozszerzenie domyślne (jeżeli nie wynika z filtru)
- **InitialDirectory** – katalog początkowy (jeżeli nie podany –
katalog ostatnio używany do zapisywania pliku)
- **FileName** – kwalifikowana (z pełną ścieżką) nazwa pliku:
 - proponowana (przed wyświetleniem okna),
 - wybrana przez użytkownika (po zamknięciu okna)

Komponenty

SaveFileDialog– sposób użycia

- Należy wyświetlić okno i koniecznie sprawdzić rezultat; Jeżeli rezultat jest DialogResult.OK, zapisać plik (uwaga – zmieniona nazwa komponentu, na *saveDlg*)

```
DialogResult res;  
res = saveDlg.ShowDialog();  
if (res == DialogResult.OK)  
{  
    // zapisanie pliku  
}
```

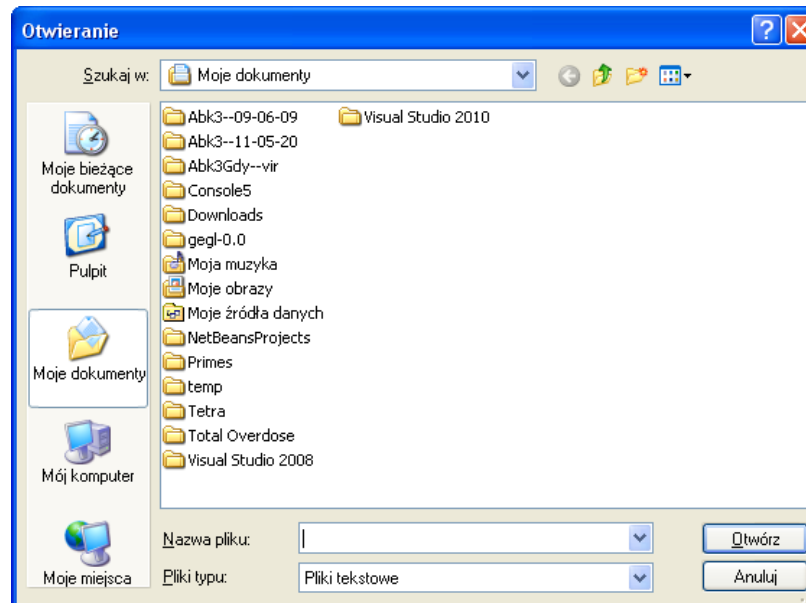
- To samo w skróconym zapisie:

```
if (saveDlg.ShowDialog() == DialogResult.OK)  
{  
    // zapisanie pliku  
}
```

Komponenty

OpenFileDialog – Okno dialogowe "Otwieranie"

- Komponent nie ma widocznej reprezentacji na oknie (jak pozostałe dialogi)
- Metoda *ShowDialog* wyświetla standardowe okno dialogowe systemu Windows otwierania pliku;
Właściwości komponentu określają jego wygląd i zachowanie



Komponenty

OpenFileDialog – właściwości

- Title – tekst wyświetlany na pasku tytułu okna dialogowego
- Różne opcje
 - CheckFileExists – można wybrać tylko istniejący plik
 - CheckPathExists – można wybrać tylko istniejący folder
 - Multiselect – możliwość wskazania kilku plików
 - ShowReadOnly – pokazuje pole "Otwórz tylko do odczytu"
 - ReadOnlyChecked – stan ww. pola
 - SupportMultiDottedExtensions – obsługa rozszerzeń w rodzaju .exe.manifest
 - ValidateNames – nie można podać nazwy pliku z zabronionymi znakami (np. ":")

Komponenty

OpenFileDialog – właściwości

- **Filter** – filtr wyświetlanych plików
Można dać po kilka masek w każdej kategorii
Pliki tekstowe|*.txt|Pliki www|*.htm; *.html|Wszystkie pliki|*.*
- **FilterIndex** – numer początkowo używanego filtru
(numeracja zaczyna się od 1, nie od 0)
- **AddExtension** – czy do nazwy ma być dodawane rozszerzenie;
- **InitialDirectory** – katalog początkowy (jeżeli nie podany – katalog ostatnio używany do zapisywania pliku)
- **FileName** – kwalifikowana (z pełną ścieżką) nazwa pliku:
 - proponowana (przed wyświetleniem okna),
 - wybrana przez użytkownika (po zamknięciu okna)

Komponenty

OpenFileDialog– sposób użycia

- Należy wyświetlić okno i koniecznie sprawdzić rezultat; Jeżeli rezultat jest DialogResult.OK, zapisać plik (uwaga – zmieniona nazwa komponentu, na *openDlg*)

```
DialogResult res;  
res = openDlg.ShowDialog();  
if (res == DialogResult.OK)  
{  
    // odczytanie pliku  
}
```

- To samo w skróconym zapisie:

```
if (openDlg.ShowDialog() == DialogResult.OK)  
{  
    // odczytanie pliku  
}
```

Komponenty

TextBox – pole edycyjne

- Prosty edytor, odpowiednik systemowego Notatnika



- Właściwości:
 - Dock – dostosowanie rozmiaru i położenia (przyda się *Fill*)
 - Text – wyświetlany tekst, dostępny do edycji
 - Multiline – czy może zawierać więcej niż jedną linię tekstu
 - MaxLength – maksymalna długość tekstu
 - WordWrap – zawijanie tekstu
 - ScrollBars – paski przewijania (domyślnie brak)
 - TextAlign – wyrównanie tekstu
 - Font – cechy czcionki
 - ForeColor – kolor tekstu
 - BackColor – kolor tła
 - **Modified** – tekst został zmodyfikowany; Uwaga! Automatyczna zmiana tylko false -> true, odwrotnie "ręcznie"

Komponenty

TextBox – czcionka

- Font – cechy czcionki, której używa komponent: Name – krój (np. „Bookman Old Style”), Size – rozmiar w punktach, itd.

