C#
Primul meu program
Visual Studio .NET 2022
Community Edition

Prefață
- Mediul de programare Visual Studio: instalare, aspect general

Aspectul și comportarea de bază ale mediului de programare Visual Studio
- Stabilirea unor opțiuni pentru scrierea de cod, pentru crearea programului ș.a.

Editor de text, asemănător cu Notepad
- Mediul de programare Visual Studio: crearea unui nou Proiect (program integral/parte a unuia); *Evenimentul implicit* al unui Control
- Controale: *Button*, *TextBox*, *OpenFileDialog*, *SaveFileDialog*
- Entități: *MessageBox*, *StreamReader*, *StreamWriter*; *namespace*-ul *System.IO*

Cronometru
- Mediul de programare Visual Studio: secțiunea *Properties*-Proprietățile unui Control
- Controale: *Timer*, *Label*
- Entități: *DateTime*
- Locul în computer unde se află programul; versiunile *Debug* și *Release*

Evaluarea unor variabile numerice și interacțiunea lor cu valori de tip text
- Entități: *Arrays* (tablouri de valori); blocul *if-else*
- *Transformarea/convertirea/conversia* unei variabile de tip text într-una de tip numeric

Crainic
- Entități: blocul *switch-case*

Meniuri
- Controale: *MenuStrip*
- Interacțiunea cu o casetă de dialog *MessageBox* în funcție de ce anume prezintă aceasta
- Închiderea din cod a programului

Accesarea unor părți ale sistemului de operare Windows
- Detalierea funcționalității casetei de dialog cu utilizatorul *MessageBox*
- Entități: *namespace*-ul *Environment*; Directoare Speciale (My Documents ș.a.); *namespace*-ul System. Diagnostics
- Pornirea altui program de pe computer și apelarea pentru deschidere a unor fișiere

Operațiuni realizate la anumite intervale
- Controale: *Timer*
- Afișarea periodică a unui mesaj dinamic – adaptat la valoarea unei variabile

Imagini
- Afișarea unor imagini aflate pe computerul unde rulează programul și a unora aflate pe Internet
- Controale: *PictureBox*
- Entități: *namespace*-ul *System.IO*



2023.04 – 2023.09, ©SINAPSA Infocomplex®™ http://sinapsaro.ro / Victor Emanuel Ștefănescu

Colecţii

- Entităţi: *namespace*-ul *System.Collections.Specialized;* tipul *NameValueCollection*
- Controale: *ComboBox*; forma *pointer*-ului deasupra unor Controale

Operaţiuni frecvente cu text (String)

- Secţionarea unui text după (în funcţie de) un caracter (o literă) anume
- Entităţi: *Substring*; *LastIndexOf*; *Count*; *Contains*

Paradoxul logic al Mincinosului (sau al lui Epimenides)

Controlul ListView

- Prezentarea în format tabelar a unor informaţii

Controlul TreeView

- Prezentarea în stil arborescent unor informaţii ramificate pe/după anumite criterii; salvarea în – şi încărcarea lor – dintr-un fişier
- Controale: *TreeView*; *MonthCalendar*; *RadioButton*; *GroupBox*

Modificarea interfeţei

- Schimbarea aspectului unui Dialog*Form* în timpul rulării programului

Recapitulare (Reamintire)

Construirea unui text folosind particule alese la întâmplare

- Entităţi: *Random*
- Controale: *ListBox*

Colecţii de String-uri: *StringCollection*

- Entităţi: *namespace*-ul *System.Collections.Specialized;* tipul *StringCollection*

Fişiere conţinând text în format XML

- Entităţi: *namespace*-ul *System.Xml*; obiectele: *XmlDocument*, *XmlNodeList;*

Biblioteca XML

Biblioteca – bază de date SQL Server

Grafică – analiza simplă a unor puncte dintr-o imagine

- Entităţi: *namespace*-ul *System.Drawing*
- Controale: *ImageList*

Recapitulare (Reamintire)

Visual Studio - Taste utile şi combinaţii utile de taste

Folosirea claselor: porţiuni de cod cu funcţionalitate distinctă de a altora

Controlul CheckedListBox

Duplicarea unor fişiere (Fişiere de rezervă)

- Controale: *StatusStrip*, *CheckedListBox*

Biblioteca – bază de date SQL Server- separarea unor înregistrări: LIKE

- Controale: *TabControl*, *ListBox*
- Entităţi: *SqlDataReader*

Crearea unei aplicaţii de tipul / pentru Consolă

- Etichete pentru controlarea fluxului programului

Corespondenţa între tablouri

- Entităţi: *DataSource* pentru un Control de tipul *ComboBox*
- crearea în cod a unui Control de tipul *SaveFileDialog* şi folosirea sa

Prefaţă - mediul de programare Visual Studio

Acestea sunt exemple de Prim Program De Un Anumit Fel.
Un program cu adevărat Prim, Cel Dintâi, poate fi unul elementar,
în care să fie creată o singură metodă (funcţie) care să adune două numere şi să întoarcă suma lor.

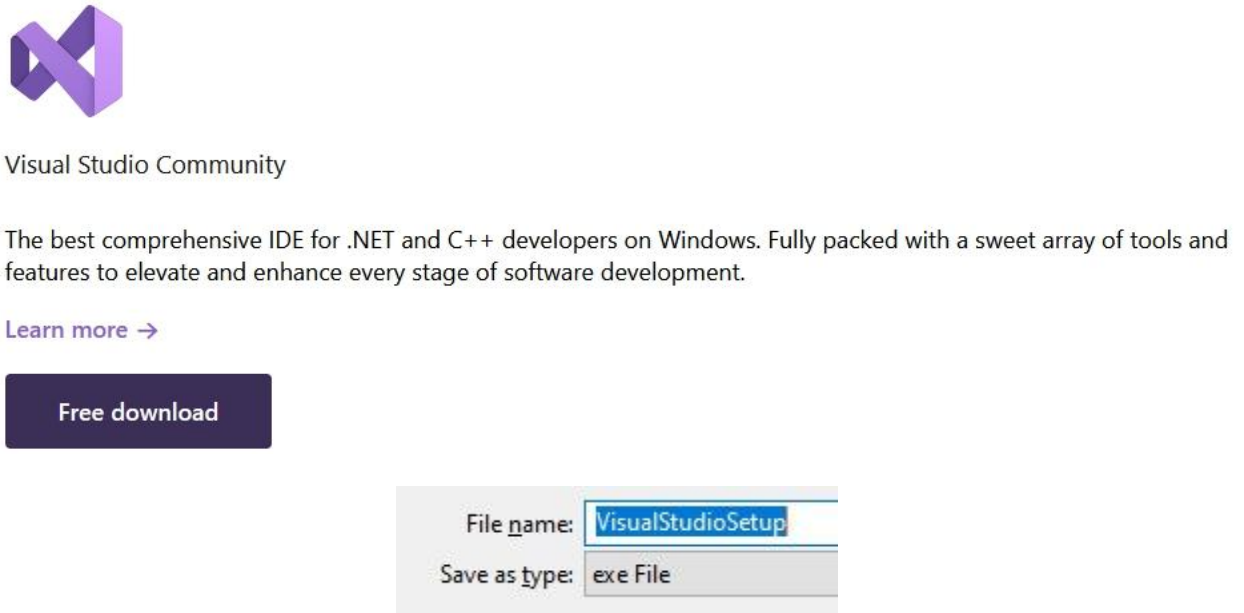
Vor fi create aplicaţii de tip *desktop* – adică, programe alcătuite din *Form*-e, ori "ferestre de dialog", ori "dialoguri" – regiuni (de obicei, dreptunghiulare – contur care asigură loc pentru cât mai mult/ă conţinut/ informaţie pe o suprafaţă cât de mică posibil) pe ecran care implică interacţiune cu utilizatorul – aşa cum sunt programele din pachetele Office şi Adobe, familiare utilizatorului obişnuit.

