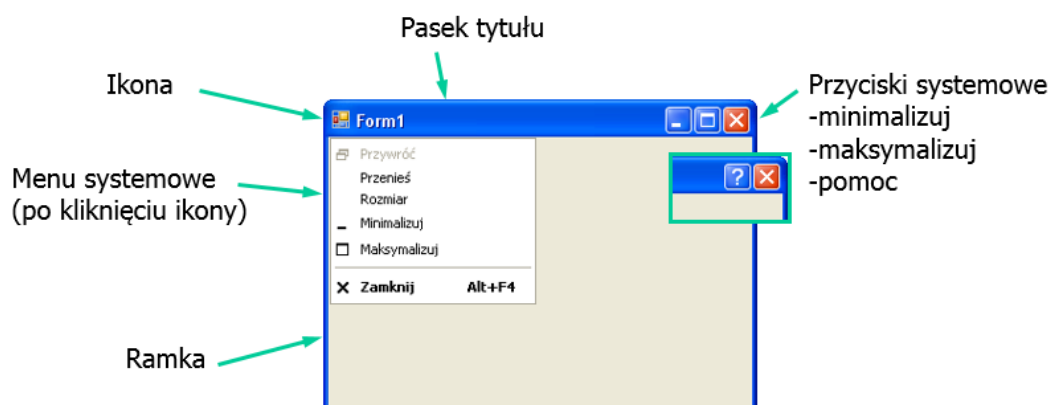


Formularze

Formularz

Formularz (ang. *Form*) to przyszłe okno programu

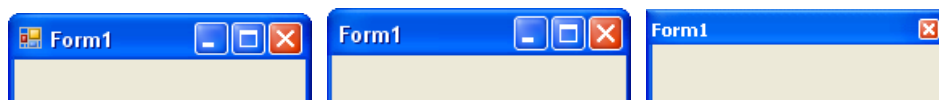
- Nowo utworzony projekt C# typu Windows Forms ma jeden formularz,
- Jest to formularz główny (główne okno) aplikacji: jego zamknięcie powoduje zakończenie działania programu,
- Pasek tytułu (wraz z ikoną i przyciskami) i ramka należą systemu operacyjnego, a nie do aplikacji; Nie można zmienić ich wyglądu ani umieścić tam komponentów,
- Można je dostosować (np. zmienić ikonę lub rodzaj ramki) lub całkowicie wyłączyć



Właściwości formularza

FormBorderStyle (enum System.Windows.Forms.FormBorderStyle)

- Rodzaj ramki (kolor i grubość – w ustawieniach systemu!)
- Możliwe ustawienia:
 - **Sizable** – standardowe
 - **Fixed3D** – stały rozmiar
 - **FixedSingle** – stały rozmiar, ramka bez "3D"
 - **FixedDialog** – typowe dialogowe
 - **SizableToolWindow** – narzędziowe
 - **FixedToolWindow** – narzędziowe, stały rozmiar
 - **None** – brak ramki



Pasek tytułu i ramka różnych okien: Sizable, FixedDialog i FixedToolWindow

ControlBox (Boolean)

- Czy okno ma ikonę z menu systemowym i kontrolki (minimalizuj, maksymalizuj, zamknij)

Icon

- Ikona (.ico) wyświetlana na pasku tytułu i pasku zadań



Pasek tytułu ze standardową i zmienioną ikoną

HelpButton (Boolean)

- Czy wyświetlany jest przycisk pomocy (działa pod warunkiem, że nie ma minimalizuj, maksymalizuj)

MinimizeBox (Boolean)

MaximizeBox (Boolean)

- Czy mają być dostępne na pasku tytułu przyciski minimalizacji oraz maksymalizacji okna (jeżeli nie, to nie działa też klikanie w pasek tytułu)



Pasek tytułu z różnymi ustawieniami MinimizeBox, MaximizeBox i HelpButton

ShowIcon (Boolean)

- Czy okno ma ikonę z menu systemowym (nie da się włączyć np. dla ToolWindow)

ShowInTaskBar (Boolean)

- Czy okno ma reprezentujący je przycisk na pasku zadań

TopMost

- Zawsze na wierzchu, nawet jeżeli nie jest aktywne (jak np. Menedżer Zadań)

Opacity

- Stopień przezroczystości okna (można wpisywać w procentach)

StartPosition (enum System.Windows.Forms.FormStartPosition)