Poszczególne cechy czcionki (np. rozmiar) są dostępne programowo tylko do odczytu. Aby zmienić tylko jedną cechę, należy utworzyć nowy obiekt Font, kopiując pozostałe cechy:

```
Font cf = textBox1.Font;  
Font nf = new Font(  
    cf.FontFamily, 15.0f, cf.Style);  
textBox1.Font = nf;
```

Komponenty

TextBox – czcionka (c.d.)

- Najwięcej kłopotu sprawia właściwość *Style* klasy Font; Może przyjmować wartości: Regular, Bold, Italic, Underline oraz Strikeout.

Jest to typ wyliczeniowy z atrybutem FlagsAttribute, przez co na jego wartościach można wykonywać operacje binarne OR ("|") oraz XOR ("^"), np.:

```
Font cf = textBox1.Font;  
Font nf = new Font(cf.FontFamily, cf.Size,  
    cf.Style ^ FontStyle.Bold);  
textBox1.Font = nf;
```

spowoduje "przełączenie" (włączenie, jeżeli wyłączone lub odwrotnie) pogrubienia

Komponenty

TextBox – pole edycyjne

- Zdarzenia:

- TextChanged – informuje o każdej zmianie tekstu
(nie można zmieniać właściwości Text – nieskończona pętla!)
- [-] – brak jest zdarzenia informującego o zmianie zaznaczenia lub położenia kursora;

Zamiast tego można obsłużyć wszystkie zdarzenia, których pośrednim skutkiem jest ww. zmiana, tj.:

- OnKeyDown,
- OnKeyPress,
- OnMouseDown,
- OnMouseUp

Komponenty

TextBox – pole edycyjne

- Metody

- Niemal wszystkie podstawowe operacje wykonują łatwe do użycia metody:

- Cut()
 - Copy()
 - Paste()
 - SelectAll()
 - Undo() – cofa tylko jedną, ostatnią czynność
 - Redo()
 - Clear()
 - Nie ma metod do odczytu i zapisu plików

Komponenty

TextBox – odczyt i zapis plików

- W .Net jest kilka klas, które można do tego użyć.
Np. klasa File dostarcza szereg statycznych metod, m.in. do tworzenia, kopiowania, usuwania, odczytywania i zapisu plików.

- String ReadAllText (String path)

Odczytuje zawartość pliku tekstowego i dostarcza w postaci pojedynczego łańcucha znaków; *path* to kwalifikowana nazwa pliku (tj. ze ścieżką dostępu), jaką dostarcza np. okno dialogowe OpenFileDialog

```
textBox.Text = File.ReadAllText(openDlg.FileName);
```

- void WriteAllText(String path, String contents)

Zapisuje do pliku (*path*) wielolinijkową zawartość (*contents*)

```
File.WriteAllText(saveDlg.FileName, textBox.Text);
```

Komponenty

RichTextBox – pole edycyjne RTF

- Edytor z obsługą formatu RTF, odpowiednik WordPada; posiada niemal wszystkie właściwości, metody i zdarzenia komponentu TextBox oraz kilka dodatkowych
- Właściwości:
 - Text – wyświetlany tekst, dostępny do edycji
 - Multiline – czy może zawierać więcej niż jedną linię tekstu
 - MaxLength – maksymalna długość tekstu (2.147.483.647 zamiast 32.767 w TextBox)
 - WordWrap – zawijanie tekstu
 - ScrollBars – paski przewijania (domyślnie brak)
- Brakuje kilku właściwości TextBox (TextAlign, AcceptsReturn), nie ma też menu kontekstowego

Komponenty

RichTextBox – pole edycyjne RTF

- Właściwości specyficzne (niedostępne w TextBox) określają cechy zaznaczonego fragmentu lub tzw. "punktu wstawiania"

Cechy akapitów:

- SelectionAlignment – wyrównanie tekstu
- SelectionBullet – wypunktowanie akapitów
- SelectionIndent, SelectionHangingIndent – lewy margines
- SelectionRightIndent – prawy margines

Cechy tekstu:

- SelectionBackColor – kolor tła
- SelectionColor – kolor tekstu
- SelectionFont – czcionka (krój, wielkość, styl)

Komponenty

RichTextBox – pole edycyjne RTF

- Zdarzenia:
 - TextChanged – informuje o każdej zmianie tekstu
(nie można zmieniać właściwości Text – nieskończona pętla!)
 - SelectionChanged – informuje o zmianie obszaru zaznaczenia lub przesunięciu kursora; można wykorzystać do "sprzężenia zwrotnego" – zmiany stanu komponentów zależnie od cech akapitu lub tekstu

Komponenty

RichTextBox – pole edycyjne RTF

- Metody:
 - Wszystkie znane z TextBox
 - Cut()
 - Copy()
 - Paste()
 - SelectAll()
 - Undo() – obsługuje wiele czynności
 - Redo()
 - Clear()
 - Dodatkowo potrafi zapisywać i odczytywać pliki, zarówno w formacie RTF, jak i zwykłe pliki tekstowe
 - SaveFile(String, RichTextBoxStreamType)
 - LoadFile(String, RichTextBoxStreamType)

Komponenty

RichTextBox – odczyt pliku

- Metody:
 - LoadFile(String) – pliki RTF
 - LoadFile(String, RichTextBoxStreamType) – dowolne pliki
- RichTextBoxStreamType – typ wyliczeniowy, określający sposób potraktowania pliku:
 - RichText – format RTF
 - PlainText – zwykły tekst
- Próba wczytania pliku nie-RTF jako RTF kończy się błędem; do sprawdzenia można wykorzystać:
 - klasę Path (udostępnia statyczne metody)
 - obiekt FileInfo (udostępnia metody i właściwości instancji)