Începem cu prezentarea mediului de programare, de construire a unei aplicaţii de orice fel (*desktop*, *web application* ş. a.). Anume, Visual Studio, ediţia Community 2022. "*Community*" şi "*Express*" arată că versiunile astfel marcate ale mediului de programare Visual Studio sunt gratuite, având funcţionalitate redusă (nu cu mult, totuşi) în comparaţie cu ediţiile complete care nu au în denumire aceste particule; ediţiile *Community* şi *Express* pot fi preluate de pe site-ul Microsoft, instalate şi folosite cu rezultate identice cu ale ediţiilor complete.

Cei care nu vor să folosească acest mediu de programare pot folosi alte asemenea unelte – de asemenea gratuite, de exemplu "*SharpDevelop*" (https://sharpdevelop.en.softonic.com/) .
Totuşi, se recomandă ca, dacă se intenţionează crearea unui program care să ruleze pe sistemul de operare *Windows*, atunci să fie creat cu o unealtă de la aceeaşi companie (fiindcă, exact, ei ştiu cel mai bine cum interacţionează şi cum trebuie să fie proiectate componentele uneltelor făcute de ei; deci, nu din alt motiv, ci din acesta pur logic).

Ediţia Community 2022 a Visual Studio poate fi preluată de aici (în 2023.04.24):

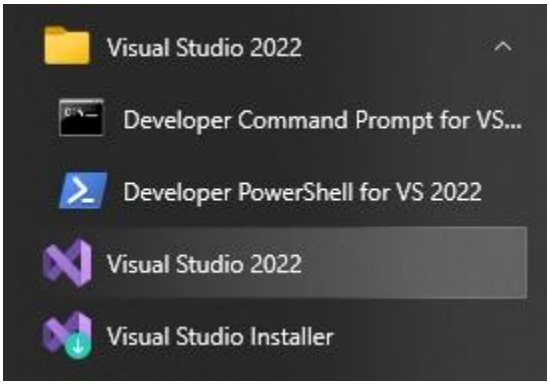
https://visualstudio.microsoft.com/free-developer-offers/



Acesta este, de asemenea, un program de tip *aplicaţie* ***desktop***, deci unul care implică proceduri efectuate (intenţionat) de către utilizator cu ajutorul unor Controale plasate pe interfaţă (adică, pe diversele Dialoguri – fie şi unul singur – care compun aplicaţia).

Instalarea se face lansând/pornind executabilul astfel preluat de pe acel site şi selectând-bifând - în dialogul principal care va fi afişat după descărcarea mult mai marelui pachet de instalare, conţinând mediul de programare în sine - componentele dorite, de exemplu cele necesare creării de aplicaţii .NET.

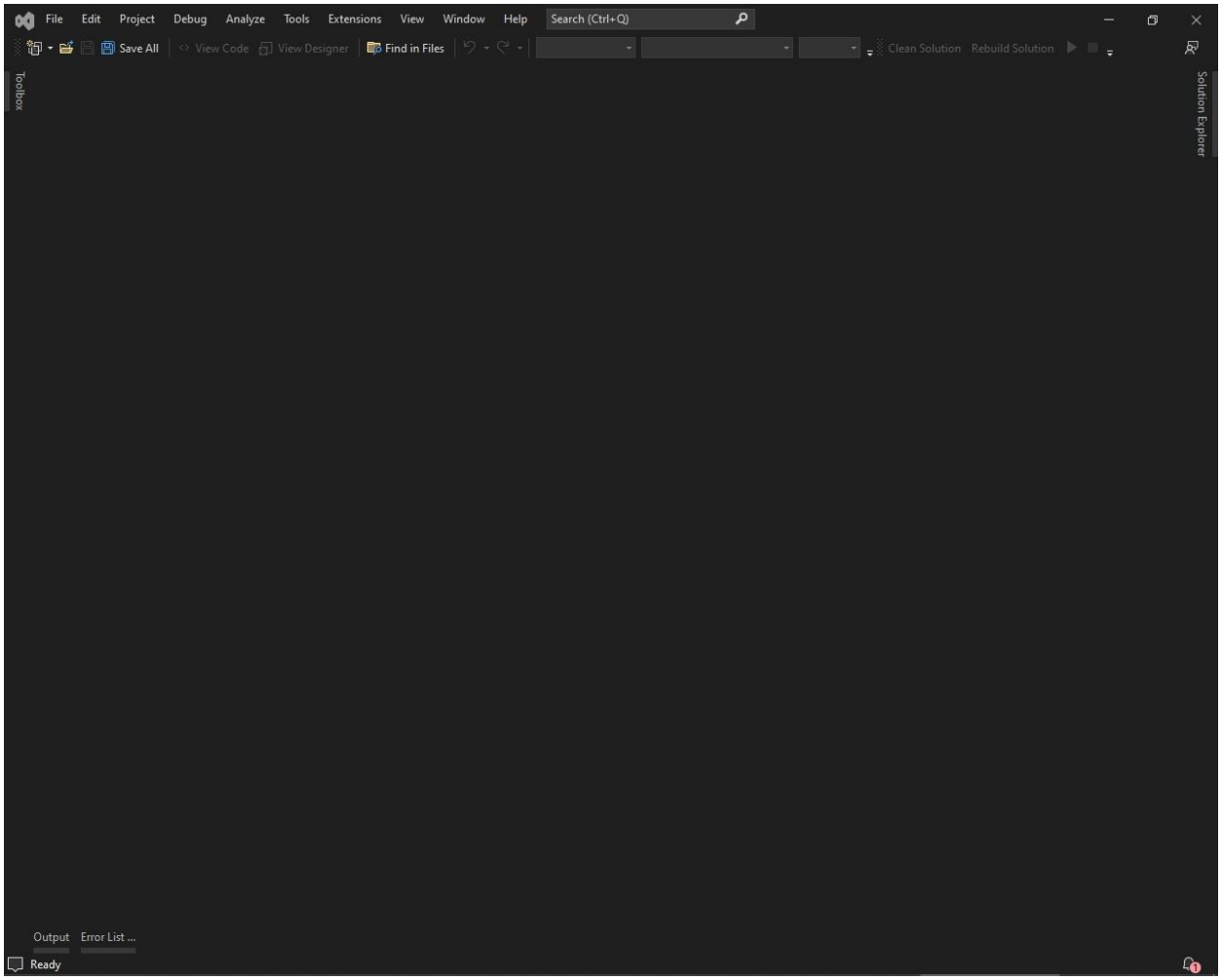
Odată instalat, Visual Studio va fi disponibil pentru utilizare sub *Windows*, unele din componentele sale instalate apărând astfel (din butonul *Start*) într-o ediţie a *Windows 10*:



Acum se poate purcede la crearea unui program.
Se lansează Visual Studio:

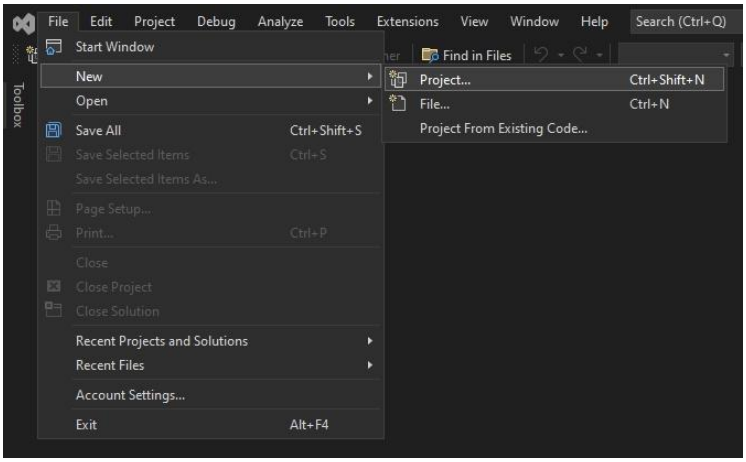


Imediat după pornire, dialogul care va fi afişat poate fi aşa-numita *Start Page*, unde sunt prezentate diverse resurse utile. Pagina aceea poate fi închisă, rămânând una goală de conţinut:



Această pagină, însă, cea principală a mediului de programare, oferă tot ceea ce trebuie pentru a construi o aplicaţie în toată regula.

Anume, printre altele, în intrarea de meniu *File* sunt comenzi similare celor din imaginea următoare, dintre care cele mai importante sunt "*New... -> Project...*" şi bineînţeles *Exit*.

