ZAAWANSOWANE PROGRAMOWANIE
LAB02: KOMUNIKACJA MIĘDZY OKNAMI
KOMUNIKACJA PRZEZ DELEGACJĘ

Data: 22.03.2025 v1.4

WYKŁAD:

http://www.cs.put.poznan.pl/mradom/resources/lectures/ZaawansowaneProgramowanie/WindowsFormsApp_MultiWindowsAndDelegates.zip

LABORATORIUM

Uniemożliwienie zamykania okna:

Aby zapobiec zniszczeniu obiektu typu Form po wciśnięciu przycisku zamykania okna w prawy górnym rogu każdego okna, należy dla danego okna programu (Form) przejść do jego widoku Zdarzeń (Event, ikona błyskawicy w polu Właściwości (Properties, prawe dolne podokno VS)) a następnie wybrać właściwość FormClosing (CLOSING, nie CLOSED!) i w jej kodzie umieścić dwa polecenia:

```
this.Hide();
e.Cancel = true;
```

Ogólnie kod wygląda wtedy tak:

```
private void FrmDialog_FormClosing(object sender, FormClosingEventArgs e) {
    this.Hide();
    e.Cancel = true;
}
```

PROJEKT Z LINKU - KOMUNIKACJA POMIĘDZY OKNA PROGRAMU

Zadanie: stworzenie projektu komunikacji delegatami opisanego na stronie:

https://www.c-sharpcorner.com/article/using-delegates-to-communication-between-windows-forms/

Kilka uwag:

Program tworzy kilka okien. W przypadku dwóch górnych przycisków: „Open Child Form 1” oraz „Open Child Form 2” kod je tworzoący wygląda tak:

< code01: >

```
FrmChild1 frm = new FrmChild1();
this.SetParameterValueCallback += new SetParameterValueDelegate(frm.SetParamValueCallbackFn);
frm.Show();
```

Aby go zrozumieć, należy zauważyć, że główne okno programu, to, w którym mamy między innymi te dwa przyciski definiuje obiekt delegata oraz w kolejnej linii, tworzy jego obiekt (na początku pusty):

< code02: >

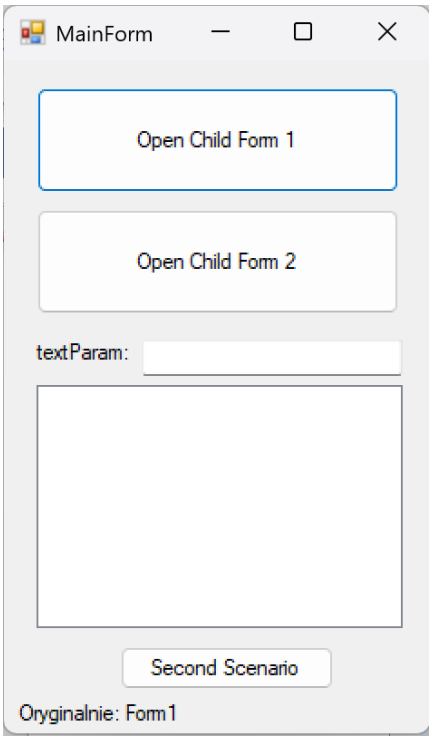
```
public delegate void SetParameterValueDelegate(string value);
public SetParameterValueDelegate SetParameterValueCallback;
```

W obu oknach typu “Child Form” znajduje się metoda:

< code03: >

```
public void SetParamValueCallbackFn(string param) {
    txtParam.Text = param;
}
```

Która ma za zadanie wpisać to textBoxa o nazwie txtParam tekst, który przyjdzie jako argument.



Ostatnia rzecz do zauważenia: z textboxem (pośrodku okna pokazanego po lewej) związana jest motoda-event (zdarzenie) która aktywuje się za każdym razem, gdy cokolwiek wpiszemy wewnątrz tego textBoxa o nazwie „textParam”:

< code04: >

```
private void textParam_TextChanged(object sender, EventArgs e)
{
    SetParameterValueCallback(textParam.Text);
}
```

To są wszystkie „klocki” potrzebne do komunikacji. Ta wygląda następująco:

- Najpierw główne okno przygotowuje definicję i obiekt delegata odpowiedzialnego za komunikację: **<code02>**
- Potem, gdy klikniemy „Open Child Form 1” oraz „Open Child Form 2” pokażą się nowe okna. Jedna to jako ostatnia rzecz, bo najpierw : **<code01>** w drugiej linii stworzy prawdziwy obiekt delegata z głównego okna (**<code02>**), a ten delegat będzie zawierać metodę z **<code03>** z okien Child1 i Child2. Finalnie, gdy wpiszemy cokolwiek do textBoxa na prawo od etykiety „textParam:”, odpali się **<code04>** który:
    - Uaktywni obiekt delegata `SetParameterValueCallback` z parametrem będącym tekstem aktualnie wpisanym w textParam, powiedzmy, że jest to „abc”
    - Ponieważ ten delegat zawiera metody **<code03>** z okien Child, wyśle on (delegat) do nich (metod z Child) tekst „abc”.
    - Metody te w swoich oknach wpiszą to do textBoxów w tychże oknach typu ChildForm.