Komponenty

Path – analiza kwalifikowanej nazwy pliku

- Metody (statyczne!):
 - GetExtension (String) – rozszerzenie (np. ".rtf", ".txt")
 - GetName (String) – nazwa z rozszerzeniem (np. "Plik.rtf")

FileInfo – informacja o pliku

- Konstruktor wymaga podania kwalifikowanej nazwy:

```
FileInfo f = new FileInfo (openDlg.FileName);  
if (f.Extension == ".rtf")  
    // tu obsługa plików RTF
```

- Właściwości
 - Name, - Extension, - Attributes, - Exists, - Length, ...
- Metody
 - CopyTo, - MoveTo, - Delete,

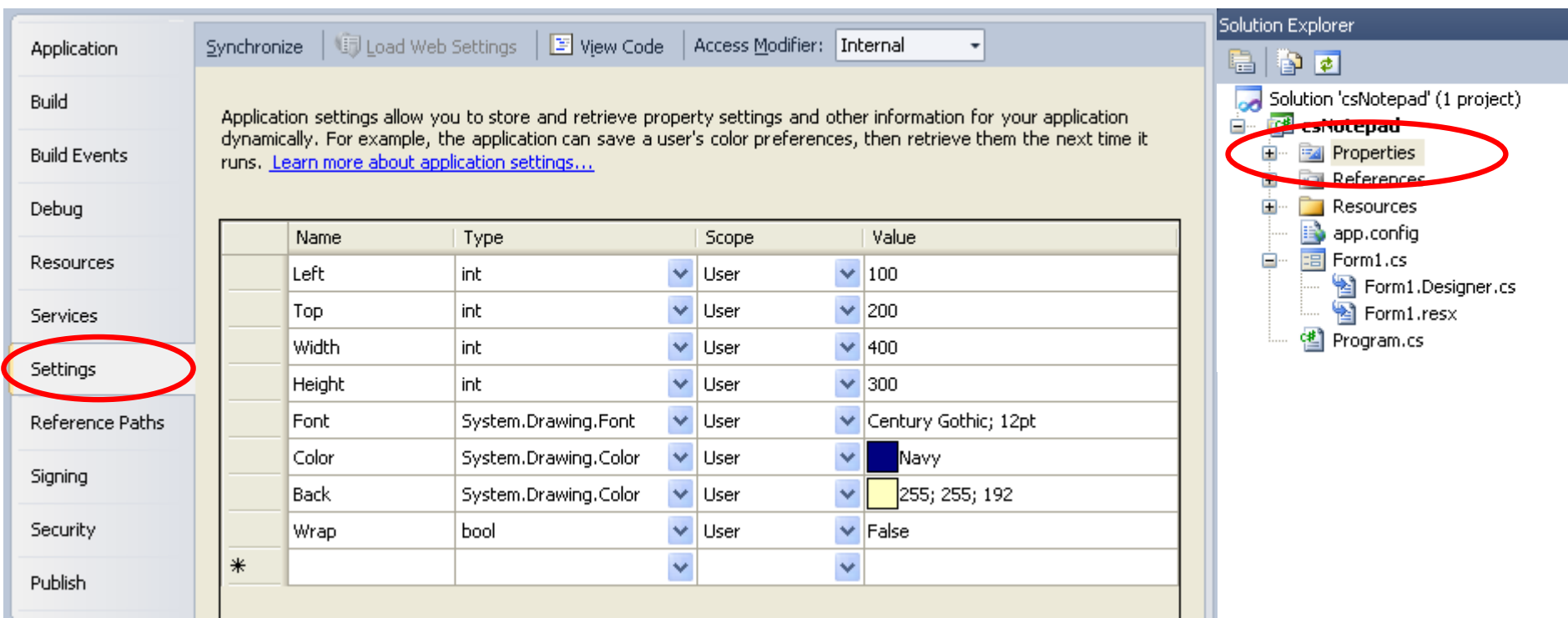
Ustawienia

Ustawienia (ang. settings)

- Umożliwia dynamiczne zapisywanie i odtwarzanie dowolnych ustawień aplikacji oraz preferencji użytkownika
- Najczęściej stosowane do (m.in.):
 - rozmiar i położenie okna
 - opcje (np. zawijanie wierszy, widok pasków narzędzi, ...)
 - czcionki i kolory
- VS ma wbudowany mechanizm zarządzania ustawieniami
 - nie trzeba sięgać do bardziej złożonych i wymagających mechanizmów, takich jak pliki INI lub rejestr Windows

Ustawienia

Zdefiniowanie nazw, typów i domyślnych wartości



The screenshot displays the Visual Studio interface. On the left, the 'Settings' menu item is circled in red. The main window shows the 'Application Settings' dialog for a project named 'csNotepad'. The 'Access Modifier' is set to 'Internal'. The dialog contains a table of settings with columns: Name, Type, Scope, and Value. The settings include Left, Top, Width, Height, Font, Color, Back, Wrap, and an asterisk (*) for additional settings. The 'Color' and 'Back' settings are highlighted with color swatches. On the right, the 'Solution Explorer' shows the project structure, with the 'csNotepad' project and its sub-items (Properties, References, Resources, app.config, Form1.cs, Form1.Designer.cs, Form1.resx, Program.cs) circled in red.