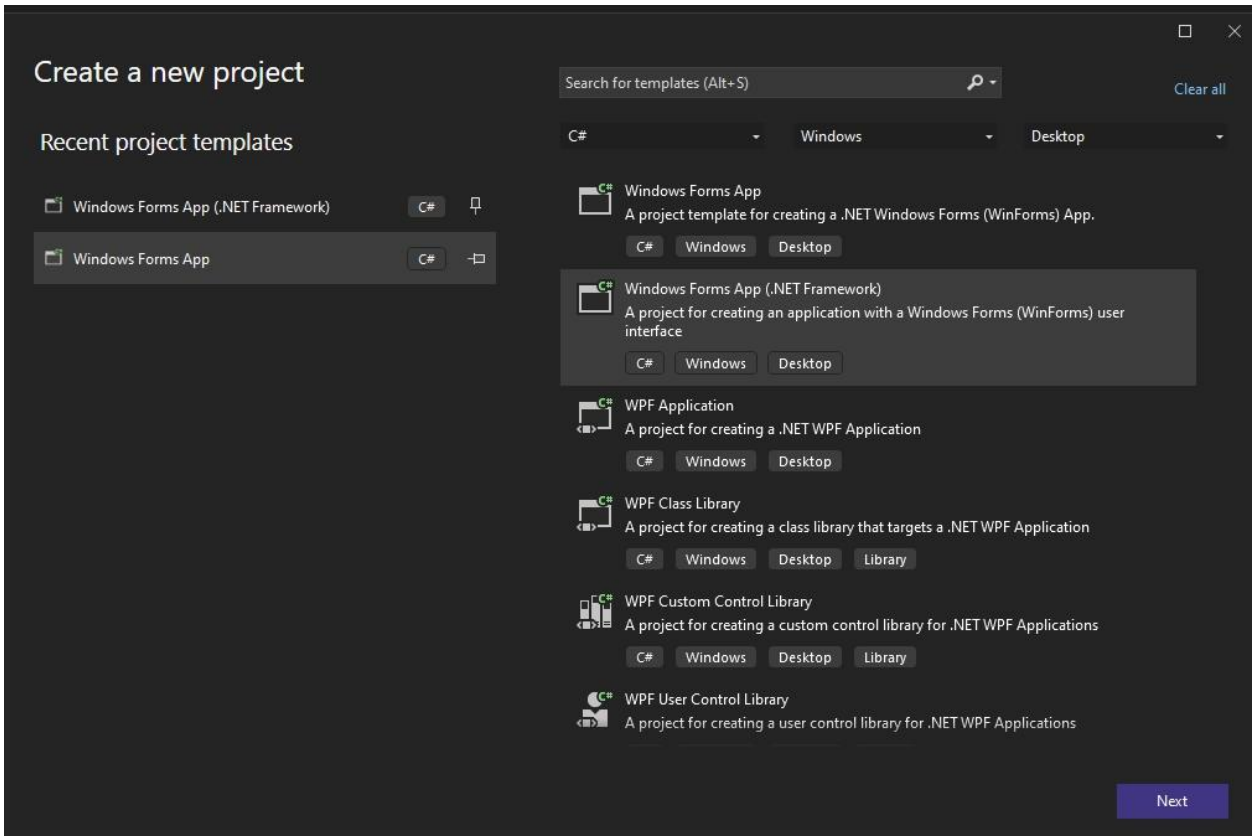


Începem, aşadar, să proiectăm şi să scriem programele noastre.

Click pe *New... Project...*

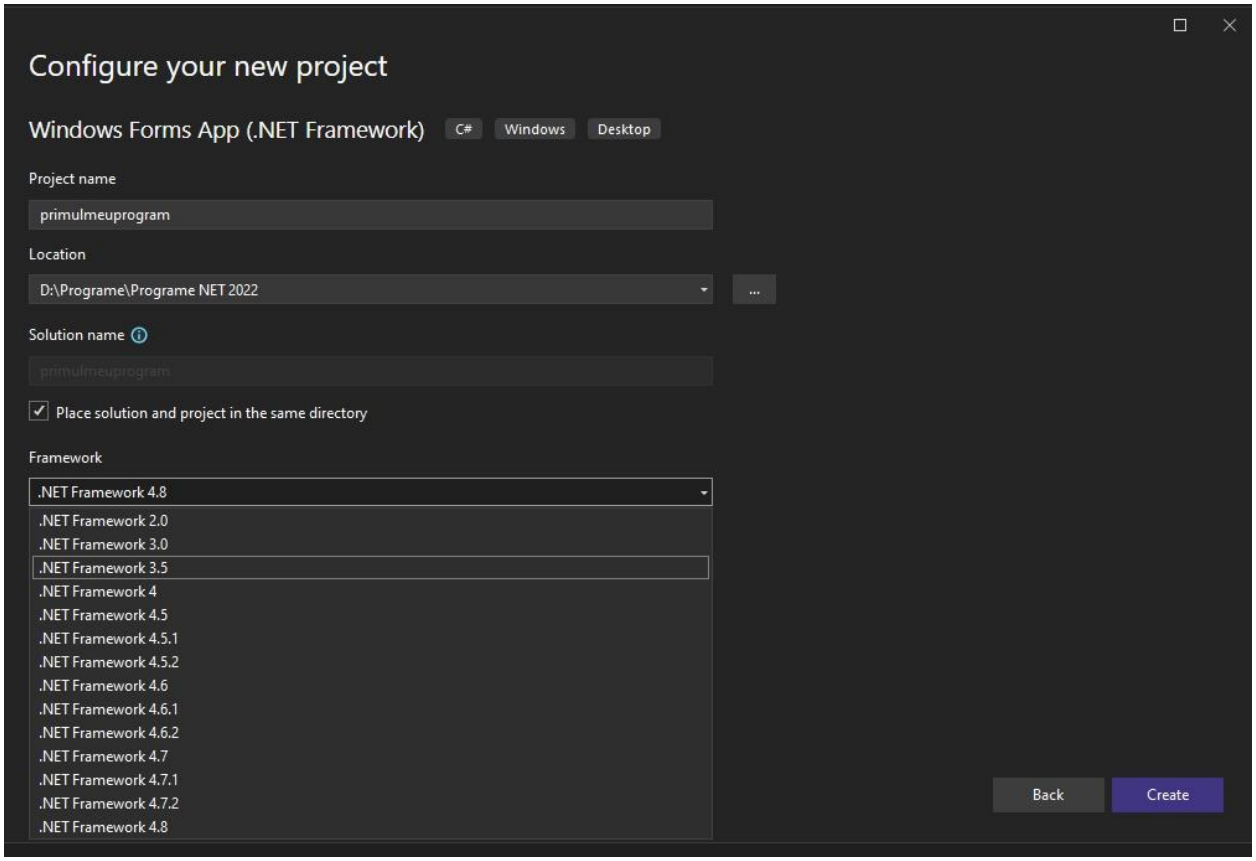
Din oferta de tipuri de aplicaţii se alege astfel: în stânga, "Windows Forms App" ; în dreapta, "Windows Forms App (.NET Framework)".

Deci, vrem să construim o aplicaţie (un program) de tip *Desktop* (interacţiune cu utilizatorul), bazată pe platforma .NET, scrisă în limbajul C# şi destinată sistemului de operare *Windows*:



În dialogul următor se solicită denumirea aplicaţiei, locul (directorul, folderul) unde să fie stocate fişierele sale componente şi versiunea platformei .NET cu care va fi construită aplicaţia.

Teoretic, e bine/recomandat să fie selectată o versiune "medie" a .NET – pentru a asigura funcţionarea aplicaţiei pe sisteme de operare mai vechi decât cel pe care este construită şi sisteme de operare care conţin componentele acelei versiuni de .NET. De exemplu, 3.5 este prezentă sau poate fi instalată pe *Windows* de la versiunea *Vista* încolo şi este prezentă implicit în unele versiuni ulterioare acesteia. Pentru versiunea 10 a *Windows* pot fi alese versiuni de la 4.0 sau chiar 4.5 încolo.