Zadanie: stworzyć trzecie okno typu ChildForm, które będzie się zachowywać tak samo jak dwa już istniejące.

Całość jest sposobem komunikacji pomiędzy oknem głównym, które coś wysyła do okien potomnych. A teraz odwrotna sytuacja:

KOMUNIKACJA Z OKNA POTOMNEGO DO OKNA GLÓWNEGO

Przycisk „Second Scenario” tworzy okno Form4 o nazwie FrmDialog. Okno to ma swojego delegata:

```
public delegate void AddItemDelegate(string item);
public AddItemDelegate AddItemCallback;
```

Który jest wywoływany przyciskiem „AddItem” w Form4:

```
private void button1_Click(object sender, EventArgs e) {
    AddItemCallback(txtItem.Text);
}
```

Czyli to co wpiszemy w txtItem, idzie jako tekst-argument do delegata `AddItemCallback` . Ten jednak musi zawierać jakaś metodę, żeby to zadziało. I zawiera, jest to oprogramowane kodem przyciski „Second Scenario” głównego okna:

```
private void btnScenario2_Click(object sender, EventArgs e) {
    FrmDialog dlg = new FrmDialog();
    dlg.AddItemCallback = new FrmDialog.AddItemDelegate(this.AddItemCallbackFn);
    dlg.ShowDialog();
}
```

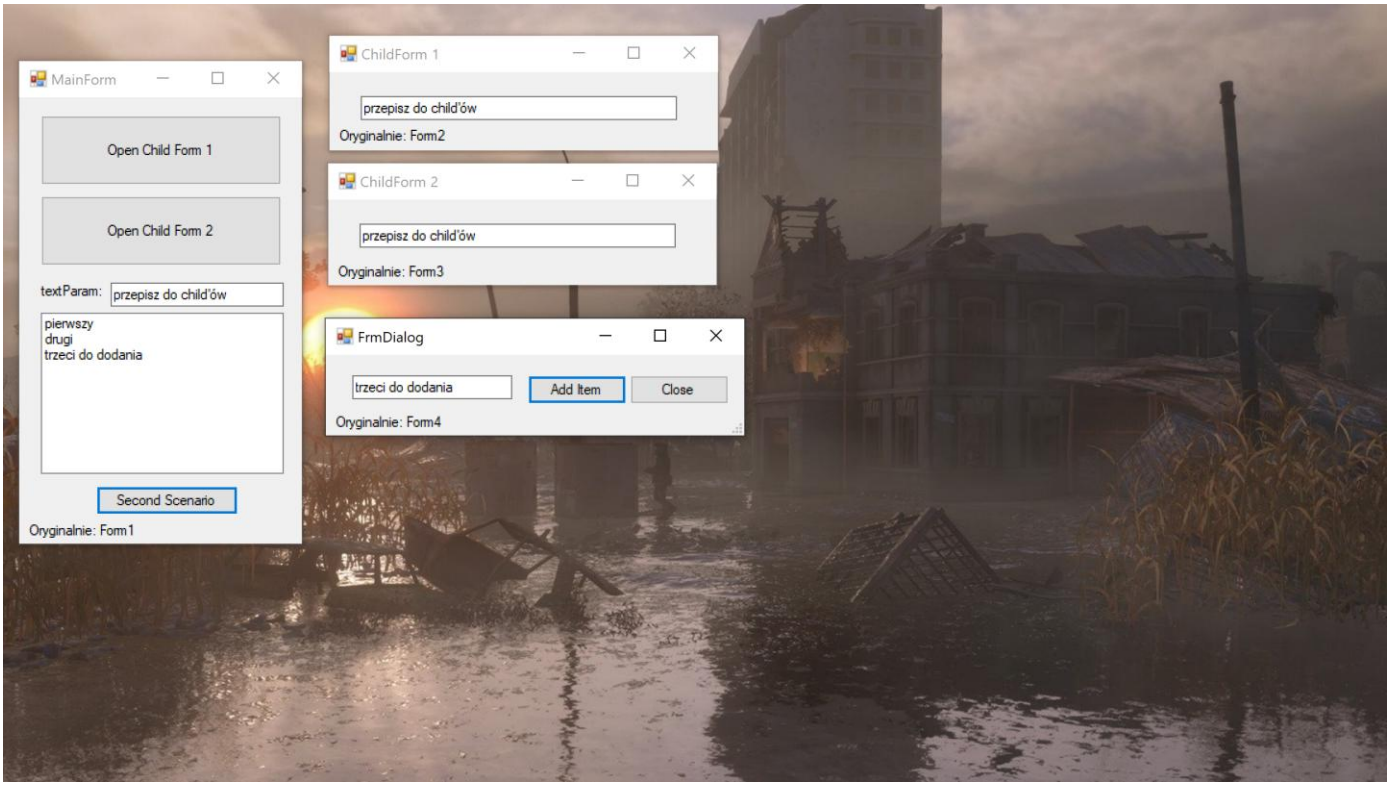
Czyli tutaj w drugiej linii: tworzony jest delegat okna FrmDialog (ten o nazwie `AddItemCallback`) i wkładana jest do niego metoda `AddItemCallbackFn`. Ona z kolei należy do kodu głównego okna:

```
private void AddItemCallbackFn(string item) {
    lstBx.Items.Add(item);
}
```