Application Settings allow you to store and retrieve property settings and other information for your application dynamically. For example, the application can save a user's color preferences, then retrieve them the next time it runs. [Learn more about application settings...](#)

Name	Type	Scope	Value
Left	int	User	100
Top	int	User	200
Width	int	User	400
Height	int	User	300
Font	System.Drawing.Font	User	Century Gothic; 12pt
Color	System.Drawing.Color	User	Navy
Back	System.Drawing.Color	User	255; 255; 192
Wrap	bool	User	False
*			

Solution Explorer: Solution 'csNotepad' (1 project)

- csNotepad
 - Properties
 - References
 - Resources
 - app.config
 - Form1.cs
 - Form1.Designer.cs
 - Form1.resx
 - Program.cs

Ustawienia

Odczytanie i zastosowanie ustawień

- W tym celu najlepiej wykorzystać konstruktor klasy formularza (znajduje się na początku definicji klasy)

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

        this.Top = Properties.Settings.Default.Top;
        this.Left = Properties.Settings.Default.Left;
    }
}
```

Ustawienia

Zapisanie ustawień

- Tu najbardziej właściwe jest użycie zdarzenia FormClosed

```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    Properties.Settings.Default.Top = this.Top;
    Properties.Settings.Default.Left = this.Left;

    Properties.Settings.Default.Save();
}
```

Blokada zamknięcia programu

Zamknięcie programu – możliwe przyczyny

- Polecenie Plik/Zakończ
- Kliknięcie x na pasku tytułu
- Skrót klawiaturowy [Alt]+[F4]
- Zamknięcie systemu

Niezależnie od przyczyny, system wysyła do aplikacji komunikat, nakazujący zakończenie działania. Objawia się to wystąpieniem **dwóch** zdarzeń.

Blokada zamknięcia programu

Zamknięcie programu - skutek

- Zdarzenie `OnFormClosing`
(**pytanie** czy aplikacja może zostać zamknięta)
- Zdarzenie `OnFormClosed`
(**informacja** że aplikacja jest zamykana)
 - Jeżeli aplikacja odmawia zamknięcia (w tym celu należy ustawić właściwość *Cancel* argumentu zdarzenia na *true*, jak poniżej), to zdarzenie `OnFormClosed` nie wystąpi,