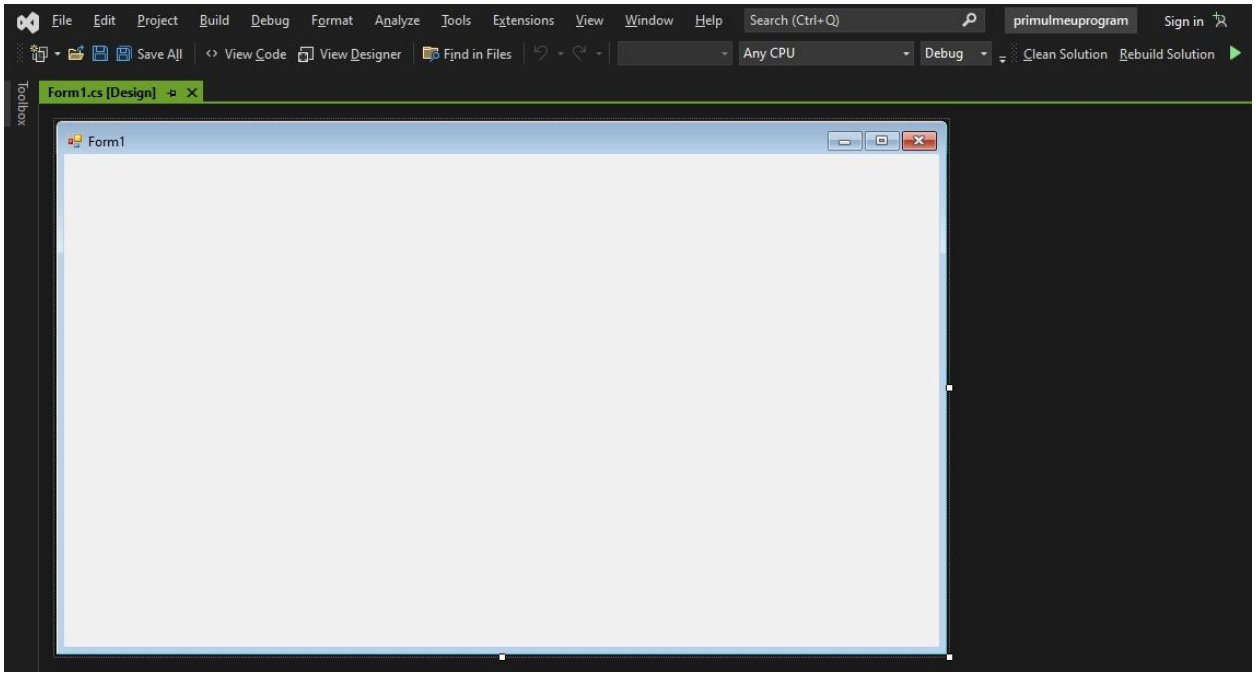


Se observă că:

- am denumit aplicaţia "primulmeuprogram"
- am specificat că fişierele sale componente vor fi plasate în directorul (folderul) "D:ţProgrameţPrograme NET 2022"
- am selectat versiunea 3.5 a platformei .NET. Aceasta înseamnă că pentru crearea aplicaţiei vom avea la dispoziţie numai Controalele care pot fi plasate pe interfaţă (pe *Dialogurile* aplicaţiei) disponibile în versiunea selectată.

În imaginea precedentă se vede caseta care poate fi bifată (este un Control de tipul *CheckBox*) pe care scrie "Place solution and project in the same directory". Aceasta se referă la aceea, că un Proiect poate fi parte a unei Soluţii alcătuite, deci, din mai multe Proiecte: un Proiect poate avea numai un Dialog, altul zece, iar alte Proiecte pot avea numai **clase**.

Apăsăm butonul "*Create*", ceea ce va rezulta în afişarea unui *Dialog* de pornire a programului nostru:



Acum vedem deci o *Form*-ă - un *Dialog* propus de mediul de programare şi pe care putem plasa Controalele care vor asigura funcţionalitatea programului.

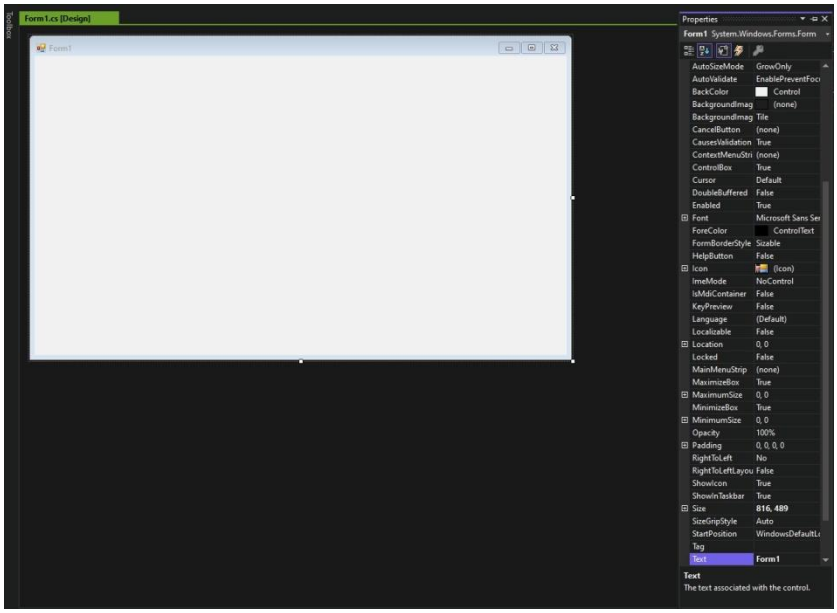
Dialogul acesta este cel care va fi afişat la pornirea programului.

Deşi acum este gol-goluţ şi programul nu face absolut nimic în afară de a afişa acest Dialog, acestuia îi pot fi stabilite unele Proprietăţi – de exemplu, titlul de pe "bara de sus" şi dimensiunea pe monitor.

Vedem cum deasupra Dialogului scrie "îDesignş" – adică, suntem în modul de Proiectare vizuală a acestui Dialog; codul propriu-zis care asigură funcţionalitatea Dialogului, este încă ascuns şi îl vom accesa mai târziu.

Click-stânga pe Dialog şi apoi pe tasta F4.

Tasta F4, odată ce un Control a fost selectat cu click-stânga (şi Dialogurile/Form-ele fiind Controale) "are focusul" (adică, l-am selectat pe un Dialog), afişează secţiunea de Proprietăţi ale acelui Control (în dreapta, în imaginea următoare):



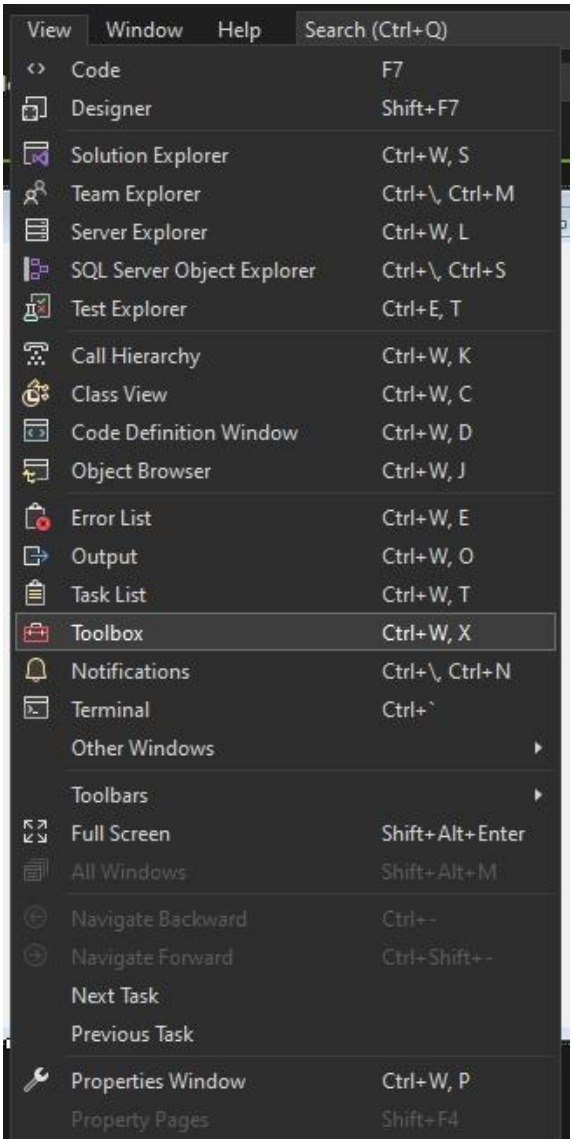
O Proprietate simplă a unui Dialog este "Text" – anume, titlul sau textul de afişat pe "bara de sus" a Dialogului. Implicit, acest Text este "Form1" – anume, dnumirea ***clasei*** ce reprezintă Dialogul.