- Początkowe położenie okna
- Możliwe ustawienia:
 - Manual – określone przez właściwość Location
 - CenterScreen – środek okna
 - **WindowsDefaultLocation** – domyślne systemu
 - WindowsDefaultBounds – domyślne położenie i rozmiar
 - CenterParent – środek okna rodzica (tylko MDI)
- Alternatywnie można zapisywać położenie i wielkość okna w chwili zamykania aplikacji i odtwarzać te cechy przy następnym uruchomieniu

Location (Point), **Top**, **Left** (Int32)

- Położenie - współrzędne lewego górnego narożnika względem ekranu

Size (Point), **Width**, **Height** (Int32)

- Rozmiar

MinimumSize (Size),

MaximumSize (Size)

- Minimalny i maksymalny rozmiar okna;

Font

- Cechy czcionki, której używa komponent:
 - Name – krój (np. „Bookman Old Style”)
 - Size – rozmiar w punktach
 - Bold, Italic – pogrubienie, kursywa

BackColor

- Kolor tła

ForeColor

- Kolor tekstu

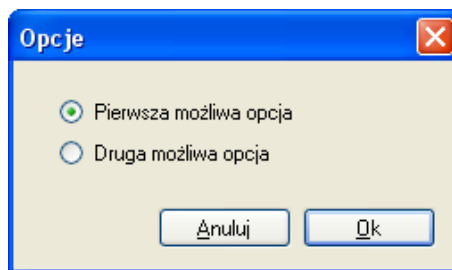
Cechy *Font*, *BackColor* i *ForeColor* są dziedziczone przez komponenty umieszczane na formularzu (w różnym stopniu przez różne rodzaje komponentów)

AcceptButton

- Przycisk "klikany" przez naciśnięcie <Enter> tzw. domyślny przycisk okna, w oknach dialogowych przycisk "Ok"

CancelButton

- Przycisk "klikany" przez naciśnięcie <Esc>, w oknach dialogowych przycisk "Anuluj"



Okno dialogowe z przyciskami *CancelButton* (Anuluj) i *AcceptButton* (Ok)

Tworzenie formularzy

Projekt C# typu Windows Forms ma jeden formularz, ale może być ich więcej

- Do projektu można dodawać kolejne formularze (Project / Add Windows Form),
- Domyślnie wyświetlane jest tylko główne okno, dodatkowe okna trzeba utworzyć i wyświetlić programowo,
- Do wyświetlania okna można użyć jednej z dwóch metod: *Show* lub *ShowDialog*. Sposób zachowania okien i ich programowej obsługi zależy od wyboru jednej z nich.

Okno niemodalne

- Okno wyświetlone metodą *Show* jest niemodalne (użytkownik może przełączać się między oknem głównym a dodatkowymi oknami niemodalnymi);
- Program nie jest wstrzymywany – kolejne (po *Show*) instrukcje są natychmiast wykonywane
- Okno jest zamykane, a nie ukrywane (np. po kliknięciu w znak "x" w rogu okna)
 - zwalniane są wszystkie jego zasoby,
 - generowane są zdarzenia *FormClosing* i *FormClosed*,
 - próba ponownego wyświetlenia lub odwołania się do jego komponentów lub właściwości kończy się błędem,
 - aby ponownie wyświetlić okno, trzeba je na nowo utworzyć
- Sposób wyświetlenia

```
Form2 f2 = new Form2();  
f2.Show();
```
- Do komunikacji między formami trzeba napisać własne, dedykowane metody

Okno modalne

- Okno wyświetlone metodą *ShowDialog* jest modalne (ang. *modal dialog window*; użytkownik nie może przełączać się między oknem dialogowym a innymi oknami);
- Program jest wstrzymywany
 - kolejne (po *ShowDialog*) instrukcje są wykonywane dopiero po zamknięciu okna
- Okno jest ukrywane, a nie zamykane (np. po kliknięciu w znak "x" w rogu okna)
 - jego zasoby nie są zwalniane,
 - można je wyświetlić ponownie (stan komponentów nie zmienia się w międzyczasie),
 - dostępna je właściwość *DialogResult*,
 - należy upewnić się, że jest tworzone tylko raz (a nie przy każdym wyświetleniu)
- Sposób wyświetlenia
`Form2 f2 = new Form2();`
`f2.ShowDialog();`
- Do komunikacji między formami można napisać własne, dedykowane metody (np. aby sprawdzić jakie opcje zostały wybrane)
- Alternatywnie można odczytać stan komponentów
 - trzeba mieć pewność, że formularz istnieje
 - modyfikator dostępu komponentów w oknie dialogowym trzeba zmienić na *public*

