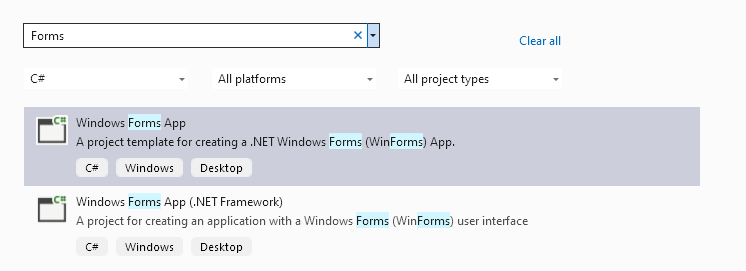
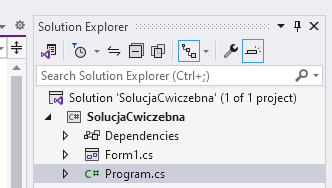
**Aplikacje z interfejsem graficznym**

**Lekcja 1**: Pierwsza Solucja Forms: omówienie istotnych fragmentów otwartego szablonu Forms.

1. Otwieramy Visual Studio
2. Create a new project
3. W -> „Search for templates” piszemy Forms wybieramy pierwsze wskazanie bez bibliotek.

NEXT

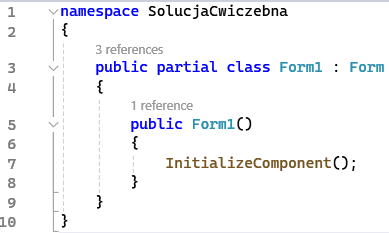
1. Nazywamy solucje np.:SolucjaCwiczebna.
2. Stworzona solucjaCwiczebna z prawej strony monitora posiada układ szablonu



1. W Program.cs znajdziemy  oraz  W ciele funkcji main() znajdują się:

* ApplicationConfiguration.Initialize(); - ta metoda inicjalizująca wszystkie komponenty.
* Application.Run(new Form1()); Czyli wywołanie metody Run klasy Application ( jest to również metoda statyczna). Jako argument został jej przekazany nowo utworzony obiekt typu Form. Jest to właśnie sygnał do uruchomienia aplikacji okienkowej, a ponieważ argumentem jest obiekt typu Form również do wyświetlania okna na ekranie.

1. W szablonie wskazaniem do uruchomienia jest plik Form1 o czym świadczy zapis …Run(new Form1()). Jeżeli z prawej strony najedziemy myszą na ten plik i rozwiniemy menu wybierając – View Code - Uzyskamy możliwość kodowania w następującym dokumencie:

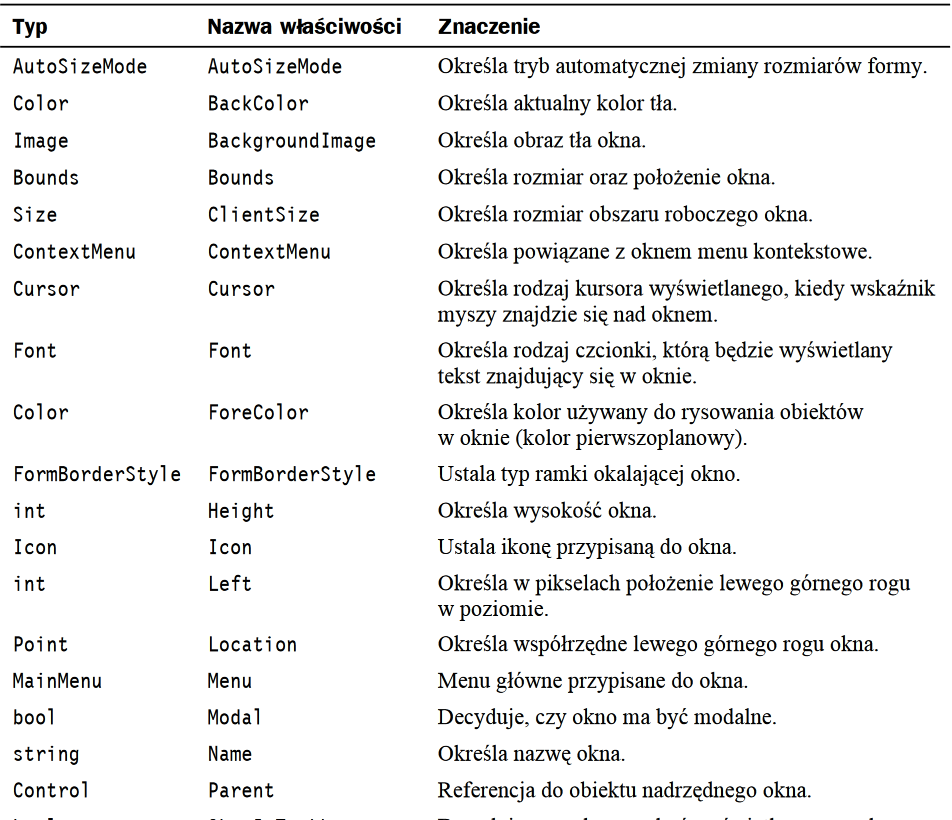


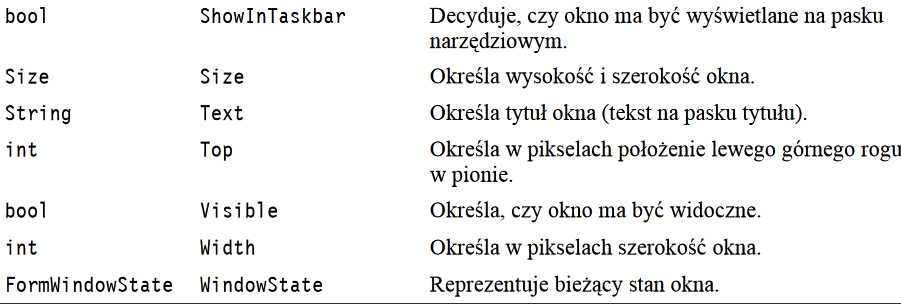
Gdzie linijka 3 to klasa Form1, która jest dziedziczona od klasy Form. Słowo partial oznacza możliwość pracy w kilku plikach przy jednym wywołaniu. Linijka 7 rozpoczyna inicjalizację okna w konstruktorze.

**Lekcja 2**: Klasa Form oraz jej wybrane właściwości.

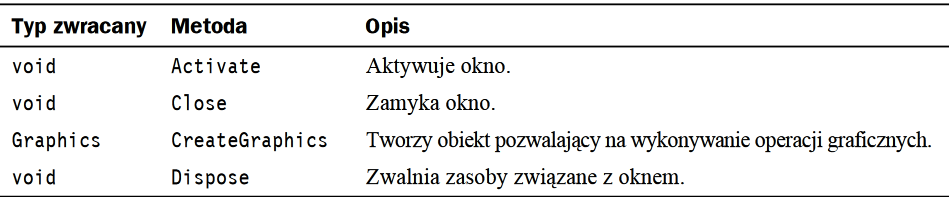
Okno aplikacji jest opisywane przez klasę Form.