Pentru a schimba această Proprietate la un text demn de programul nostru, click-stânga pe denumirea Proprietăţii "*Text*", sau dublu-click în dreapta, pe "Form1" şi scriem ce dorim, de exemplu "Primul meu program", apoi apăsăm tasta Enter pentru ca noul Text să fie reţinut.

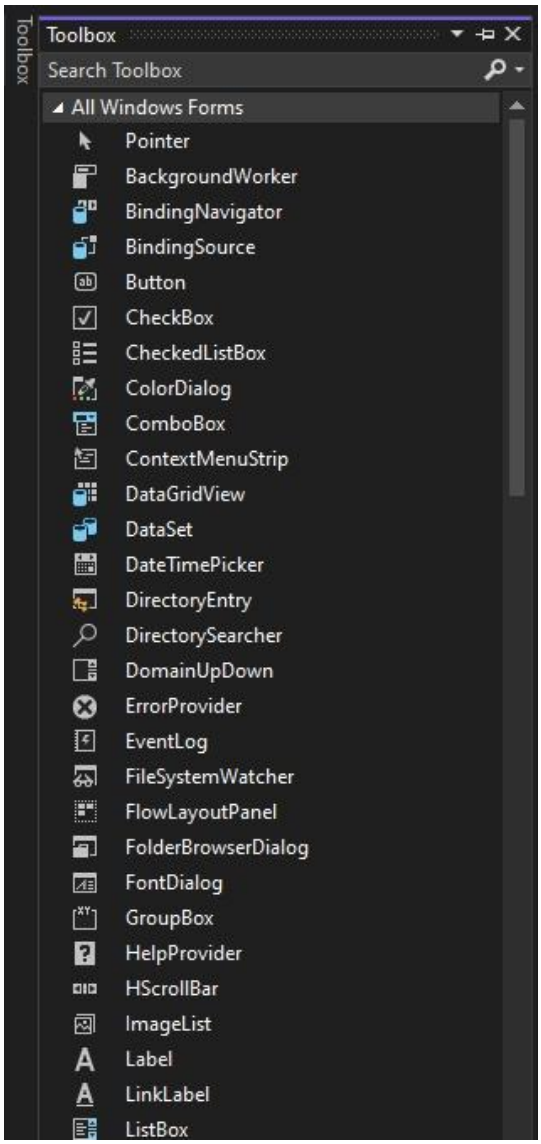
Acum e timpul ca pe Dialog să punem nişte Controale, ca totuşi programul să facă diverse chestii.

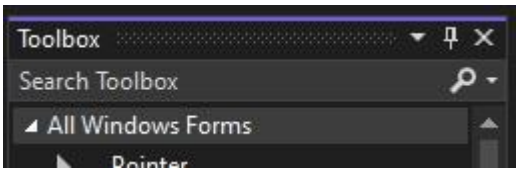
Pentru aceasta, trebuie ca întâi să aflăm ce Controale avem la dispoziţie – cele din versiunea 3.5 a platformei .NET pe care am selectat-o la început, sau din oricare versiune am selectat.

Click pe intrarea de meniu "*View*", apoi pe "*Toolbox*":



De-acum, în stânga sau altundeva pe ecran trebuie să fie afişată Cutia de Unelte din care pot fi selectate Controale de pus pe Dialog. Această secţiune (la fel ca orice altă secţiune a mediului de programare, de exemplu *Properties*) poate fi făcută să rămână afişată permanent prin click-stânga pe "acul" "*Auto-Hide*" din dreapta-sus, încât acesta să fie vertical, nu orizontal ca în prima imagine:





În această secţiune, click-stânga pe "*All Windows Forms*" – adică, vrem să alegem din toate Controalele (Forms) disponibile în .NET 3.5 pentru construirea unei aplicaţii *desktop*.

Vedem o mulţime de denumiri de Controale (derulând în jos şi în sus le aflăm pe toate), primele care atrag atenţia (apărând aranjate alfabetic, crescător) fiind "*Button*" (desigur, un buton pe care se poate face Click şi astfel executa o comandă/apela o metodă), "*CheckBox*" (o casetă pătrată care poate fi bifată) şi "*Label*" (o "etichetă" pe care este afişat un text care în timpul rulării programului nu poate fi schimbat manual de către utilizator).

Aspectul şi comportarea de bază ale mediului de programare Visual Studio

Pentru a crea cât mai uşor şi repede un program, trebuie ca navigarea/accesul printre/la secţiunile mediului de programare să fie cât mai facile.
Astfel, pot fi modificate aspectul interfeţei Visual Studio – culori, caractere, poziţia pe ecran ale unor secţiuni ca Properties (pentru Controale) ş. a.
Programatorul stabileşte Opţiuni referitoare la acestea.

Majoritatea imaginilor din continuare nu necesită explicaţii, fiindcă textele indicatoare sunt intuitive în raport cu opţiunile şi comportarea la care se referă, ale mediului de programare.

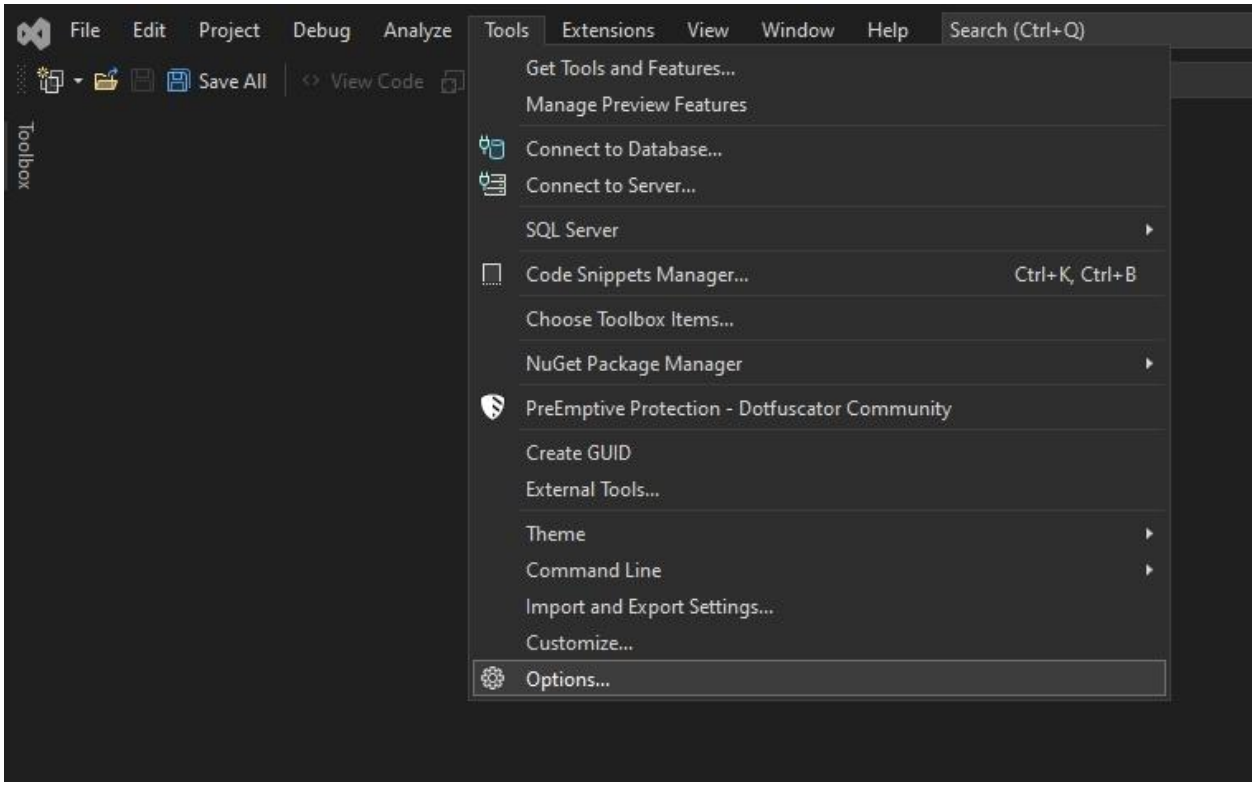
Testaţi diverse opţiuni până când aspectul mediului de scriere a codului este plăcut sau măcar suportabil