Rezultat okna modalnego

- Ważnym aspektem okien dialogowych jest ich rezultat (zwykle do wyboru są minimum 2 możliwości, np. Ok/Anuluj, Yes/No itp.); do określenia rezultatu służy właściwość *DialogResult*;
- Nadanie wartości właściwości *DialogResult* formularza powoduje "zamknięcie" (tak naprawdę tylko ukrycie) okna modalnego
`this.DialogResult = DialogResult.OK;`
- Jeżeli przycisk, umieszczony na formularzu modalnym, ma przypisaną wartość *DialogResult*, to jego kliknięcie powoduje przypisanie tej wartości do właściwości *DialogResult* formularza, a tym samym jego "zamknięcie"
- Zwykle przyciski *Ok* i *Anuluj* są przyciskami domyślnymi formularza:
 - **Ok** (albo Zapisz, Otwórz itp.)
`AcceptButton` okna
`DialogResult.OK`
 - **Anuluj**
`CancelButton` okna
`DialogResult.Cancel`
- Ponieważ wykonanie funkcji wyświetlającej okno modalne zostaje wstrzymane do jego zamknięcia, to odczytanie wartości *DialogResult* zwykle umieszcza się w kolejnej instrukcji
- Niezależnie od wartości *DialogResult*, formularz nic sam nie robi - za to odpowiada program, np.:
`f2.ShowDialog();`
`if (f2.DialogResult == DialogResult.OK)`
`{`
`// działania jeżeli użytkownik kliknie "OK"`
`}`

- W przypadku okien z opcjami programu, jeżeli wybrano *Anuluj*, to zwykle odtwarzane są opcje sprzed wyświetlenia okna, np. tak:

```
op1 = fOpt.radioButton1.Checked;

if (fOpt.ShowDialog() == DialogResult.Cancel)
{
    fOpt.radioButton1.Checked = op1;
}
```

- Bardziej eleganckie rozwiązanie polega na zdefiniowaniu i użyciu właściwości

```
Boolean op = true;
Form2 fOpt = new Form2();
```

```
... btnOptions_Click ...
{
    fOpt.op = op;
```

```
    if (fOpt.ShowDialog() == DialogResult.OK)
        op = fOpt.op
}
```

- Można też zadbać o to, aby okno nie było tworzone bez potrzeby (aplikacja mniej obciąża system)

```
Form2 fOpt = null;
```

```
... btnOptions_Click ...
{
    if (fOpt == null)
        Form2 fOpt = new Form2();

    fOpt.ShowDialog();
    ...
}
```

Dostęp do komponentów okna dialogowego

- Domyślnie komponenty mają modyfikator dostępu **private**, co oznacza że nie są dostępne z innego formularza

```
Form2 fOpt;
```

```
... btnDoIt_Click ...
{
    if (fOpt.radioButton1.Checked)
    {
        // ...
    }
}
```

Błąd kompilacji:
*Error: 'radioButton1' is inaccessible
 due to its protection level*



- Najprostszym rozwiązaniem jest zmiana modyfikatora dostępu na **internal** (dostęp możliwy z innych klas należących do tego samego zestawu) lub **public**
- Bardziej eleganckie rozwiązanie polega, jak wspomniano, na zdefiniowaniu właściwości do odczytu i zapisu wartości opcji

Definicja właściwości :

Właściwość jest definiowana jako połączenie dwóch metod – *get*, do pobierania wartości, oraz *set*, do zmiany wartości – ale jest używana jak zmienna

```
internal Boolean Option1
{
    get
    {
        return radioButton1.Checked;
    }
    set
    {
        radioButton1.Checked = value;
    }
}
```

Składnia właściwości

- *Modyfikator* – domyślnie modyfikatorem jest *private*; należy użyć *internal* lub *public*
- *Typ* – określa typ właściwości
- *Sekcja_get* – musi zwrócić wartość właściwości zgodną z *Typem* (przy pomocy instrukcji *return*)
- *Sekcja_set* – otrzymuje niejawnny argument o nazwie **value**, którego typ jest identyczny jak *Typ* właściwości, który powinna (choć nie musi) zapisać
- Jedną z sekcji można pominąć
jeżeli np. będzie tylko sekcja *get*, to uzyskamy właściwość "tylko do odczytu"

```
[Modyfikator] Typ Nazwa
{
    get { Sekcja_get }
}
```