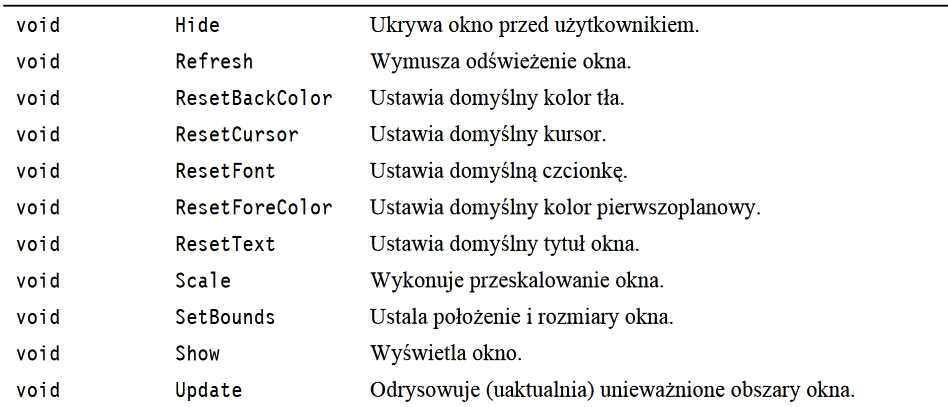
**Wybrane właściwości klasy Form**



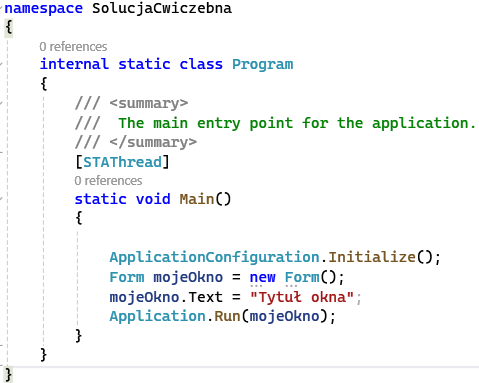


**Wybrane metody klasy Form**



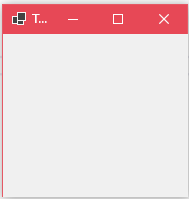


**Przykład** okna zawierającego określony tytuł należy zwrócić uwagę na sposób wywołania.

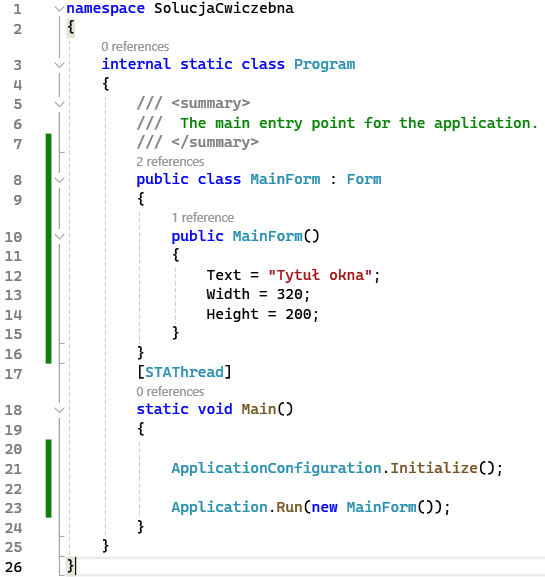


W przykładzie stworzyliśmy zmienną mojeOkno typu Form Dzięki temu możemy zmieniać jego właściwości. Modyfikujemy więc właściwość Text. Przypisując jej ciąg znaków „Tytuł okna”.

Po dokonaniu tego przypisania przekazujemy zmienną mojeOkno jako argument metody Run, dzięki czemu aplikacja okienkowa rozpoczyna swoje działanie, a okno (forma, formatka) pojawia się na ekranie:



Wprowadzenie własnej klasy pochodnej od Form. W praktyce wyglądałoby to tak:



Powstała klasa MainForm dziedziczy po Form więc przejmuje jej cechy i właściwości. Mamy tu konstruktor o tej samej nazwie co klasa a w nim trzy cechy tytuł okna szerokość -Width i wysokość Height .

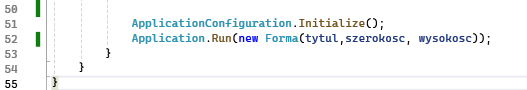
W tym przypadku jako argument metody Run jest przekazywany nowo utworzony obiekt naszej klasy MainForm a nie Form. I tak okno jest reprezentowane przez klasę MainForm

Zadanie 1

Okno o tytule i rozmiarze wyprowadzane z wiersza poleceń.

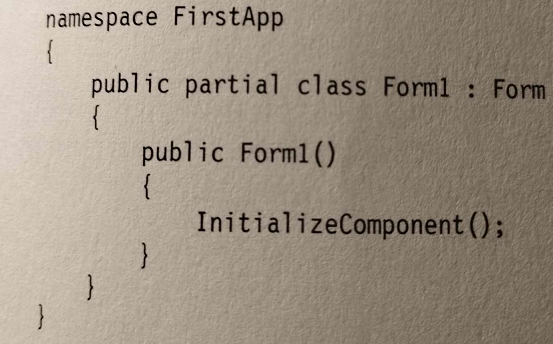
Konstruktor przyjmie trzy argumenty określające parametry okna: tytuł, szerokość i wysokość.





**Lekcja 3: Projekt okna**

1. Otwieramy pierwszy projekt, FirstApp.
2. W oknie Solution Explorer nakierowujemy wskaźnik myszy na formularz Form1.cs
3. Klikamy prawym przyciskiem nazwę formularza i z menu wybieramy View Code
4. Zostaniemy przeniesieni do kodu programu
5. Na początku kodu programu znajduje się jego nazwa; w tym przypadku po namespace występuje nazwa FirstApp.
6. Następnie jest definicja klasy Form1, będącej rozszerzeniem klasy ogólnej Form.
7. Wewnątrz klasy znajduje się metoda Form(); która jest wywoływana automatycznie po rozpoczęciu działania programu.
8. W tej metodzie znajduje się metoda inicjalizująca wszystkie komponenty, InitializeComponent().
9. Dopiero po tej metodzie możemy zamieścić swój kod.



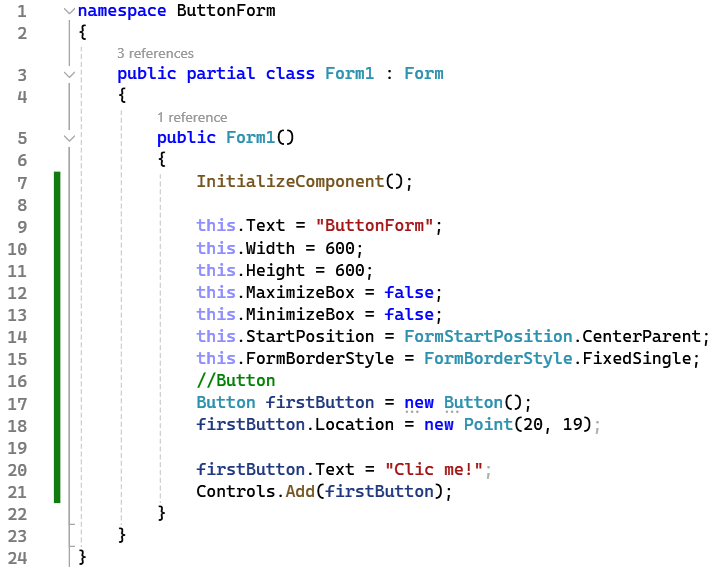
1. Okno powinno mieć tytuł „My first application” szerokość 800, wysokość 600px. Będzie pozbawione przycisków do maksymalizacji i minimalizacji okna a jego szerokość i wysokość zostaną ustawione jako niezmienne.



Dzięki kontrolkom można zbudować aplikację – to dzięki nim powstaje interfejs użytkownika. Budując aplikację potrzebujemy pola do wpisywania tekstu, przycisku, po którego kliknięciu była wykonywana jakaś akcja czy też nowego okna wyświetlającego informację na temat programu.

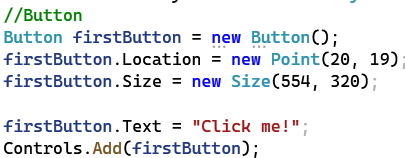
Zadanie 2.