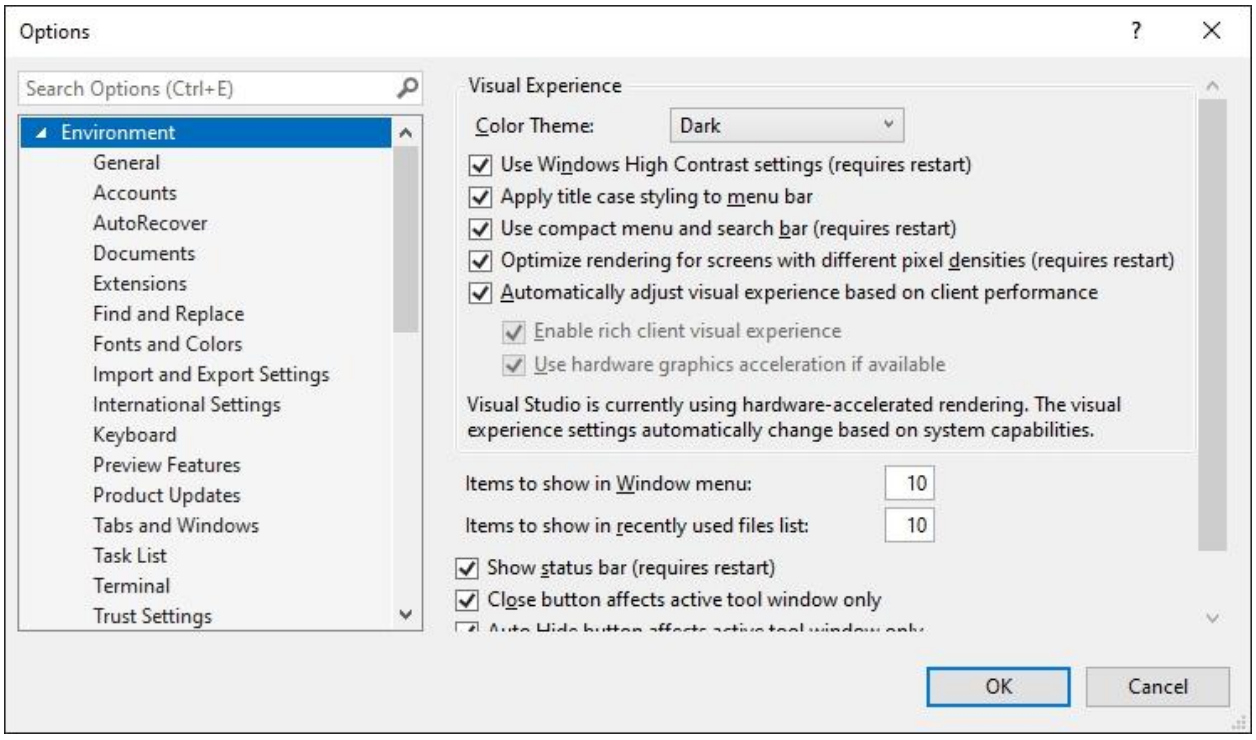


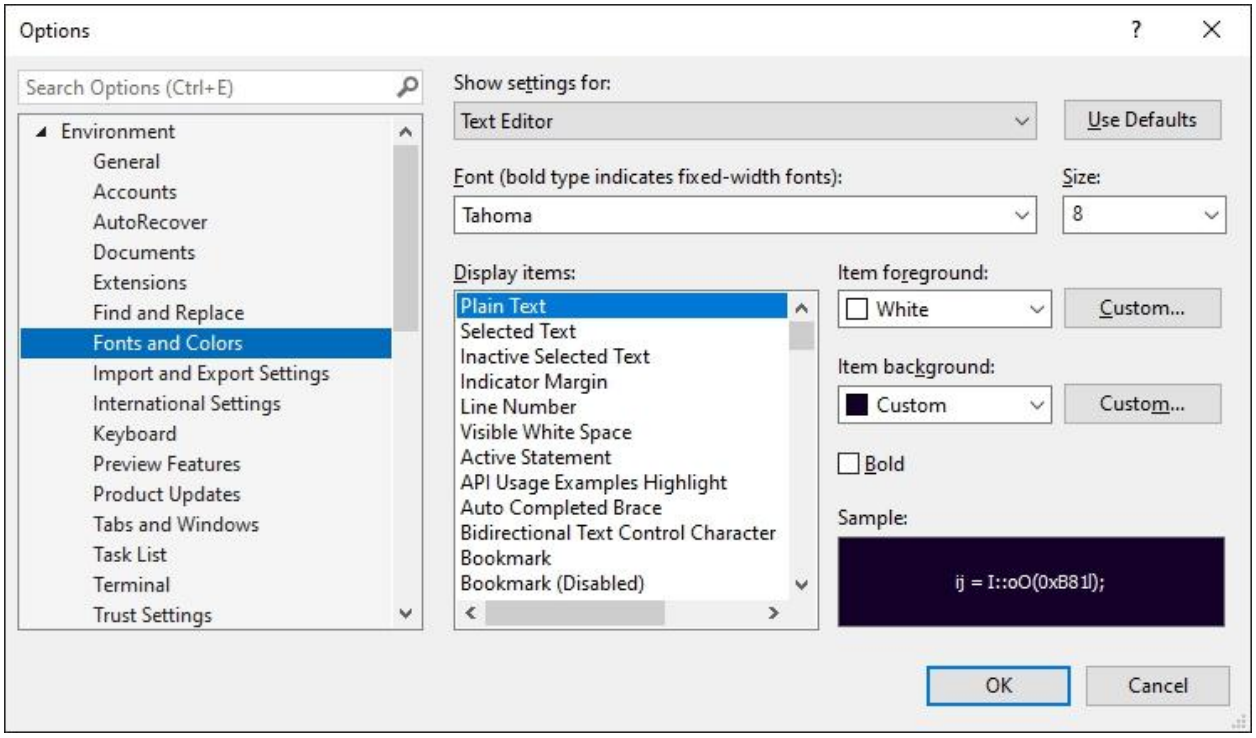
Unele noţiuni (de exemplu, versiunile programului/executabilului – "Debug" şi Release" folosite în această secţiune sunt expuse fie în curs, fie aici - în alte secţiuni; dacă lipsesc cu totul, atunci detalii pot fi găsite pe Internet.