Użycie właściwości

- Właściwość jest używana poprzez obiekt formularza, do którego należy (podobnie, jak metoda)

```
... btnOptions_Click ...
{
    fOpt.Option1 = opt1;

    if (fOpt.ShowDialog == DialogResult.OK)
    {
        opt1 = fOpt.Option1;
    }
}
```

Implementacja Ok i Anuluj, scenariusz 1

Scenariusz 1 jest zdecydowanie łatwiejszy w realizacji, ale nie jest zgodny z tzw. dobrymi praktykami programowania – łamie kilka ważnych zasad programowania obiektowego. Takie podejście jest poprawne składniowo i, co ważniejsze, działa, ale przez łamanie wspomnianych zasad bardzo utrudnia tworzenie bardziej złożonych programów.

1. Zmiana kwalifikatorów dostępu

Komponenty okna opcji, tylko te służące bezpośrednio do wyboru opcji (np. przyciski radiowe i pola wyboru), muszą być „widoczne” z okna głównego – należy zmienić ich kwalifikatory dostępu (ang. *Modifier*) z domyślnego *private* na *public*.

Właściwości tych komponentów, np. *Checked* dla przycisków radiowych, będą wartościami opcji w chwili startu programu i powinny być ustawione odpowiednio do tego.

2. Utworzenie okna opcji

Opcje będą odczytywane bezpośrednio z komponentów okna opcji, dlatego okno to musi istnieć przez cały czas działania programu. W przestrzeni pomiędzy funkcjami obsługi zdarzeń okna głównego, najlepiej gdzieś na początku klasy *Form1*, należy dodać kod:

```
Form2 formOptions = new Form2();
```

3. Uwzględnianie opcji w programie

Aby uwzględnić opcje, należy w funkcjach obsługi zdarzeń sprawdzać właściwości komponentów okna opcji. Trzeba przy tym użyć nazwy okna opcji nadanej w kroku 2, np.:

```
... button1_Click(...)
{
    if (formOptions.radioButton1.Checked)
    {
        // jeżeli opcja 1 włączona
    }
    else
    {
        // jeżeli opcja 1 wyłączona
    }
}
```

4. Wyświetlenie okna opcji (tylko OK)

Wyświetlenie okna opcji bez możliwości anulowania zmian opcji jest trywialnie proste, należy użyć metody *ShowDialog*, nic więcej nie trzeba robić:

```
... button2_Click(...)
{
    formOptions.ShowDialog();
}
```

5. Wyświetlenie okna opcji (Ok i Anuluj)

Dodanie możliwości anulowania zmian opcji wymaga działania w trzech krokach:

(1) zapamiętanie stanu opcji przed wyświetleniem okna opcji, (2) wyświetlenie okna opcji oraz (3) przywrócenie opcji sprzed wyświetlenia okna opcji, jeżeli okno opcji zostanie zamknięte przyciskiem *Anuluj*. Do zapamiętania stanu opcji należy zadeklarować odpowiednie zmienne, należy przy tym zapamiętać stan *wszystkich* komponentów opcji, np.:

```
... button2_Click(...)
{
    Boolean opcjaObliczenia, opcjaCzyszczenie;

    // zapamiętanie
    opcjaObliczenia = formOptions.radioButton1.Checked;
    opcjaCzyszczenie = formOptions.checkBox1.Checked;

    if (formOptions.ShowDialog() == DialogResult.Cancel)
    {
        // przywracanie
        formOptions.radioButton1.Checked = opcjaObliczenia;
        formOptions.checkBox1.Checked = opcjaCzyszczenie;
    }
}
```

Implementacja Ok i Anuluj, scenariusz 2

Scenariusz 2 jest nieco trudniejszy, ale zdecydowanie lepszy – jest zgodny z tzw. dobrymi praktykami programowania i sprawdzi się nawet w bardzo złożonych programach.