1. Otwieramy projekt (nazwa).
2. Klikamy prawym przyciskiem nazwę formularza i z menu wybieramy View Code.
3. Zostaniemy przeniesieni do kodu programu.
4. Musimy określić jego nazwę przypisaną do klasy przycisku, podać jego lokalizację i wpisać tekst do jego zawartości. Na koniec musimy dodać przycisk do formularza.
5. Aby dodać nowy przycisk:
6. Określamy nazwę naszej aplikacji (this.Text).
7. Następnie ustawiamy szerokość (this.Width) oraz wysokość (this.Height) formularza.
8. Ukrywamy przycisk do maksymalizacji (this.MaximizeBox), i minimalizacji (this.MinimizeBox) .
9. Aplikacja powinna być uruchomiona na środku ekranu (this.StartPosition).
10. Wielkość aplikacji będzie można zmieniać także przez przeciąganie jej aktywnych brzegów, dlatego musimy ustawić this.FormBorderStyle na obramowanie pojedyncze.
11. Definiujemy obiekt przycisku w którym wstawiamy naszą nazwę elementu a następnie po znaku równości definiujemy utworzenie nowego elementu Button firstButton = new Button()
12. We właściwościach, które posiada zdefiniowany element wskazujemy jego lokalizację punkt określony przez podane punkty X i Y w naszym formularzu (firstButton.Location()).
13. Napis wprowadzamy przez firstButton.Text
14. Dodajemy do bieżącego formularza przez dodanie nowej kontrolki: Controls.Add(firstButton).



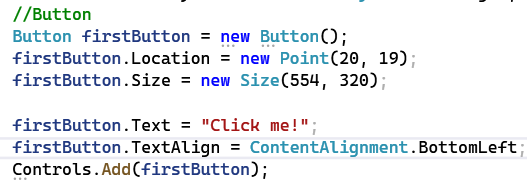
**Określenie wysokości i szerokości**

Do poprzedniego zadania należy dopisać określenie szerokości i wysokości naszego przycisku:



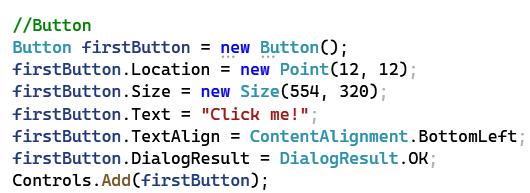
**Wyrównywanie tekstu względem granic przycisku:**

Przycisk posiada opcję wyrównywania tekstu, który jest w nim zawarty, w zależności od ustawień. Można wyrównać tekst od lewej , prawej od środka ale także względem narożników.



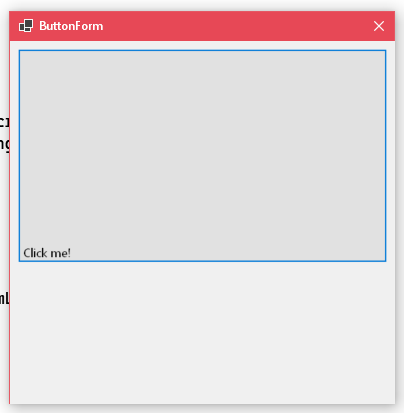
**Rezultat naciśnięcia przycisku (47)**

Naciśnięcie przycisku zwraca re-zultat, który możemy przechowywać jako informację zwrotną z danego komunikatu czy też okna utworzonego w celu zadania zapytania spoza zestawu domyślnych w Visualu … .

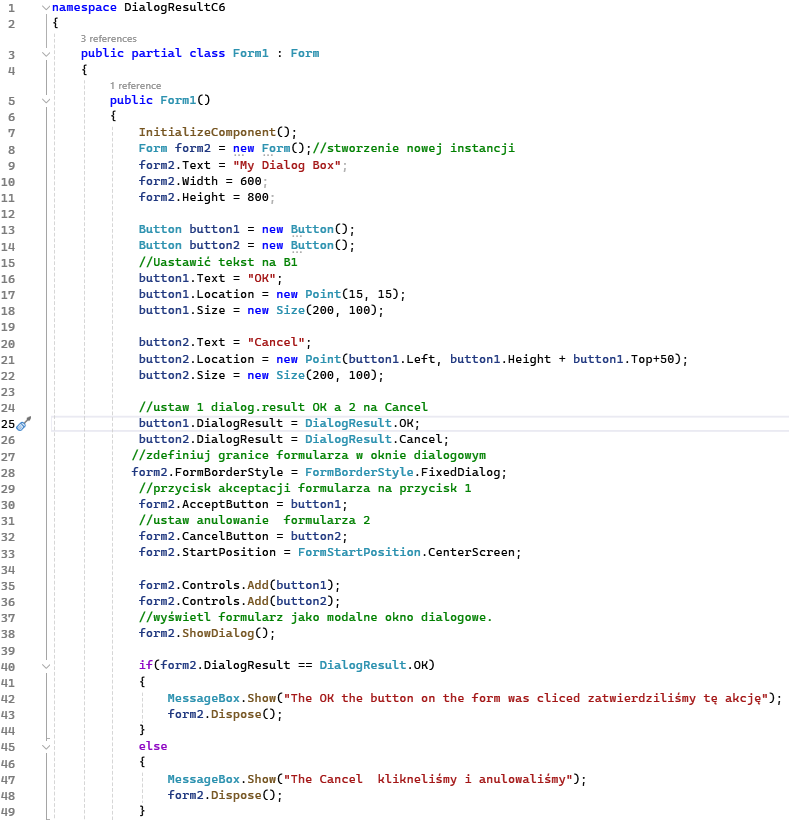


We właściwościach przycisku wartość, którą ma zwrócić w wyniku kliknięcia. Tą wartością jest OK przypisujemy ją do właściwości *firstButton.DialogResult*.

Po zakończeniu musimy dodać do bieżącego formularza przez dodanie nowej kontrolki: *Controls.Add(firstButton);*

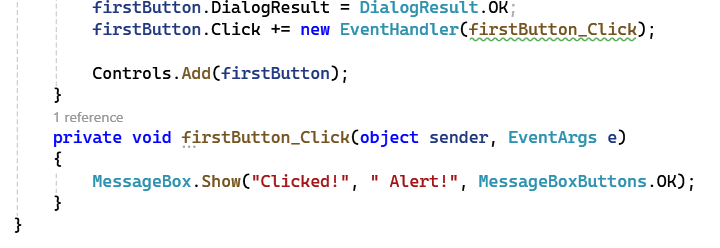


|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Form.DialogResult*** Właściwość  Wartość właściwości DialogResult  Obiekt DialogResult reprezentujący wynik formularza, który jest używany jako okno dialogowe.  public enum DialogResult  Określa identyfikator (wartość)zwracaną przez okno dialogowe   |  |  |  | | --- | --- | --- | | Abort | 3 | The dialog box return value is Abort (usually sent from a button labeled Abort). | | Cancel | 2 | The dialog box return value is Cancel (usually sent from a button labeled Cancel). | | Continue | 11 | The dialog box return value is Continue (usually sent from a button labeled Continue). | | Ignore | 5 | The dialog box return value is Ignore (usually sent from a button labeled Ignore). | | No | 7 | The dialog box return value is No (usually sent from a button labeled No). | | None | 0 | Nothing is returned from the dialog box. This means that the modal dialog continues running. | | OK | 1 | The dialog box return value is OK (usually sent from a button labeled OK). | | Retry | 4 | The dialog box return value is Retry (usually sent from a button labeled Retry). | | TryAgain | 10 | The dialog box return value is Try Again (usually sent from a button labeled Try Again). | | Yes | 6 | The dialog box return value is Yes (usually sent from a button labeled Yes). |   Wynikiem okna dialogowego formularza jest wartość zwracana z formularza, gdy jest wyświetlana jako okno modalne dialogowe. Jeżeli formularz jest wyświetlany jako okno dialogowe, ustawienie tej właściwości z wartością z DialogResult ustawia wartość wyniku okna dialogowego formularza, ukrywa okno dialogowe i zwraca kontrolkę do formularza wywołującego.  **modal window)** – **okno nie pozwalające na obsługę zdarzeń dotyczących pozostałych okien danej aplikacji**. Wówczas żadne inne okno aplikacji nie reaguje na działania użytkownika. W większości przypadków przyjmuje ono postać okna dialogowego z jakimś zapytaniem do użytkownika, które znika, gdy wybierzemy odpowiedź np. |



**Przypisywanie akcji**

Jedną z najważniejszych opcji przycisku jest możliwość przypisania mu akcji, czyli czynności wykonywanej w wyniku kliknięcia go myszą