```
... FormClosing(object sender, FormClosingEventArgs e)
{
    if (...)
    {
        ...
        e.Cancel = true;
    }
}
```

Blokada zamknięcia programu

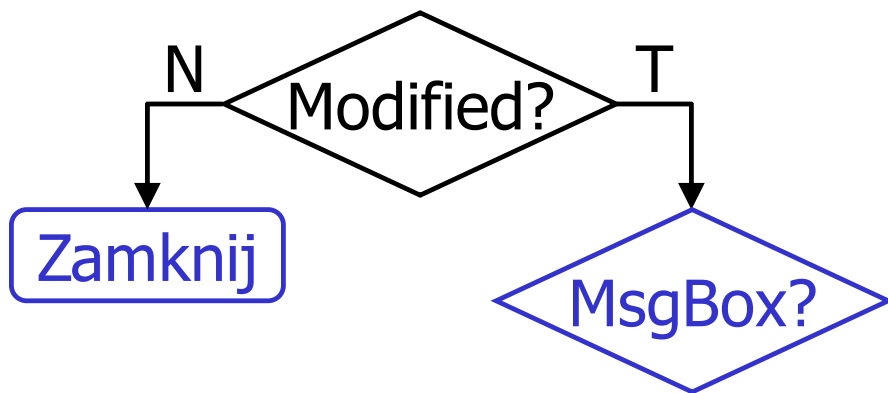
Od czego zależy decyzja programu edytora:

- Czy dokument został zmodyfikowany od czasu utworzenia/otwarcia/zapisania
→właściwość Modified komponentu TextBox (RichTextBox)
- Czy dokument posiada nazwę
→dodatkowa zmienna, ustawiana po wykonaniu poleceń nowy/otwórz/zapisz jako
- Decyzje użytkownika
→okno dialogowe MessageBox
(co robić dalej?)