1. Zmienne opcji

Opcje będą odczytywane ze zmiennych, które należy dodać do kodu pomiędzy funkcjami obsługi zdarzeń okna głównego, najlepiej gdzieś na początku klasy *Form1*. Okno opcji może zostać utworzone razem z tymi zmiennymi, ale nie jest to konieczne – w tym scenariuszu okno opcji nie musi istnieć przez cały czas.

```
Boolean opcjaRzeczywiste = true;
Boolean opcjaDodajZera = false;
Boolean opcjaCzyszczenie = true;

Form2 formOptions = null;
```

2. Uwzględnianie opcji w programie

Aby uwzględnić opcje, należy w funkcjach obsługi zdarzeń sprawdzać zmienne opcji

```
... button1_Click(...)
{
    if (opcjaRzeczywiste)
    {
        // jeżeli opcja 1 włączona
    }
    else
    {
        // jeżeli opcja 1 wyłączona
    }
}
```

3. Definicja właściwości

To najtrudniejsza część kodu – dla każdej opcji należy zdefiniować właściwość. Należy to zrobić w kodzie okna opcji. Właściwości mogą nazywać się tak samo, jak zmienne opcji okna głównego, ale nie jest to konieczne. Sekcja *get* właściwości powinna zwracać stan opcji, np. zgodnie z zaznaczeniem przycisku radiowego, natomiast sekcja *set* powinna ustawiać zaznaczenie tego przycisku radiowego zgodnie z wartością przekazywaną przez *value*, np.:

```
public Boolean opcjaRzeczywiste
{
    get
    {
        return radioButton1.Checked;
    }
    set
    {
        radioButton1.Checked = value;
    }
}
```

Ważne! Należy zdefiniować właściwość dla *każdego* komponentu wyboru opcji, np. jeżeli będą cztery przyciski radiowe (dodawanie, odejmowanie, mnożenie i dzielenie) – to dla każdego z nich. Alternatywnie można zdefiniować jedną wspólną właściwość typu *Int32*, która zwraca *numer* zaznaczonej opcji (zostało to pokazane na wkładzie).

4. Wyświetlenie okna opcji

Wyświetlenie okna opcji nadal jest proste, ale bez dodatkowych działań nie przynosi żadnych skutków (opcje programu nie zmienia się, pomimo zmiany stanu komponentów okna opcji)

```
... button2_Click(...)
{
    if (formOptions == null)
        formOptions = new Form2();

    formOptions.ShowDialog();
}
```

5. Wyświetlenie okna opcji (Ok i Anuluj)

Dodanie możliwości zapamiętania zmian opcji wymaga działania w trzech krokach:
(1) przeniesienie stanu opcji ze zmiennych okna głównego do właściwości okna opcji,
(2) wyświetlenie okna opcji oraz (3) zapamiętanie nowych wartości zmiennych opcji, jeżeli okno opcji zostanie zamknięte przyciskiem *Ok*, np.:

```
... button2_Click(...)
{
    if (formOptions == null)
        formOptions = new Form2();

    // ustawienie stanu okna opcji
    formOptions.opcjaRzeczywiste = opcjaRzeczywiste;
    formOptions.opcjaDodajZera = opcjaDodajZera;

    if (formOptions.ShowDialog() == DialogResult.OK)
    {
        // zapamiętanie nowych ustawień
        opcjaRzeczywiste = formOptions.opcjaRzeczywiste;
        opcjaDodajZera = formOptions.opcjaDodajZera;
    }
}
```

Znaczenie opcji kalkulatora

1. Rodzaj operacji

Zależnie od wariantu kalkulatora, może to oznaczać rodzaj operacji arytmetycznej (dodawanie, odejmowanie, ...), sposób obliczania pierwiastków równania kwadratowego dla delty ujemnej (opcja *rzeczywiste* – komunikat *brak pierwiastków*, opcja *zespolone* – obliczenie pierwiastków zespolonych) itp.

Niezależnie od wariantu kalkulatora, do tej opcji należy użyć przycisków radiowych.