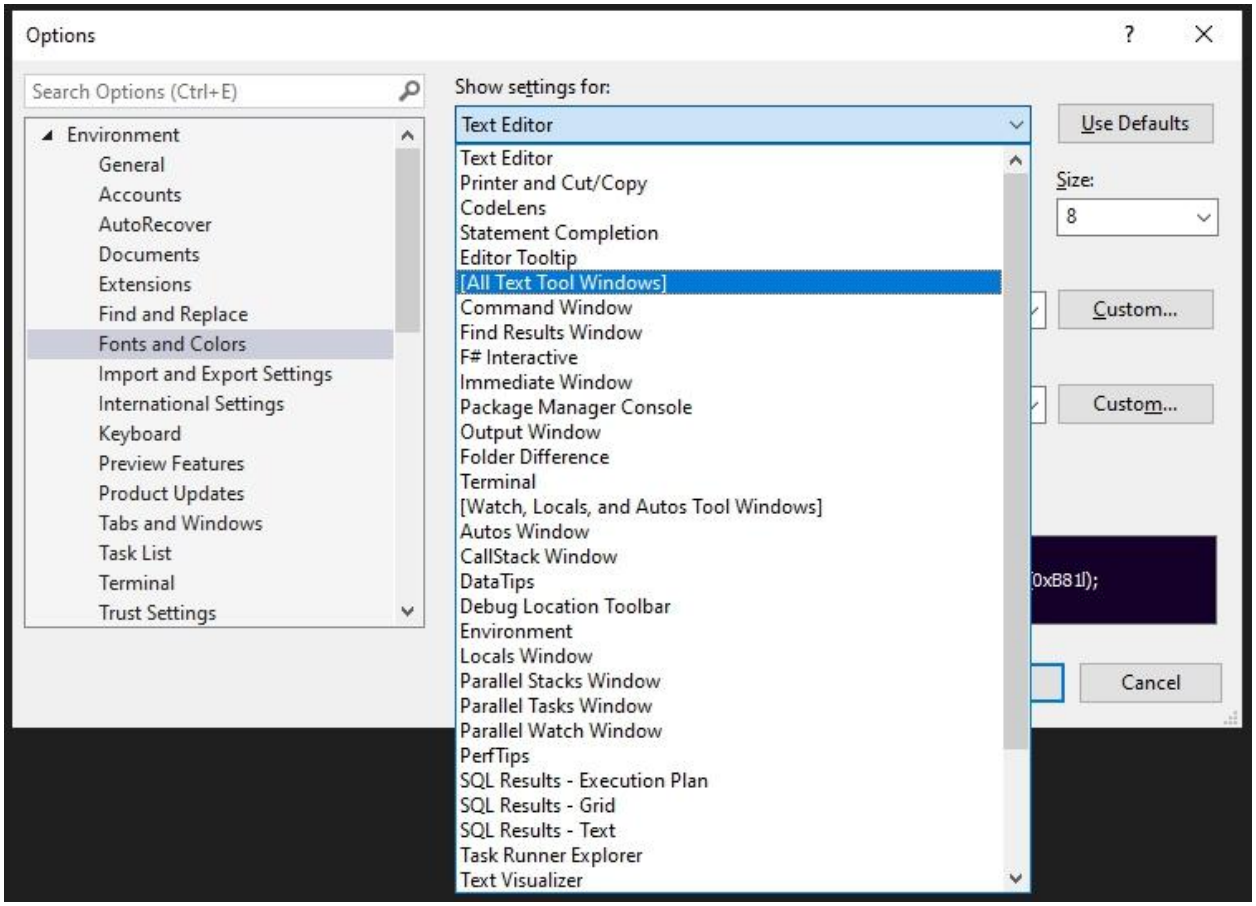
Odată pornit Visual Studio, deschidem meniul "Tools" din care selectăm "Options...".

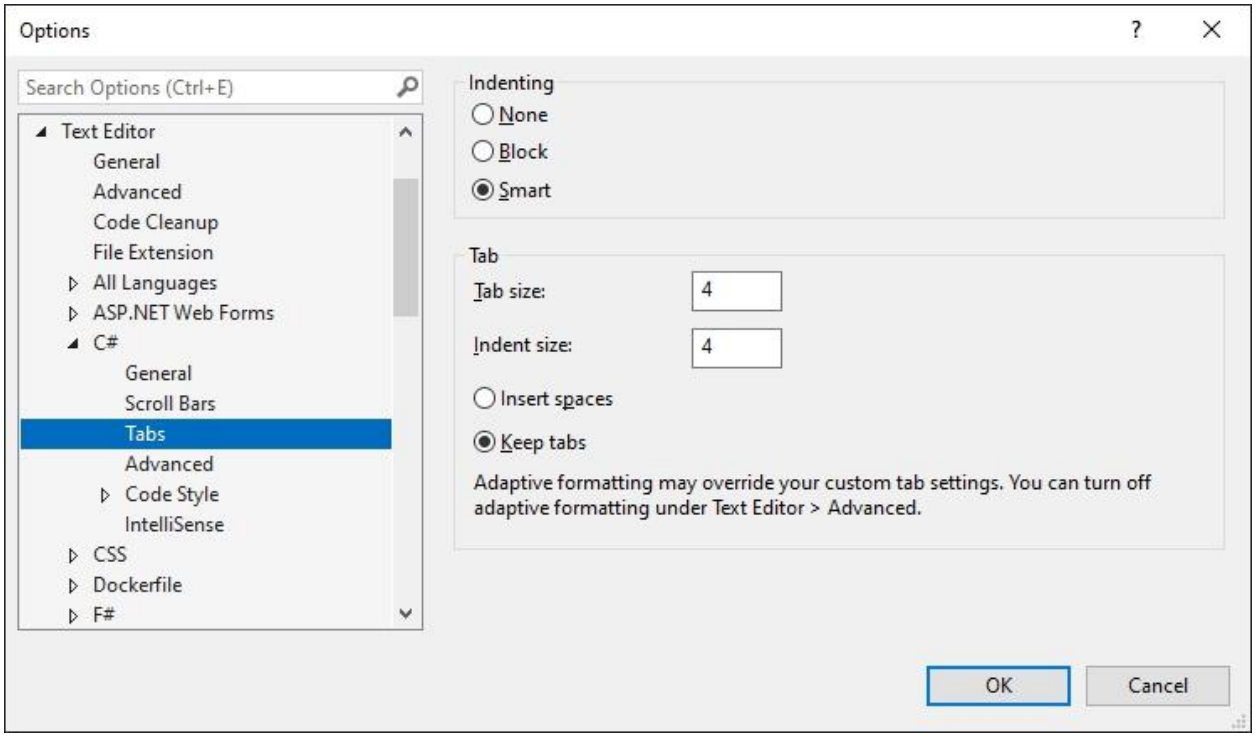
Acestea sunt opţiunile pentru Visual Studio în ansamblul său, ca întreg.