→okno dialogowe SaveFileDialog
(czy i gdzie zapisać plik?)

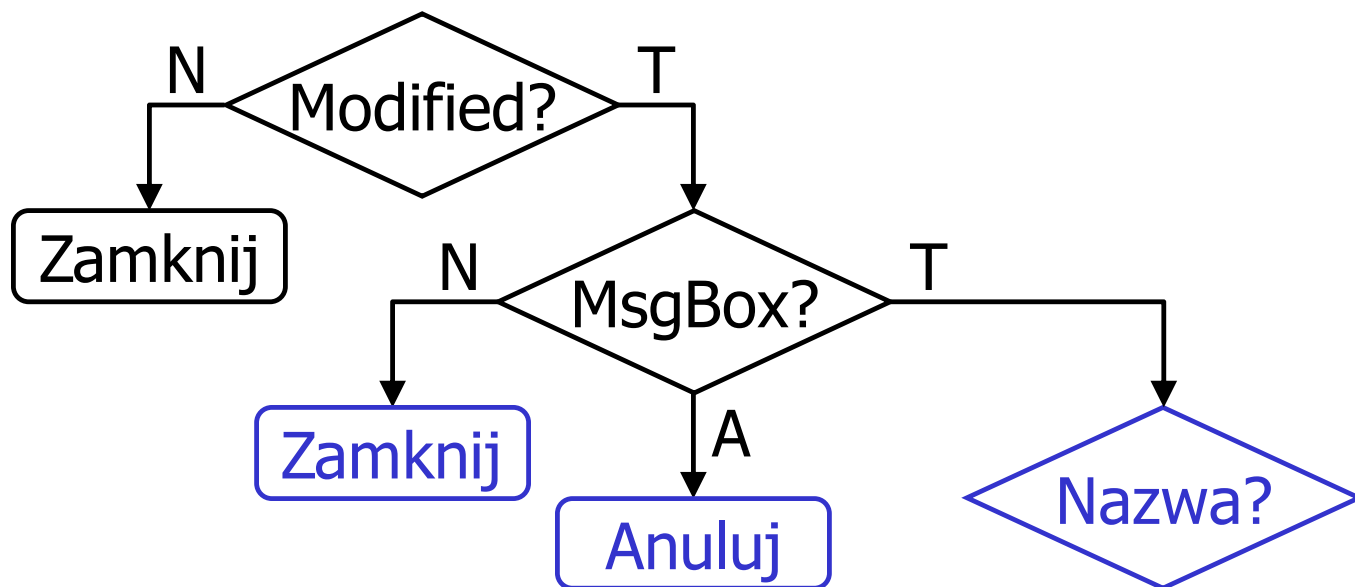
Blokada zamknięcia programu



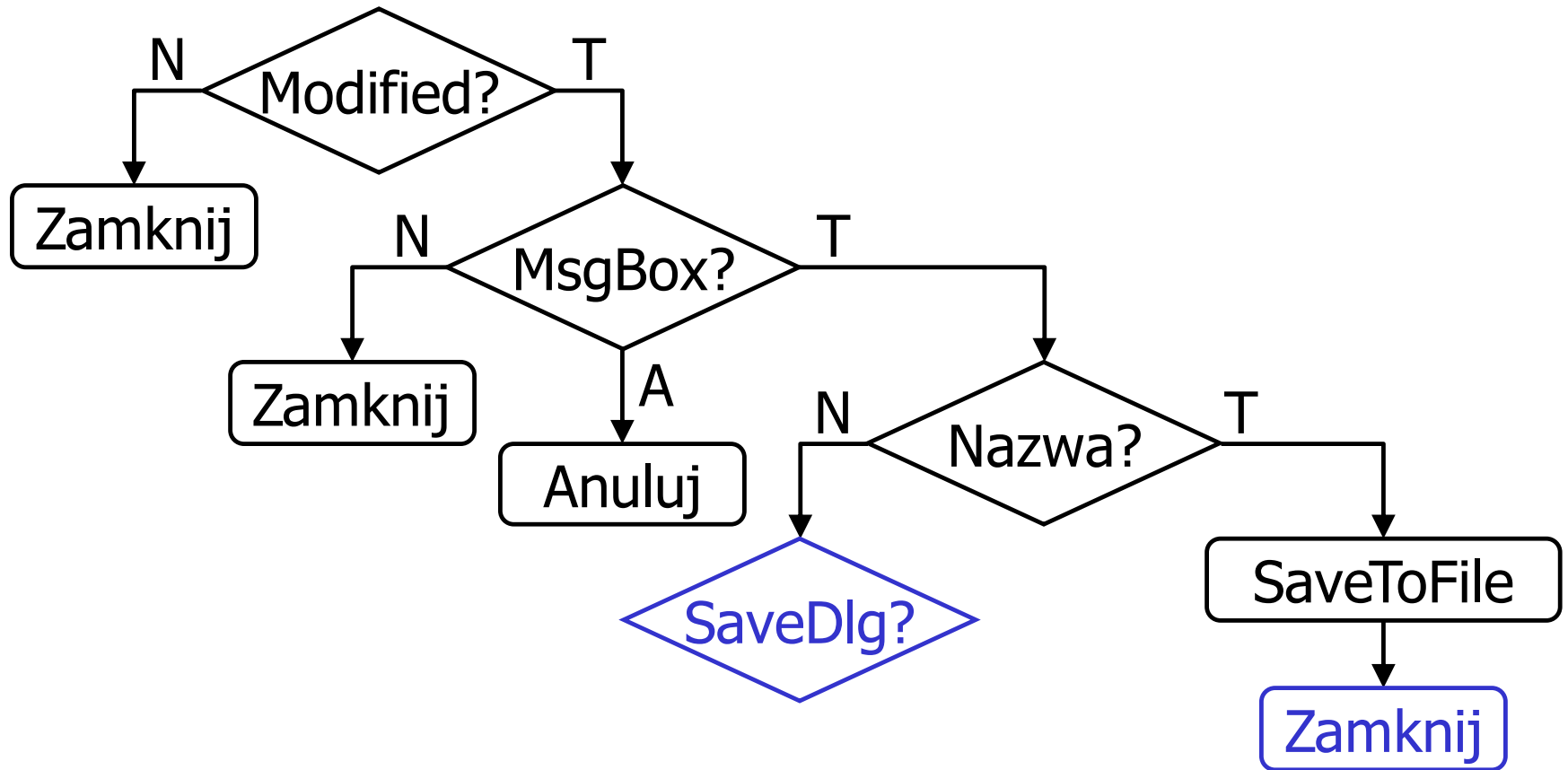
Blokada zamknięcia programu



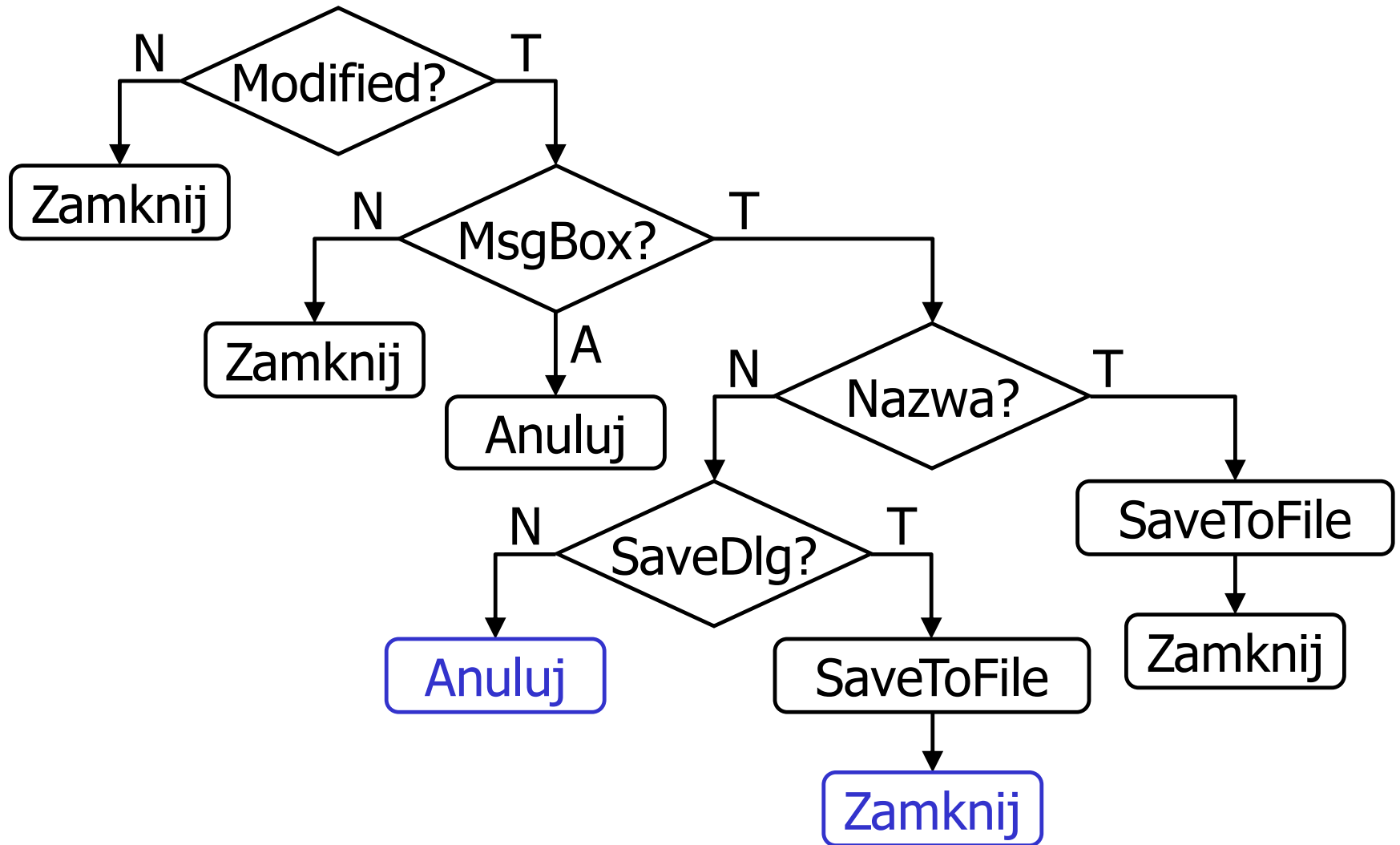
Blokada zamknięcia programu



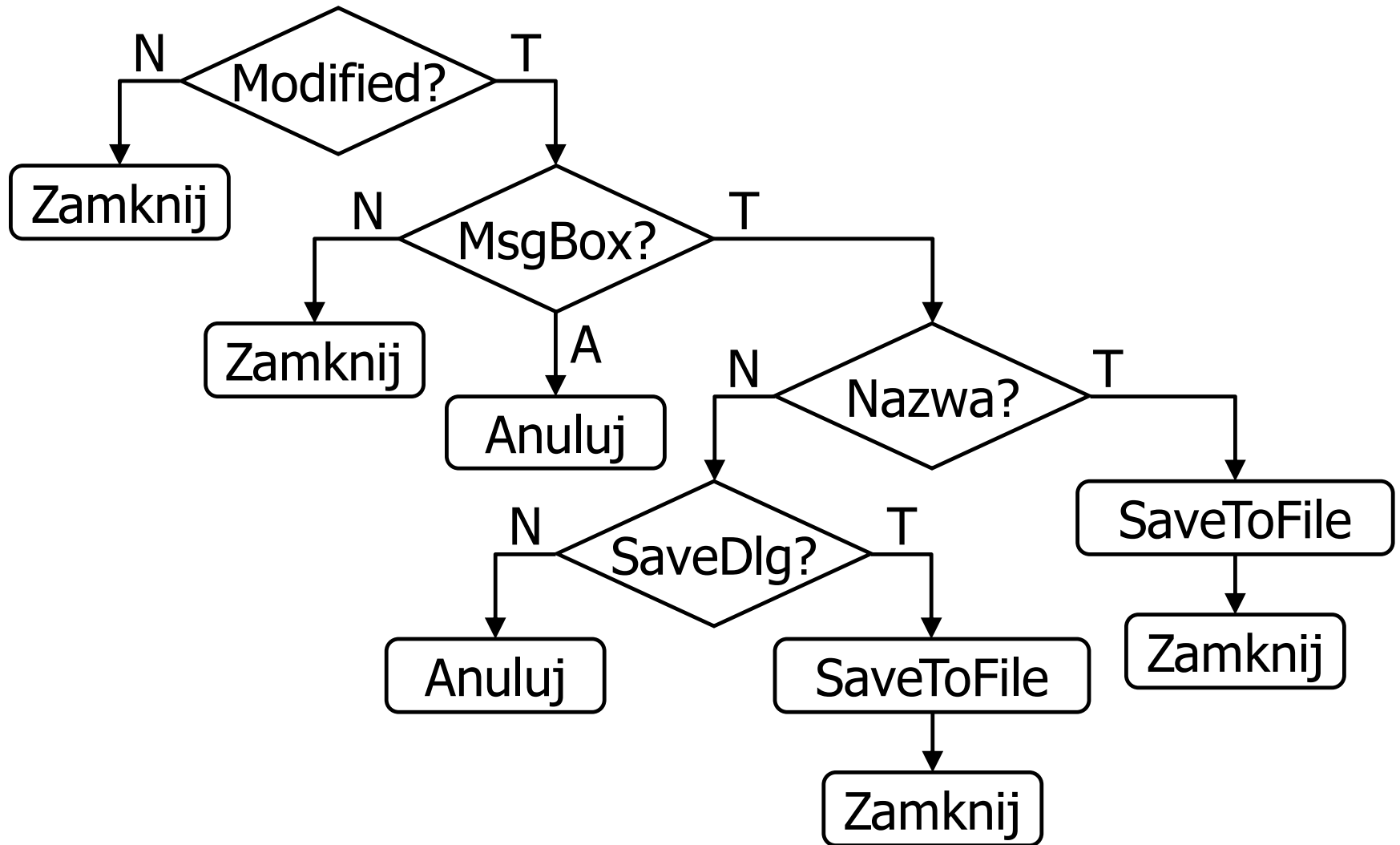
Blokada zamknięcia programu



Blokada zamknięcia programu



Blokada zamknięcia programu



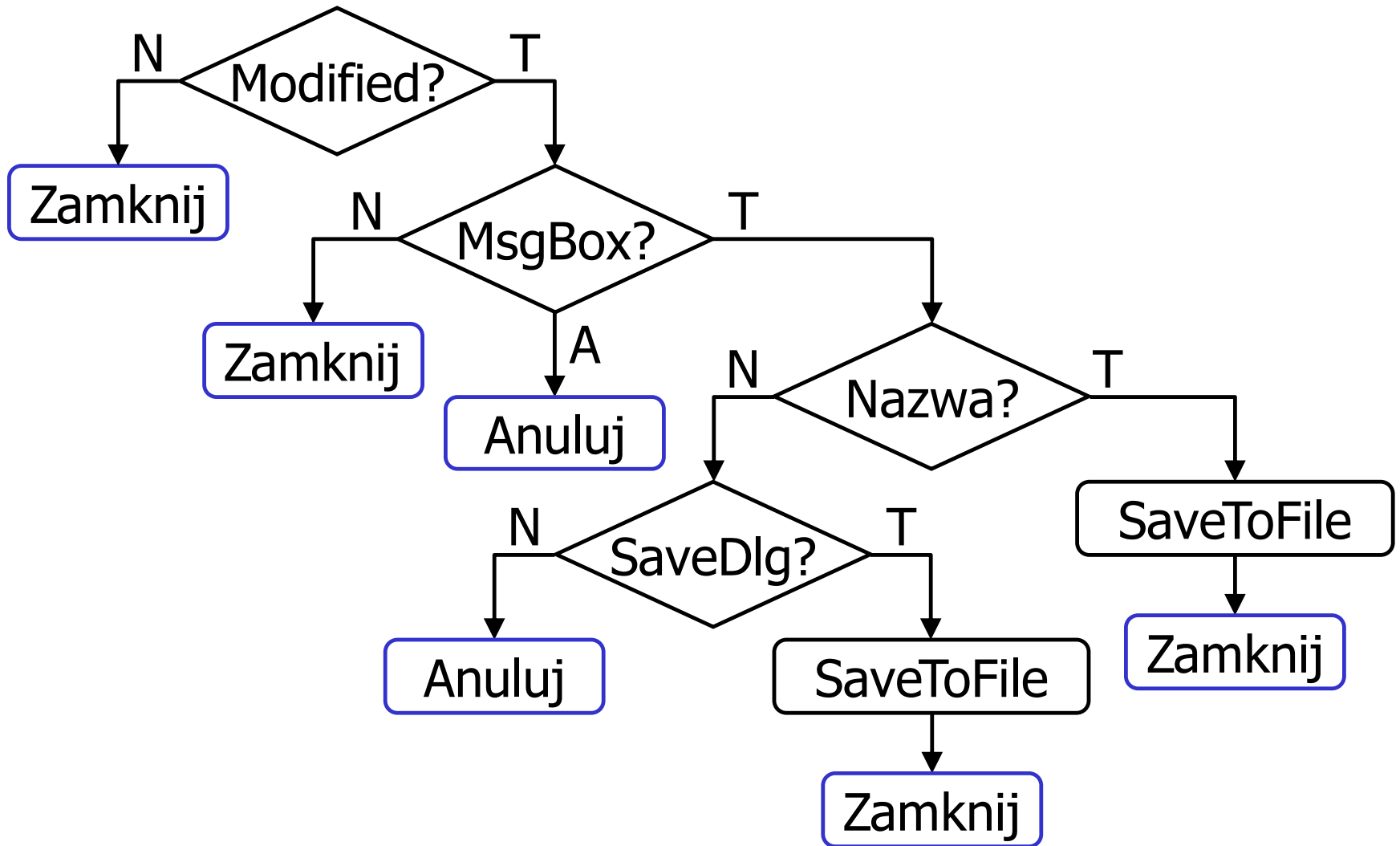
Blokada zamknięcia programu

Trzy stopnie trudności:

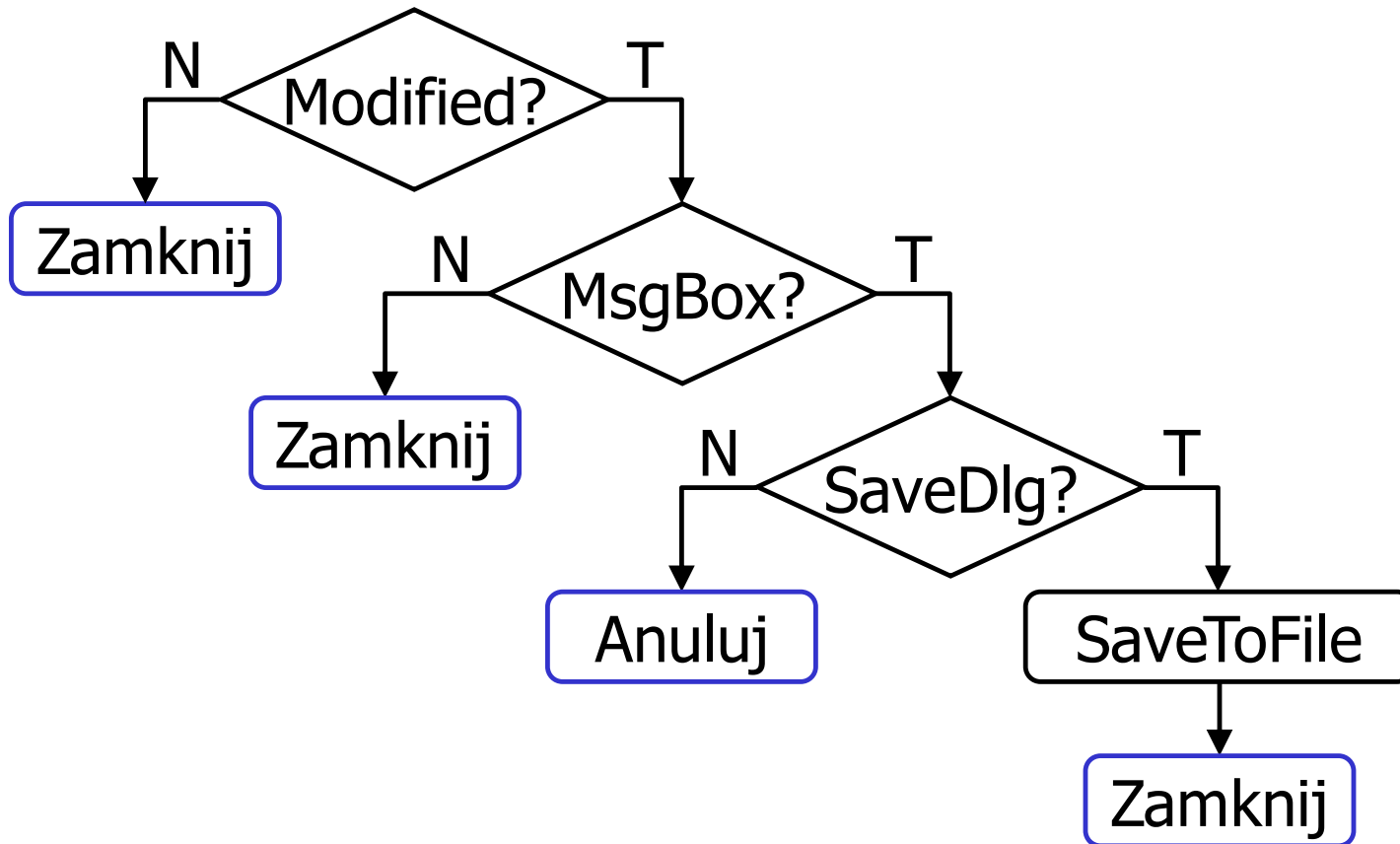
- profesjonalnie (jw.)