2. Dodawanie zer

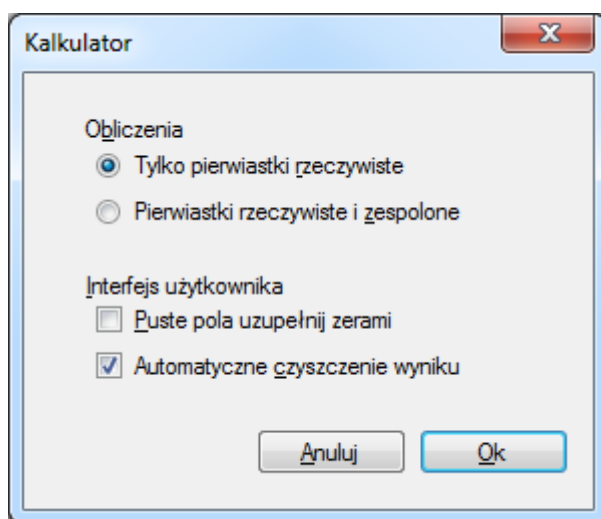
Do tej opcji należy użyć pola wyboru. Jeżeli opcja jest włączona, to puste pola tekstowe danych należy traktować jakby miały wpisane zera. Najłatwiej uwzględnić tę opcję wstawiając zera do pustych pól tekstowych *przed* wykonaniem innych czynności, np.:

```
if (opcjaDodajZera)
{
    if (String.IsNullOrEmpty(textBox1.Text))
        textBox1.Text = "0";
}
```

Funkcja *IsNullOrWhiteSpace* zwraca *true*, jeżeli łańcuch jest pusty lub zawiera tylko tzw. białe znaki (spacje i tabulatory). Można też użyć porównania `textBox1.Text == ""`

2. Automatyczne czyszczenie

Do tej opcji także należy użyć pola wyboru. Jeżeli opcja jest włączona, to zmiana zawartości dowolnego pola tekstowego wprowadzania danych powinna spowodować wyczyszczenie wszystkich pól tekstowych wyników. Aby uwzględnić tę opcję, należy utworzyć funkcję obsługi zdarzenia *TextChanged*, co ważne – wspólną dla *wszystkich* pól tekstowych *danych*, a w jej kodzie wpisać wyczyszczenie *wszystkich* pól tekstowych *wyników*.

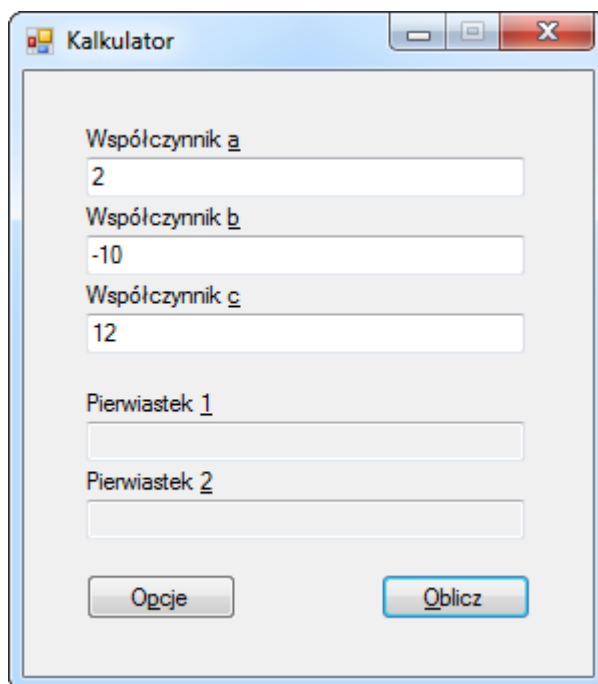


Zadania

Celem ćwiczenia jest opracowanie kalkulatora, wyposażonego w okno główne oraz okno opcji. Okno opcji, wyświetlane po kliknięciu odpowiedniego przycisku, umożliwia zmianę ustawień programu. Zmiany te można zatwierdzić lub je anulować, do czego służą odpowiednie przyciski. Należy też zadbać o szczegóły interfejsu użytkownika, tj. skróty klawiaturowe do wszystkich komponentów, użycie przycisków domyślnych, zablokowanie lub ograniczenie zmiany rozmiaru okien itd. Kalkulator może wykonywać różne obliczenia, zależnie od wyboru prowadzącego ćwiczenia. Może to być np. obliczanie sumy, różnicy, iloczynu i ilorazu dwóch liczb, obliczanie pierwiastków równania kwadratowego itp.