1. Do przycisku należy przypisać nowy element obsługi zdarzenia kliknięcia a następnie zdefiniować metodę dla akcji przycisku:

*firstButton.Click +=new EventHandler(firstButton\_Click);*

1. Następnie poza Form1() definiujemy metodę do obsługi tego zdarzenia poprzez zapis:

*Private void firstButton\_Click(object sender, EventArgs e).*

Metoda ta ma dwa argumenty: pierwszy jest element, który wywołał dane żądane, a drugim ewentualne błędy

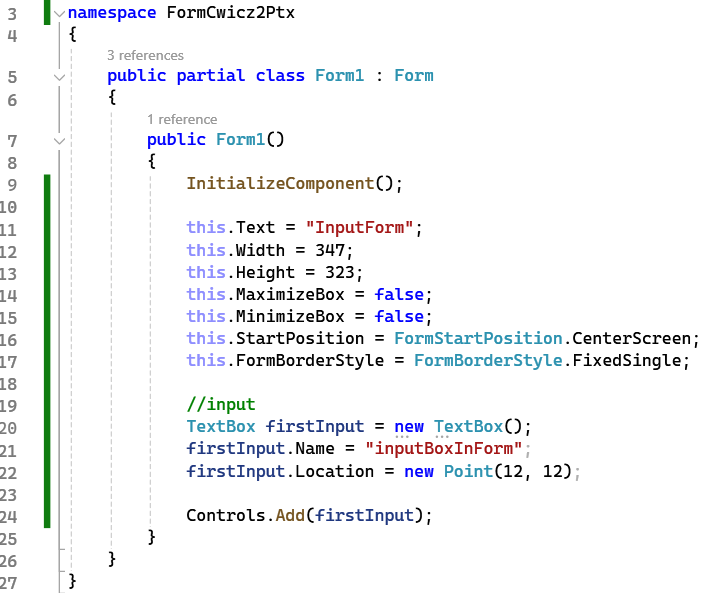
1. Kolejną rzeczą jest zdefiniowanie wewnątrz metody akcji, którą w tym przypadku będzie wyświetlenie okna zawierającego informację o tym, że dany przycisk został kliknięty (*MessageBox.Show*).
2. Po zakończeniu tworzenia przycisku trzeba go jeszcze dodać do bieżącego formularza przez dodanie nowej kontrolki: Controls.Add(firstButton).

**Pole tekstowe** (TextBox)

Pole to jest przeznaczone do wprowadzania tekstu do tworzenia oprogramowania. Może ono być w formie jednoliniowego komunikatu lub w postaci wielolinijkowego tekstu.

Dodawania pola tekstowego

Tworzenie pola tekstowego polega na określaniu jego nazwy, położenia oraz rozmiarów, a następnie dodaniu go do formularza za pomocą odpowiedniej metody.



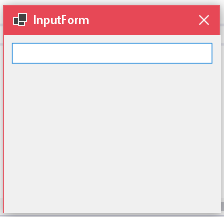
1. Tworzymy pole tekstowe przez przepisanie go do obiektu, w którym zawarta jest definicja pola: *TextBox firstInput = new TextBox();*
2. Tworzymy pole tekstowe przez przypisanie go do obiektu, w którym zawarta jest definicja pola (*firstInput.Name*).
3. Ustawiamy lokalizację przez podanie współrzędnych we właściwości *Location*.
4. Na koniec dodajemy pole tekstowe do formularza przez dodanie go jako kontrolki (*Controls.Add*).

Ustawienie szerokości i wysokości pola

Przy przekształcaniu pola w pole wieloliniowe bardzo ważnym elementem są jego szerokość oraz wysokość. Te parametry pozwolą nam na lepsze dopasowanie jego rozmiarów do planowanych czynności.



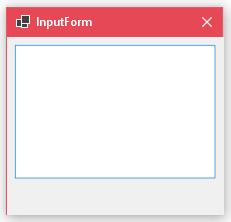
1. Aplikacja powinna się wyświetlić na środku ekranu ( *this.StartPosition*)
2. Wielkość okna będzie można zmienić przez przeciąganie jego aktywnych brzegów, dlatego należy ustawić *this.FormBorderStyle* na obramowanie niepozwalające na takie działanie.
3. Tworzymy pole tekstowe przez przypisanie go do obiektu, w którym zawarta jest definicja pola: *TextBox firstInput = new TextBox().*
4. Określamy nazwę pola (firstInput.Name).
5. Ustawiamy lokalizację pola przez podanie współrzędnych we właściwości *Location*.
6. Określamy szerokość oraz wysokość pola. Przy użyciu właściwości *Width i Height*.
7. W końcu dodajemy do formularza przy pomocy *Controls.Add*.



Dodawanie do pola tekstowego możliwości pisania w wielolinii

Musimy włączyć możliwość pisania w wielolinii.





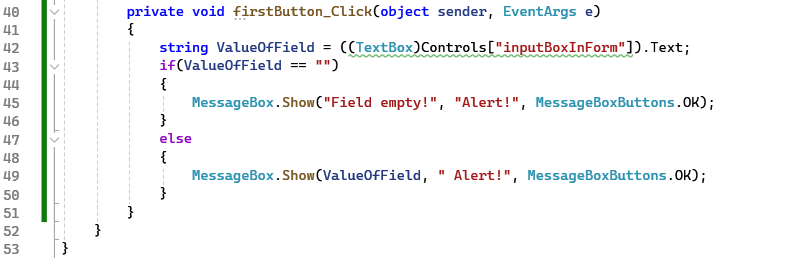
Aby uzyskać działający kod

1. Aplikacja powinna być ustawiona na środku ekranu *this.StartPosition* .
2. Ustawiamy niemożność rozszerzania okna *FotmBorderStyle.FixedSingle*. Aby uzyskać rozszerzenie należy wpisać po kropce: *Sizable*
3. Tworzymy pole tekstowe przez przypisanie go do obiektu, w którym zawarta jest definicja pola *TextBox firstInput = new TextBox().*
4. Określamy nazwę pola (*firstInput.Name*)
5. Ustalamy lokalizację przez podanie współrzędnych we właściwości.
6. Określamy szerokość oraz wysokość pola tekstowego. Robimy to z użyciem właściwości *Width Height*.
7. Ustawiamy wielolinie: *Multiline* wartość *true*. Czyli potwierdzenie kilku linijek
8. Dodajemy pole tekstowe do formularza.

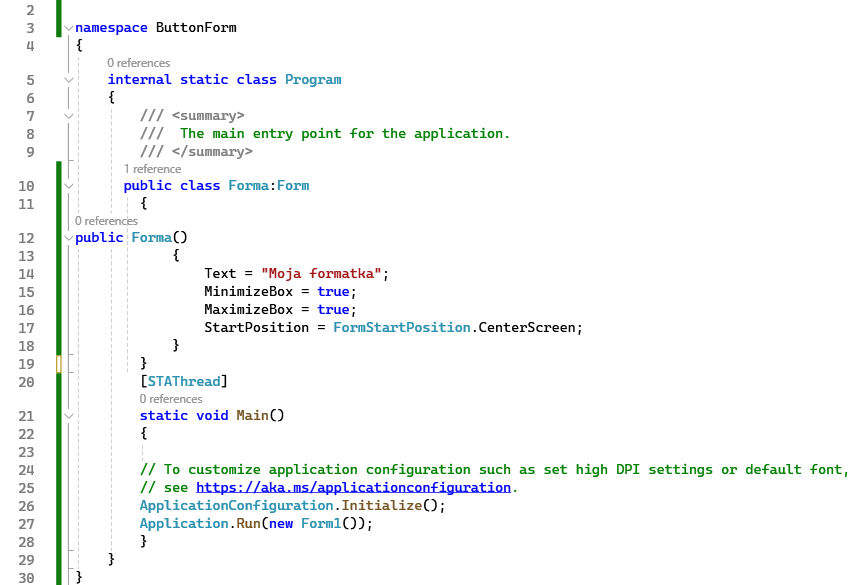
Ustawianie akcji w celu wyświetlenia zawartości pola.