- bez zapamiętywania nazwy pliku
- bez zapisywania

Oprócz blokady zamknięcia programu, podobne działania należy realizować przy obsłudze poleceń „Nowy” i „Otwórz”

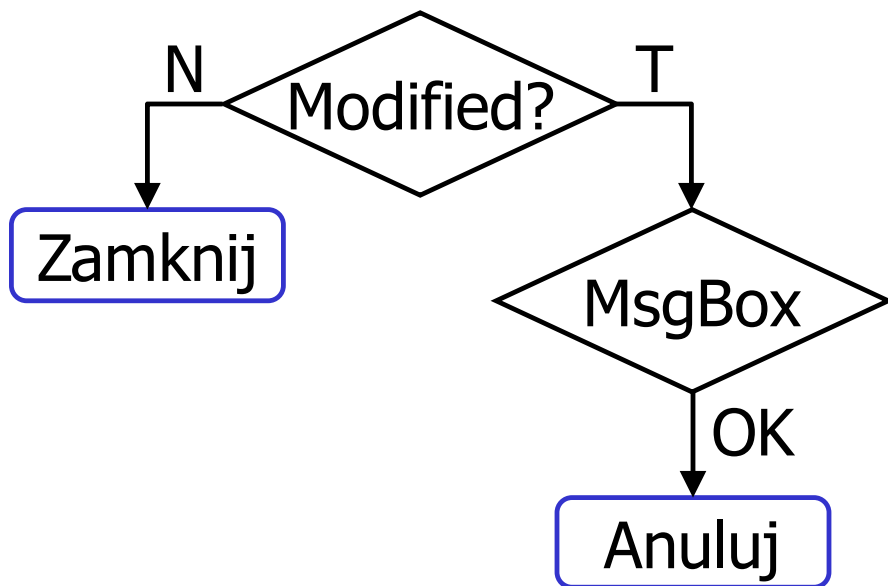
Blokada zamknięcia programu



Blokada zamknięcia programu



Blokada zamknięcia programu



Klasa MessageBox

- Wyświetlanie:

```
DialogResult MessageBox.Show(  
    String text,  
    String caption,  
    MessageBoxButtons buttons,  
    MessageBoxIcon icon,  
    MessageBoxDefaultButton defButton)
```

- Przykład:

```
DialogResult result;  
result = MessageBox.Show(  
    "Czy chcesz zapisać zmiany", "Edytor",  
    MessageBoxButtons.YesNo,  
    MessageBoxIcon.Exclamation,  
    MessageBoxDefaultButton.Button1);
```

Klasa MessageBox

- MessageBoxButtons

Zestaw przycisków, do wyboru:

- Ok, YesNo, YesNoCancel, OkCancel

- MessageBoxIcon

Rodzaj ikony wyświetlanej obok komunikatu, do wyboru:

- Exclamation, Warning – ostrzeżenie ("!"),
- Stop, Error – błąd ("x"),
- Asterisk, Information – informacja ("i")

- MessageBoxDefaultButton

Przycisk domyslny, do wyboru:

- Button1, Button2, Button3