1. Okno główne aplikacji

- Utworzyć nowy projekt, język C#, typ projektu Windows Forms,
- Umieścić na formularzu komponenty wg wzoru na rysunku poniżej;
Należy dopilnować, aby etykiety i pola tekstowe były dodawane jedno po drugim, aby poprawnie działały skróty klawiaturowe etykiet,
- Ustawić właściwości okna i komponentów: okno powinno mieć rozmiar stały albo zmieniany tylko w ograniczonym zakresie (wówczas komponenty powinny mieć adekwatnie zmienione zakotwiczenie). Tytuł okna i opisy komponentów powinny być zgodne z ich przeznaczeniem, pola tekstowe z wynikami – nieedytowalne, przycisk *Oblicz* – domyślny dla formularza; Wszystkie komponenty powinny mieć działające skróty klawiaturowe.
- Nie należy zmieniać czcionek ani kolorów tekstu i tła komponentów

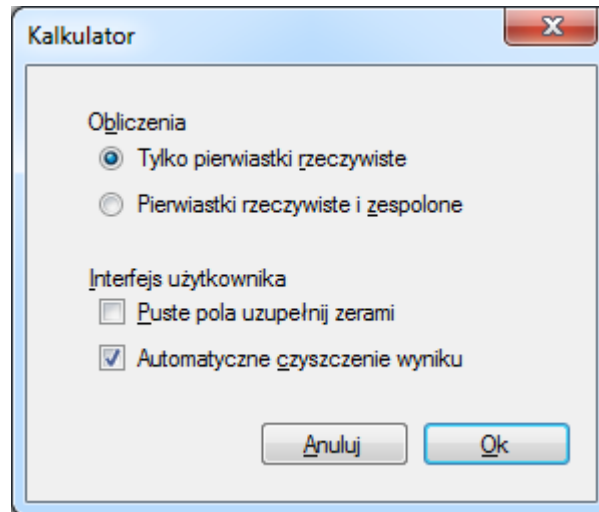


2. Obliczenia

- Dodać obsługę kliknięcia przycisku *Oblicz*. Obliczenia powinny być wykonywane według domyślnych ustawień aplikacji (np. w równaniu kwadratowym tryb *tylko pierwiastki rzeczywiste*, opcja *dodawanie zer* – wyłączona),
- Dodać obsługę zdarzenia *TextChanged* pól tekstowych z danymi – skutkiem tego powinno być wyczyszczenie wcześniej obliczonych wyników przy jakiegokolwiek zmianie danych (czyli opcja *automatyczne czyszczenie* – włączona).

3. Okno opcji

- Do projektu dodać drugi formularz i umieścić na nim komponenty wg wzoru na rysunku poniżej,
- Ustawić właściwości okna i komponentów z dbałością o szczegóły, podobnie jak dla okna głównego aplikacji, ale pamiętając że ma to być okno dialogowe



Do dalszej części ćwiczeń należy wybrać jeden scenariusz obsługi opcji, tj. zadanie 4 *albo* 5

4. Opcje programu i ich zmiana, scenariusz 1

- Zmienić kwalifikatory dostępu komponentów odpowiedzialnych za wybór opcji na oknie opcji (komponenty typu *RadioButton* i *CheckBox*) na *public*
- Dodać obsługę kliknięcia przycisku *Opcje* na oknie głównym – powinno zostać wyświetlone okno opcji w trybie modalnym
- Zmodyfikować kod programu, aby uwzględnić opcje
- Zmodyfikować kod w funkcji kliknięcia przycisku *Opcje*, aby umożliwić anulowanie i zatwierdzanie zmian opcji, tj. zapamiętanie stanu opcji przed wyświetleniem okna opcji i ich przywrócenie w przypadku kliknięcia przycisku *Anuluj* (dla kliknięcia przycisku *Ok* nie trzeba wykonywać żadnych działań)

5. ** Opcje programu i ich zmiana, scenariusz 2

- Kwalifikatory dostępu wszystkich komponentów na oknie opcji powinny mieć wartości domyślne, tj. *private*,
- W kodzie okna głównego dodać zmienne reprezentujące wartości opcji,
- Zmodyfikować kod programu, aby uwzględnić opcje,
- W kodzie okna opcji dodać definicje właściwości związanych z opcjami,
- Dodać obsługę kliknięcia przycisku *Opcje* na oknie głównym – powinno zostać wyświetlone okno opcji w trybie modalnym,
- Zmodyfikować kod w funkcji kliknięcia przycisku *Opcje*, aby umożliwić anulowanie i zatwierdzanie zmian opcji, tj. przenieść wartości zmiennych opcji z okna głównego do właściwości okna dialogowego przed wyświetleniem tego okna, następnie wyświetlić okno opcji i zapisać nowe wartości opcji w przypadku kliknięcia przycisku *OK* (dla kliknięcia przycisku *Anuluj* nie trzeba wykonywać żadnych działań)