I ma za zadanie włożyć tekst-argument jako kolejny element listy lstBx w głównym oknie. Czyli: wpisanie czegokolwiek w textBoxie okna potomnego FrmDialog i wciśnięcie w nim przycisku „Add Item” powoduje wywołanie delegata `AddItemCallback` z argumentem `txtItem.Text`, a ten delegat posyła ten tekst-argument do metody którą w sobie zawiera czyli `AddItemCallbackFn` z głównego okna.

Zadanie: spróbować dodać w oknie FrmDialog przycisk „Delete Item”, który taką samą metodologią komunikacji spróbuje usunąć dany tekst z listy lstBx głównego okna linijkę tekstu, o ile ona istnieje. Czyli jak na liście jest np. linia „usunTo”, to chodzi o to, żeby wpisać w textBox okna FrmDialog „usunTo”, wcisnąć nowy dodany teraz przycisk „Delete Item”, a to odpali delegata z metodą, która postara się tą linię usunąć.

Czemu się tak męczymy – stanie się to jasne na kolejnym wykładzie/laboratorium. Najkrótsze wyjaśnienie: gdy stworzony zostanie drugi (trzeci, czwarty, kolejny) wątek programu, to komunikacja z wątkiem głównym odbywać się będzie musiała przez mechanizm delegacji. Nie dlatego, że jak tak chcę, ale dlatego, że inaczej się nie da, bo .NET i C# są tak skonstruowane…



PROPONOWANE ZADANIE NA LABORATORIUM

Przygotować wstępną wersję interfejsu programu – albo wielookienkowego, albo w zakładkach. Na bazie tego drugiego sposobu:

- Przygotować zakładkę nr 1: dane wejściowe instancji problemu, ich edycja, zapis/odczyt do plików, itd.
- Zakładka nr 2: parametry wybranego algorytmu metaheurystycznego
- Zakładka nr 3: prezentacja wyników. (Komentarz: na tym etapie, tj. na początku semestru, niekoniecznie można mieć tu dobre pomysły, więc przede wszystkim koncentrowałbym się na zakładach/oknach nr 1 i 2).

Panele – do poćwiczenia jak może wyglądać główny projekt z przedmiotu

Program ma mieć 3 zakładki, w każdej po dwa panele – górny (tryb: Top) i dolny (tryb: Dock). Zakładka nr 2 aktywuje się dopiero gdy wciśniemy odpowiedni przycisk w Zakładce 1 (np. przycisk „Akceptuj dane”). Zakładka nr 3 aktywuje się na przycisk w zakładce nr 2 (np. „Uruchom algorytm”). I odwrotnie - aktywacja zakładki nr 2 sprawia, że przestaje być "klikalna" zakładka nr 1 (analogicznie: aktywacja zakładki 3 - przestaje być klikalna Zakładka 2).

W zakładkach stworzyć kilka nic nierobiących innych przycisków, textboxów, itd. Napisać program tak, aby aktywacja i dezaktywacja komponentów odpowiedniej zakładki odbywała się za pomocą konkretnej metody np. **void** StatusZakladki1(**boolean** enabled) {} i **void** StatusZakladki2(**boolean** enabled) {}

Wciśnięcie przycisku KONIEC w zakładce 3 przywraca początkowy stan programu, gdy aktywna i klikalna jest tylko zakładka 1.