Wyświetlenie tekstu zawartego w polu tekstowym jako treści komunikatu





Rozpoczęcie programu w Program.cs



**Etykiety (Label)**

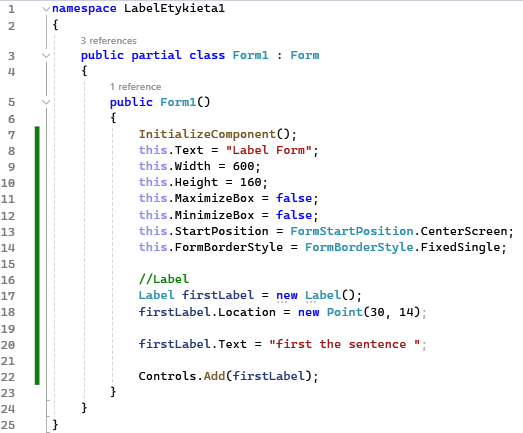
Etykieta pozwala opisać pole, do którego należy wprowadzić wartość. Osoby, które będą korzystać z naszego oprogramowania – muszą wiedzieć.

Dodawanie pola opisu

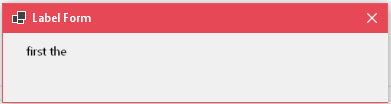
Pole opisu to przygotowany przez programistów opis pola, w którym należy wprowadzić wartość albo z którego należy ją wybrać. Etykiety można zastosować jako opis funkcjonalności w programie czy też ostrzeżenie przed wybraniem nieprawidłowej wartości w polu.

Aby uzyskać działający kod:

1. Określamy nazwę naszej aplikacji *(this.Text).*
2. Ustawiamy szerokość (*this.Width*) i wysokość (*this.Height*) formularza.
3. Ukrywamy przyciski do maksymalizacji *(this.MaximizeBox)* i minimalizacji *(this.MinimizeBox*).
4. Aplikacja powinna być wyświetlona na środku ekranu (*this.StartPosition*).
5. Wielkość aplikacji nie będzie można zmienić także przez przeciąganie jej aktywnych brzegów, dlatego musimy ustawić *this.FormBorderStyle* na obramowanie pojedyncze.
6. Definiujemy Etykietę przez przypisanie jej do obiektu klasy odpowiedzialnej za tworzenie etykiety: *Label firstLabel = new Label();*
7. Określamy położenie pola etykiety przez ustawienie w nim punktów położenia *XY(Location).*
8. Wpisanie tekstu, który będzie się wyświetlał w naszym polu, a, który ustawiamy we właściwości *Text*.
9. Na koniec dodajemy kontrolkę do naszego formularza z użyciem metody *Controls.Add(firstLabel)*, której pierwszym argumentem jest nazwa kontrolki.

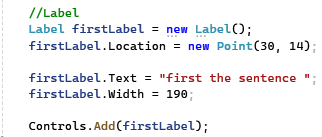


Wynik

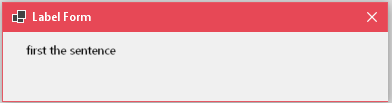


Ustawienie szerokości pola opisu

Uruchomiliśmy i już wiemy, że tekst w naszej etykiecie nie wyświetla się poprawnie. Nie widać końcówki tekstu. Zostało to spowodowane szerokością pola, w którym wpisaliśmy tekst. Musimy teraz wprowadzić jego szerokość tak, aby widoczny był cały tekst.



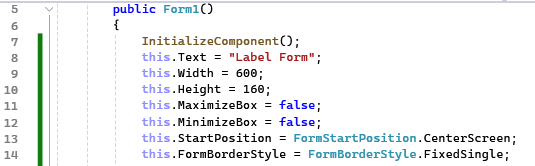
wynik

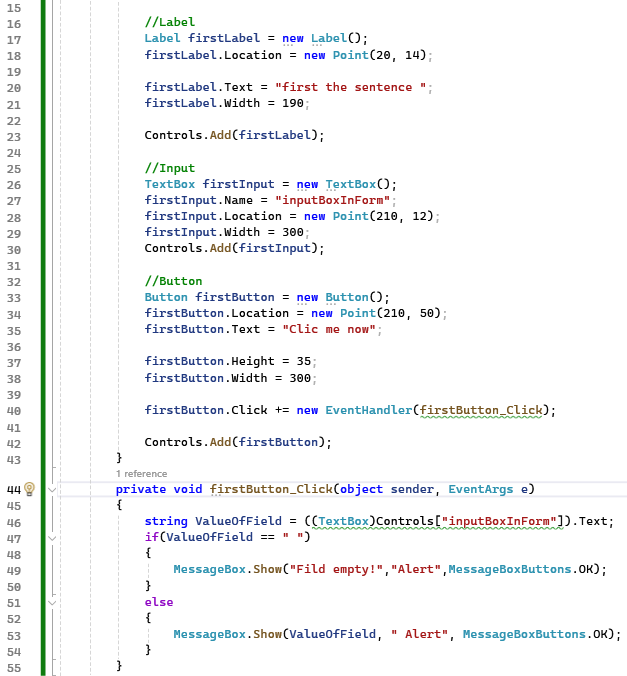


Czyli wpisujemy szerokość pola w celu wyświetlenia całego tekstu: *firstLabel.Width = 190;*

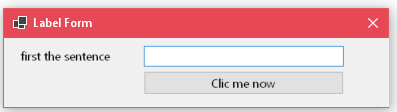
Definiowanie akcji dla pola

Jest to pole przeznaczone głównie do informowania na temat wartości, jaka ma się w nim znaleźć, w przykładzie użyję pola do wprowadzania tekstu.

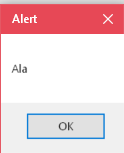




Wyniki



Po wpisaniu i kliknięciu:



20. W akcji kliknięcia przycisku dodajemy wywołanie metody, która to kliknięcie obsłuży: *firstButtonClick+=new EventHandler(firstButton\_Click)*

21. dodajemy przycisk do formularza *Controls.Add.*

22. Definiujemy metodę do obsługi kliknięcia przycisku. Będzie ona przyjmowała dwa parametry. Pierwszy będzie element, który go wysłał, a drugim błędy, które powstały w czasie jego wywołania oraz obsługi*: private void firstButton\_Click(object sender, EventArgs e)*

23.W metodzie pobieramy wartość pola z tekstem przez odwołanie się do kontrolki oraz wybranie z niej właściwości zawierającej tekst: *((TextBox)Controls[„inputBoxInForm”]).Text*

24. Następnie sprawdzamy czy wartość jest pusta. Jeśli tak, nakazujemy jej uzupełnienie przez wyświetlenie odpowiedniego komunikatu: *MessageBox.Show(„Field empty!”, Alert”, MessageBoxButtons.OK).*

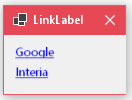
25. W przypadku gdy wartość w polu jest różna od pustej, wyświetlamy ją za pomocą komunikatu: *MessageBox.Show(ValueOfField, Alert!”, MessageBoxButtons.Ok)*

**Etykieta z odnośnikiem (LinkLabel)**

Etykieta z odnośnikiem jest tym samym co zwykła etykieta, tylko ma jedną bardzo ważną możliwość: kliknięcia i wykonania wybranej przez nas akcji.

**Utwórzmy nowy projekt:** *LinkLabelForm*



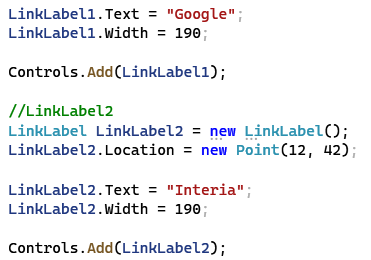


Aby uzyskać działający kod:

1. Określamy nazwę naszej aplikacji (*this.Text*).
2. Ustawiamy szerokość (*this.Width*) i wysokość (*this.Height*)
3. Ukrywamy przycisk do maksymalizacji (*this.MaximizeBox*) i minimalizacji (*this.MinimizeBox*).
4. Aplikacja po uruchomieniu powinna się znajdować na środku ekranu (*this.StartPosition*).
5. Wielkość aplikacji będzie można zmieniać także przez przeciąganie jej aktywnych brzegów, dlatego musimy ustawić *this.FormBorderStyle* na obramowanie pojedyncze.
6. Definicja etykiety: *LinkLabel LinkLabel1 = newLinkLabel();*
7. We właściwości Location ustawiamy położenie etykiety na formularzu.
8. Dodajemy tekst do etykiety (Text).
9. Dodajemy etykietę z odnośnikiem do naszego formularza (Controls.Add)
10. Definiujemy drugą etykietę – Location – Text - (Controls.Add). Powtórzenie czynności

Szerokość etykiety dopasowana do zawartości

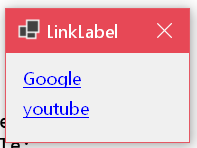
W przypadku niektórych tekstów nie mieszczących się w etykiecie wraz z łączem należy ustawić szerokość tej etykiety tak, aby tekst wyświetlał się w całości. Pozwala na to właściwość pola etykiet.

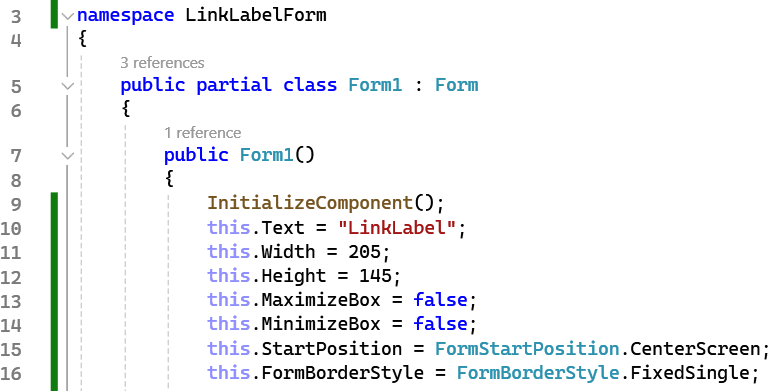


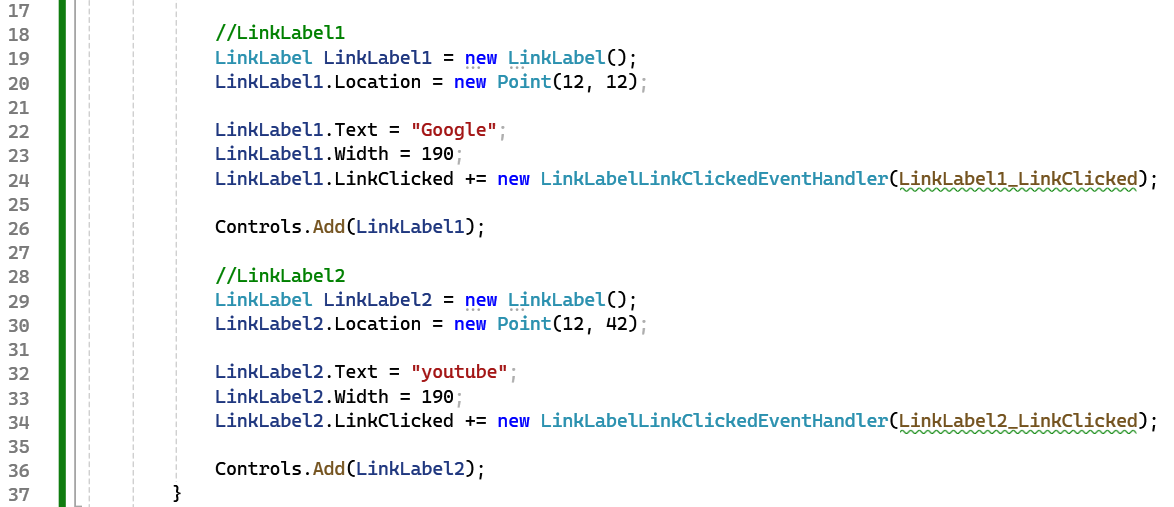
Ustawiona szerokość etykiety zarówno pierwszej jak i drugiej,np.: *LinkLabel1.Width = 190;*

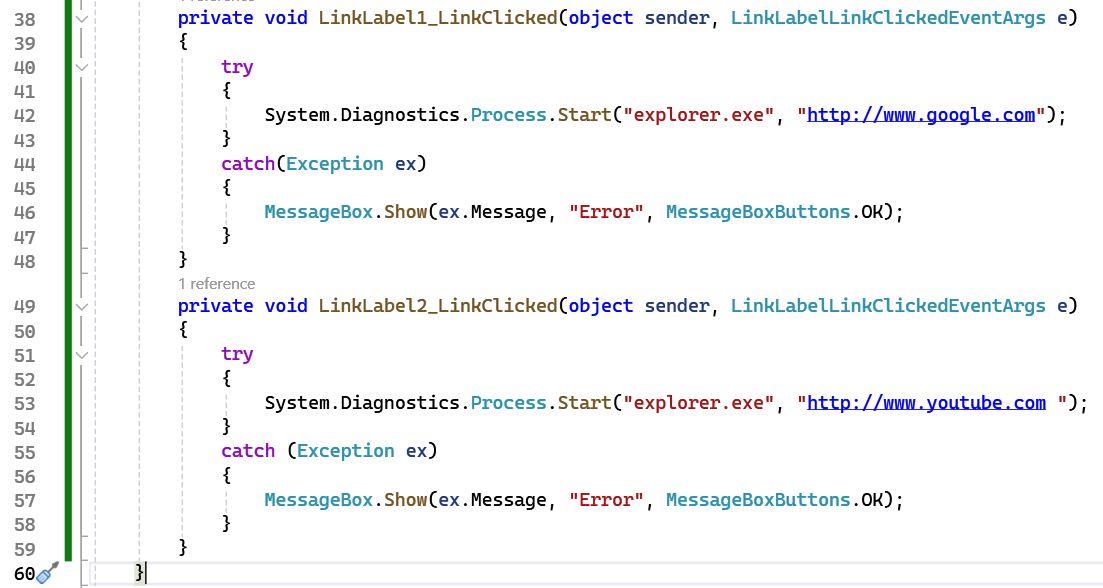
Przejście do strony po kliknięciu etykiety z łączem

Najważniejszy moment przy etykietach z łączem jest kliknięcie etykiety i wykonania wybranej akcji. Taką akcją jest na przykład otwarcie strony internetowej w domyślnej przeglądarce.









1. W każdej etykiecie : LinkLabel1 i LinkLabel2 musimy do pola dodać metodę obsługującą akcję kliknięcia etykiety:

*LinkLabel1.LinkClicked += new LinkLabelLinkClickedEventHandler(LinkLabel1\_LinkClicked)*;

*LinkLabel2.LinkClicked += new LinkLabelLinkClickedEventHandler(LinkLabel2\_LinkClicked);*

1. Obydwie etykiety dodajemy do formularza (Controls.Add)
2. Definiujemy metodę, która będzie wywoływana w wyniku kliknięcia pierwszej etykiety:

*private void LinkLabel1\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)*

1. To samo również możemy zrobić z drugą etykietą.
2. Wewnątrz obydwu metod wywołujemy opcję sprawdzania, czy dana czynność zostanie wykonane: try.
3. Wewnątrz instrukcji sprawdzającej definiujemy czynność, której zadaniem jest otwarcie strony w domyślnej przeglądarce internetowej:

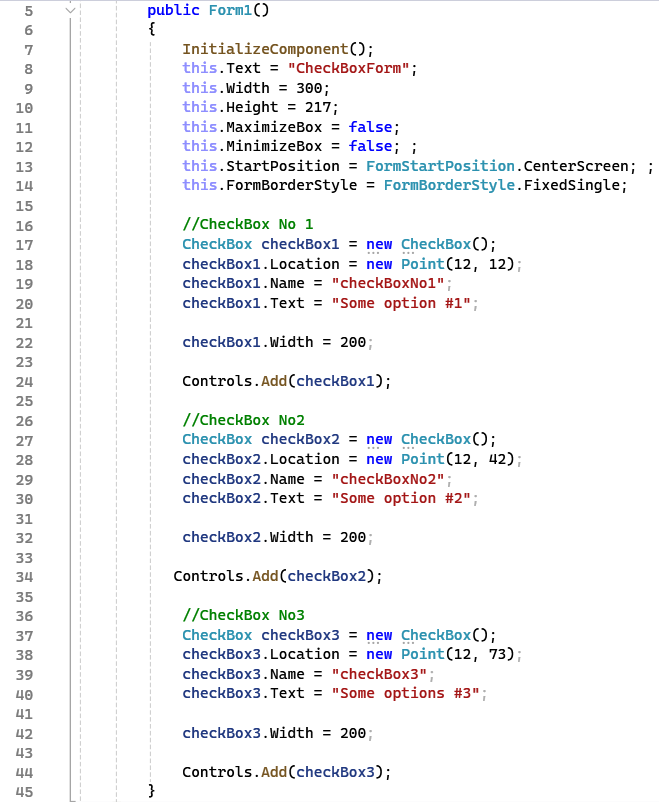
*System.Diagnostics.Process.Start("explorer.exe", "http://www.google.com");*

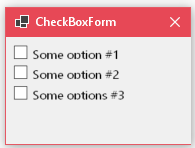
1. Jeżeli podcza wywoływania kodu, którego dotyczy poprzedni punkt, wystąpią problemy, wówczas pobierany jest błąd, wyświetlany następnie użytkownikowi jako okno z informacją:

*MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK);*

**Pola wyboru(Checkbox)**

Pola wyboru pozwalają na zdefiniowanie w formularzu opcji, których wybór chcemy zapewnić. Pola te są przydatne przy wybieraniu konfiguracji dowolnego programu.

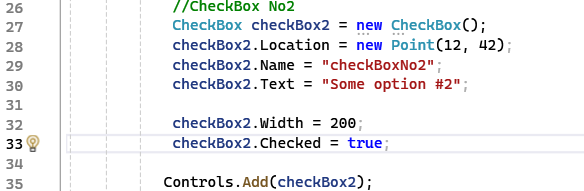


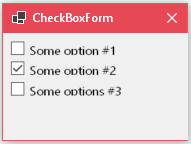


1. Do zmiennej przypisujemy klasę pola wyboru: *ChekBox chekBox1 = new CheckBox();*
2. We właściwości *Location* ustawiamy położenie pola na formularzu
3. Określamy nazwę pola, która będzie przydatna podczas odczytywania wartości (*Name*).
4. Ustawiamy tekst wyświetlany obok pola do zaznaczenia (*Text*).
5. Ustawiamy szerokość pola *Width.*
6. Dodajemy pole do formularza z użyciem metody (*Controls.Add*)
7. Powtarzamy czynność jeszcze dwa razy.

Zaznaczenie niektórych opcji wyboru

Pola przeznaczone do zaznaczania w formularzu możemy także zaznaczyć przed wygenerowaniem widoku pól formularza. W tym celu trzeba ustawić w odpowiedniej wartości true

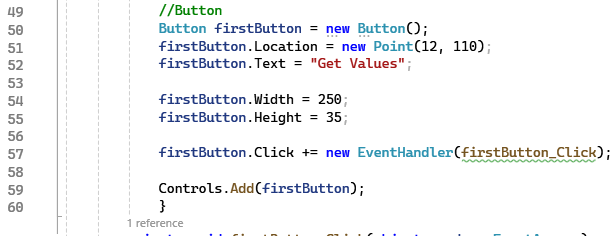




Ustawianie akcji dla pól wyboru

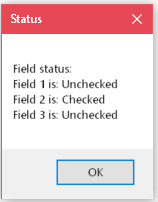
Najważniejszym elementem podczas definiowania pól wyboru jest ich obsługa. Dzięki temu będziemy mogli zapisać, które pole zostało wybrane i którego zaznaczenie zostało usunięte. Zapewni to możliwość ustawiania opcji za pomocą pól wybierania.











//Button

1. Tworzymy nowy przycisk przez przypisanie do zmiennej jego klasy:

*Button firstButton = new Button();*

1. Właściwość Location – ustawiamy położenie przycisku na formularzu.
2. Ustawiamy tekst na przycisku (Text).
3. Ustawiamy szerokość i wysokość Height i Width.
4. Przypisujemy do właściwości kliknięcia przyciski uruchomienie metody: *firstButton.Click += new EventHandler (firstButton\_Click).*
5. Dodajemy przycisk do formularza *Controls.Add*(firstButton);
6. Tworzymy nową metodę pozwalającą obsłużyć kliknięcie przycisku: *private void firstButton\_Click(object senders, Event Args e)*.
7. Umieszczamy elementy try, który w razie wystąpienia błędów w kodzie zostanie przekierowany do obsługi wyjątku.
8. Tworzymy trzy zmienne będące trzema ciągami znakowymi dla trzech pól wyboru, które mamy w formularzu.
9. Następnie sprawdzamy czy pole wyboru numer jeden z kontrolek ma wartość Checked, w przeciwnym razie przypisujemy wartość Unchecked.
10. Powtarzamy tę czynność dla każdego z pól.
11. Tworzymy ciąg, w którym będą zapisane wszystkie statusy pól w formularzu. Ciąg ten wyświetlamy w okienku dla użytkownika: MessageBox.Show(Info, ”Status”, MessageboxButton.OK.
12. W przypadku gdyby w tak utworzony kodzie wystąpi jakiś wyjątek przechwytujemy wyjątek, przechwytujemy wyjątek i wyświetlamy pochodzącą z niego informację użytkownikowi w okienku:

*MessageBox.Show(ex.Message, „Error”,MessageBoxButtons.OK)*

Lista z polem wyboru (CheckedListBox)

Dzięki różnym polom możemy tworzyć różne formularze z różnego rodzaju możliwościami wyboru z różnych pól. Takim polem jest między innymi lista z wyborem, w której mamy do dyspozycji zarówno zaznaczanie poszczególnych opcji przez postawienie obok wybranych ptaszka, jak i usunięcie zaznaczania.

Utworzymy nowy projekt o nazwie CheckedListForm.

Dodawanie listy do formularza

Aby dodać listę do formularza, musimy umieścić w kodzie wywołanie obiektu listy i przepisać go do specjalnej zmiennej. Dzięki dodaniu elementu listy z wyborem będziemy mieli możliwość jej dowolnej konfiguracji przed przesłaniem do wyświetlenia w formularzu

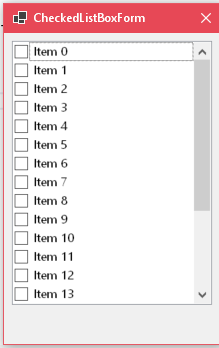


1. *this.Text = "CheckedListBoxForm";* - określamy nazwę naszej aplikacji.
2. *this.Width , this.Height* : ustawiamy szerokość i wysokość formularza
3. *this.MaximizeBox, this.MinimizeBox false* – ukrywamy przycisk do minimalizacji i maksymalizacji
4. *this.StartPosition = FormStartPosition.CenterScreen;* - aplikacja wyświetli się na środku ekranu
5. *this.FormBorderStyle = FormBorderStyle.FixedSingle*; - Wielkość (aktywne bżegi) aplikacji jest zablokowana.
6. *CheckedListBox checkedListBox1 = new CheckedListBox();* Wywołujemy zmienną wraz z przyciskiem nowego obiektu listy z wyborem.
7. Ustawiamy rozmiar tej listy (Size).orzonej przez n
8. We właściwości *Location* ustawiamy lokalizację utworzonej listy.
9. Określamy nazwę listy (Name).
10. Decydujemy, czy chcemy prezentować treści w wielu kolumnach. Właściwość *MultiColumn* ustawiamy na *false*.
11. Określamy czy będzie można zaznaczyć na liście więcej niż jedną pozycję. Chcemy , aby można było zaznaczyć tylko jedną, więc ustawiamy to we właściwości SelectionMode.
12. Teraz utworzymy pętlę,w której za pomocą kolejnych liczników będziemy dodawać nowe elementy do listy zawierającej tylko jedną kolumnę: checkedListBox1.Items.Add(„Item „+x.ToString()).

Wypełnianie listy z danymi

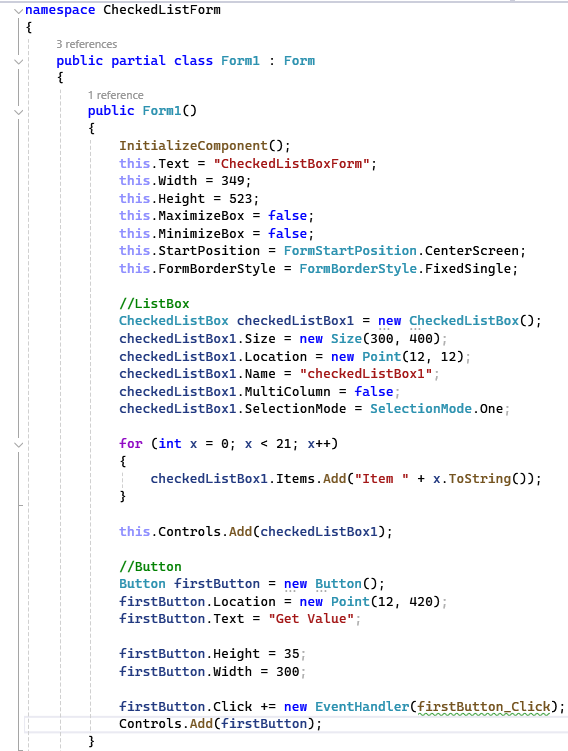
Kiedy lista z wielokrotnym wyborem, jest już stworzona, trzeba jeszcze uzupełnić danymi oraz wskazać, czy czy ma mieć wiele kolumn czy też jedną oraz czy umożliwia wielokrotny wybór.

1. *this.Controls.Add(checkedListBox);* -Dodajemy kontrolkę do formularza



Dodawanie akcji po wybraniu elementów.

Tworzymy akcję dla obecnej listy, gdyż bez pobierania elementów, które zostały zaznaczone, taka lista jest bezużyteczna. Musimy dodać przycisk, po którego kliknięciu zostaną wskazane wybrane przez nas elementy listy.





1. Tworzymy przycisk przypisujemy go do klasy obsługującej tworzenie kontrolki:

Button firstButton = new Button();

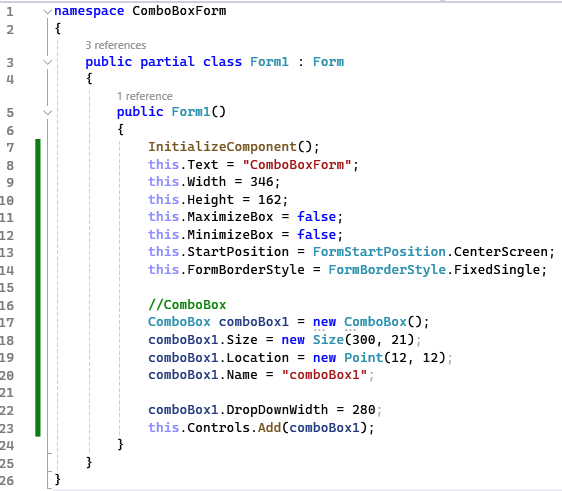
1. Właściwość Location określamy położenie przycisku
2. Dodajemy tekst wyświetlany na przycisku (Text).
3. Określamy wysokość oraz szerokość przycisku z użyciem wartości Height i Width.
4. Po kliknięciu powinna być wywołana metoda, która użyje zdefiniowanej przez nas w pierwszym parametrze akcji: *firstButton.Click+=new EventHandler(firstButton\_Click).*
5. Na koniec dodajemy przycisk do formularza za pomocą metody Controls.Add
6. Teraz definiujemy metodę, która zostanie użyta po kliknięciu przycisku. Metoda ma dwa parametry: obiekt wysyłający przycisk oraz błędy, które powstały w czasie tworzenia oraz obsługi metody: firstButton\_Click(object sender, EventArgs e).
7. Wewnątrz metody dodajemy instrukcję pozwalającą sprawdzić, czy podczas przetwarzania kodu wystąpił błąd: try.
8. Następnie sprawdzamy czy którekolwiek z pól dostępnych przy pozycjach w liście zostało zaznaczone: *if (((CheckedListBox)Controls["checkedListBox1"]).CheckedItems.Count > 0)*. Jeżeli tak, to wówczas wykonujemy akcję wyświetlenia poszczególnych pozycji w liście.
9. Akcja ta rozpoczęta jest pętlą, w której z każdego elementu dostępnego w liście jest pobierana właściwość informująca o zaznaczeniu elementu listy: *forech (object itemChecked in((CheckedListBox)Controls[„checkedListBox1”).CheckedItems).*
10. Jeżeli element został zaznaczony, wówczas w pętli wyświetlany jest komunikat o nim
11. W przypadku gdy żaden z elementów nie został zaznaczony, wyświetlamy komunikat informujący o tym użytkownika: MessageBox.Show(„None of the items are checked.”, „Checked”,MessageBox.Show(ex.Message, „Error”, MessageBox Buttons.OK).
12. Na końcu wyświetlamy komunikat o ewentualnym błędzie: *MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK);*

Lista (ComboBox)

Do formularza możemy dodać zwykłalistę, która będzie listą wybierania. Lista ta będzie dostępna z możliwością wpisywania części wartości dostępnej w opcjach lub listą, którą będzie można rozwijać, by następnie wybierać z niej właściwą wartość.

Utwórzmy nowy projekt o nazwie **ComboBoxForm**.

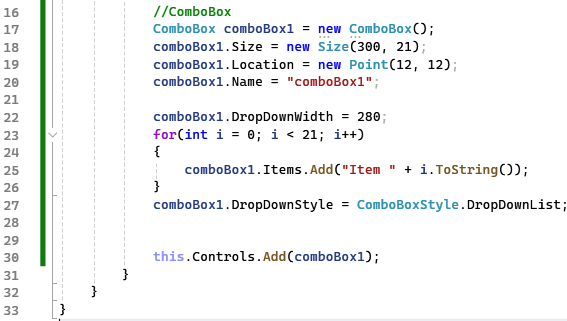
Tworzymy listę rozwijaną



1. Określamy nazwę, ustawiamy szerokość, oraz wysokość, ukrywamy przyciski maksymalizacji i minimalizacji, aplikacja będzie wyświetlać się na środku ekranu, blokujemy granice.
2. Tworzymy zmienną obiektową o nazwie comboBox1, określamy rozmiar, lokalizację, określamy nazwę list,
3. Ustalamy szerokość listy, w której będzie można wpisać nazwy poszukiwanych elementów. (DropDownWidth)
4. Dodajemy listę do formularza (Controls.Add).

Uzupełnianie listy danymi oraz zmiana jej wyglądu.

Lista pozostaje pusta, lista może przyjmować dowolną wartość wpisaną przez użytkownika z klawiatury, jednak będziemy chcieli, aby można było z niej wybrać wyłącznie elementy zdefiniowane na samym początku.



Zadania

Zadanie 1.

Wyświetl okno z dowolnym komunikatem, które pojawi się przed dowolnym komunikatem.



Zadanie 2.

Napisz program wyświetlający okno główne. Przy próbie zamknięcia aplikacji wyświetl okno z komunikatem.