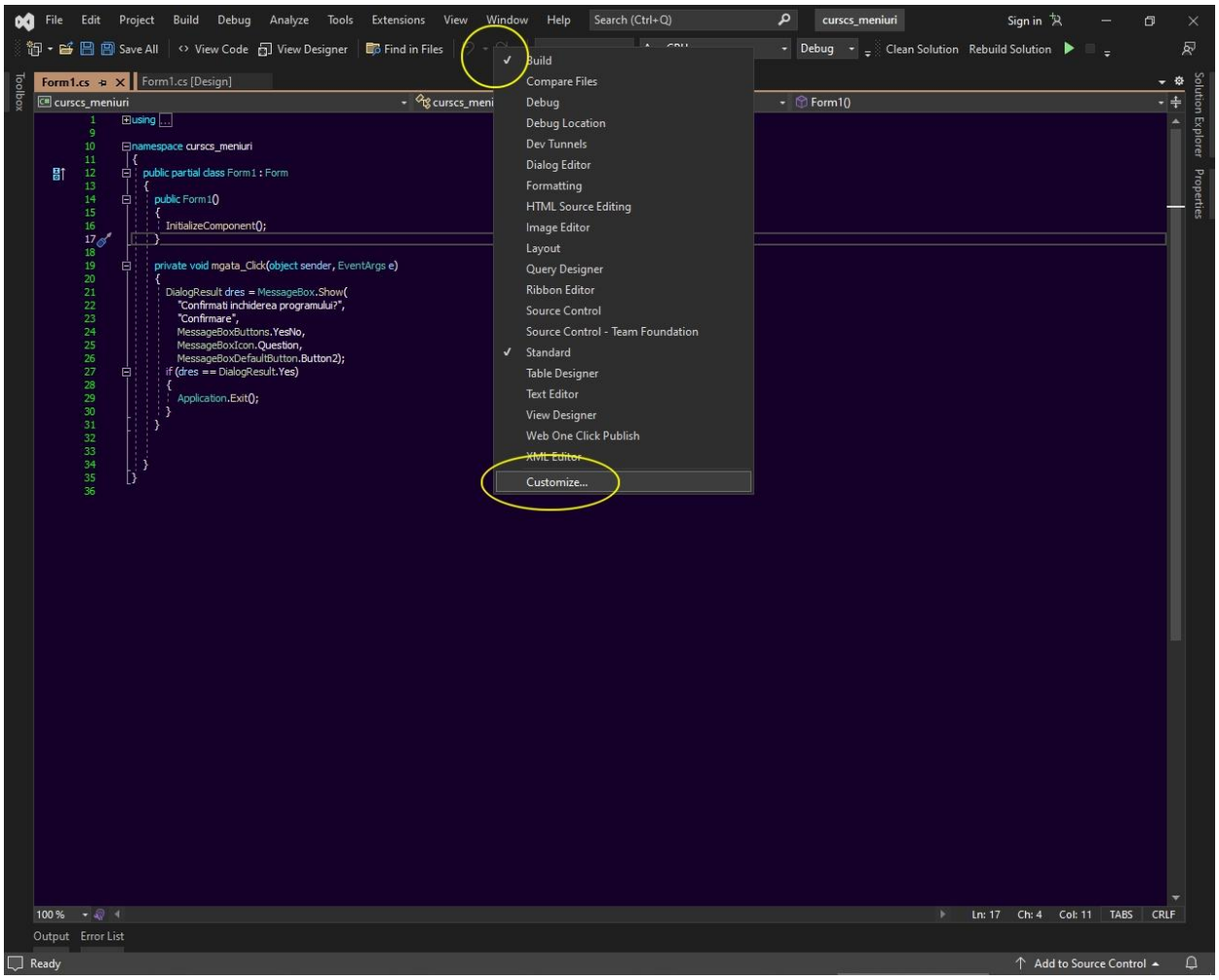

Options
Search Options (Ctrl+E)
Environment
General
Accounts
AutoRecover
Documents
Extensions
Find and Replace
Fonts and Colors
Import and Export Settings
International Settings
Keyboard
Preview Features
Product Updates
Tabs and Windows
Task List
Terminal
Trust Settings
Show settings for:
Text Editor
Use Defaults
Font (bold type indicates fixed-width fonts):
Tahoma
Size:
8
Display items:
Plain Text
Selected Text
Inactive Selected Text
Indicator Margin
Line Number
Visible White Space
Active Statement
API Usage Examples Highlight
Auto Completed Brace
Bidirectional Text Control Character
Bookmark
Bookmark (Disabled)
Item foreground:
White
Custom...
Item background:
Custom
Custom...
Bold
Sample:
ij = I::oO(0xB81l);
OK
Cancel


Options
Search Options (Ctrl+E)
Environment
General
Accounts
AutoRecover
Documents
Extensions
Find and Replace
Fonts and Colors
Import and Export Settings
International Settings
Keyboard
Preview Features
Product Updates
Tabs and Windows
Task List
Terminal
Trust Settings
Show settings for:
Text Editor
Use Defaults
Text Editor
Printer and Cut/Copy
CodeLens
Statement Completion
Editor Tooltip
[All Text Tool Windows]
Command Window
Find Results Window
F# Interactive
Immediate Window
Package Manager Console
Output Window
Folder Difference
Terminal
[Watch, Locals, and Autos Tool Windows]
Autos Window
CallStack Window
DataTips
Debug Location Toolbar
Environment
Locals Window
Parallel Stacks Window
Parallel Tasks Window
Parallel Watch Window
PerfTips
SQL Results - Execution Plan
SQL Results - Grid
SQL Results - Text
Task Runner Explorer
Text Visualizer
Size:
8
Custom...
Custom...
0xB81l);
Cancel

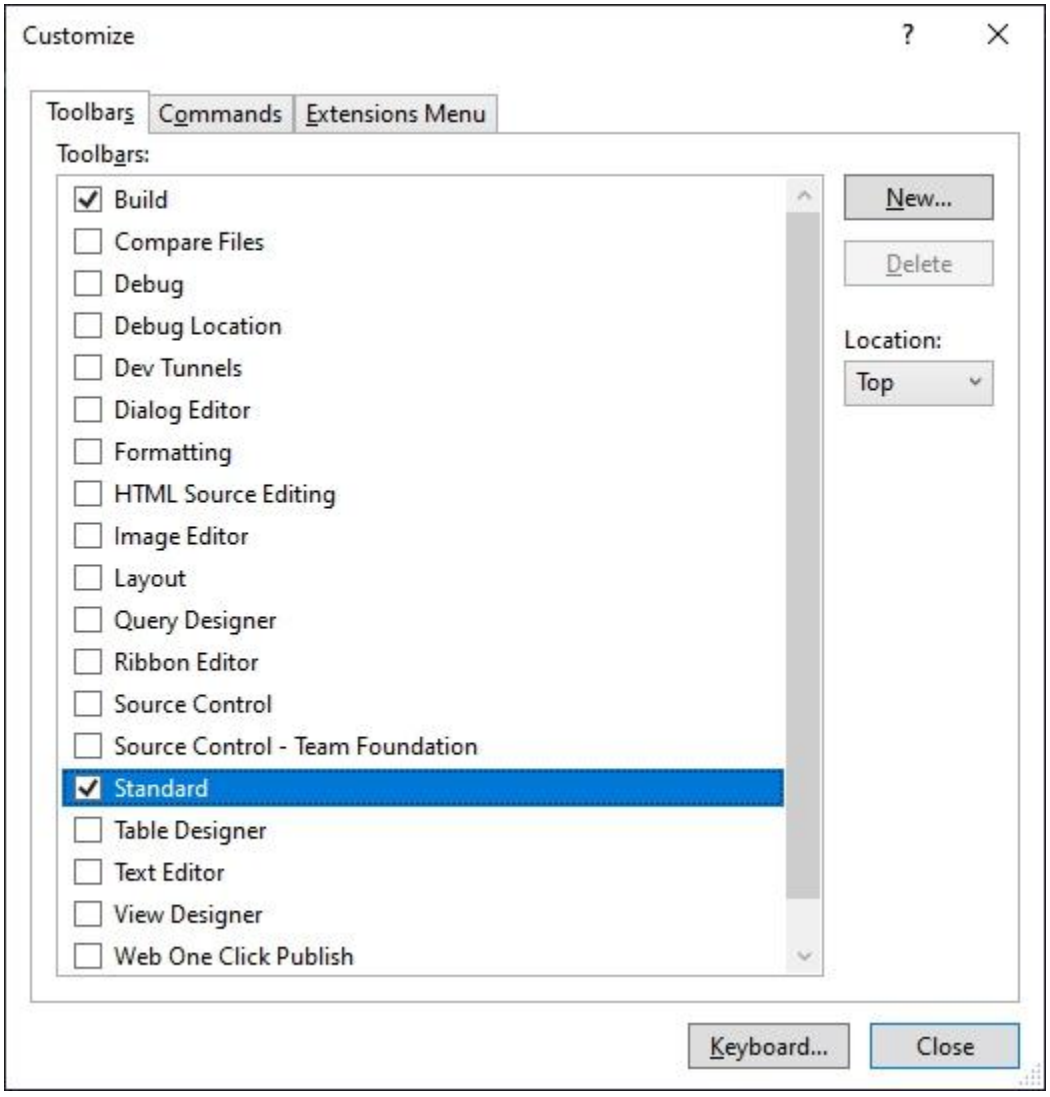

Options
Search Options (Ctrl+E)
Text Editor
General
Advanced
Code Cleanup
File Extension
All Languages
ASP.NET Web Forms
C#
General
Scroll Bars
Tabs
Advanced
Code Style
IntelliSense
CSS
Dockerfile
F#
Indenting
None
Block
Smart
Tab
Tab size:
4
Indent size:
4
Insert spaces
Keep tabs
Adaptive formatting may override your custom tab settings. You can turn off adaptive formatting under Text Editor > Advanced.
OK
Cancel

Pot fi modificate meniurile şi comenzile de sub meniul principal Visual Studio:
Click-dreapta pe "bara" de sub meniul principal:
Va fi afişat un *meniu de context*, sau *contextual* – asociat Controlului tocmai acţionat prin click-dreapta:

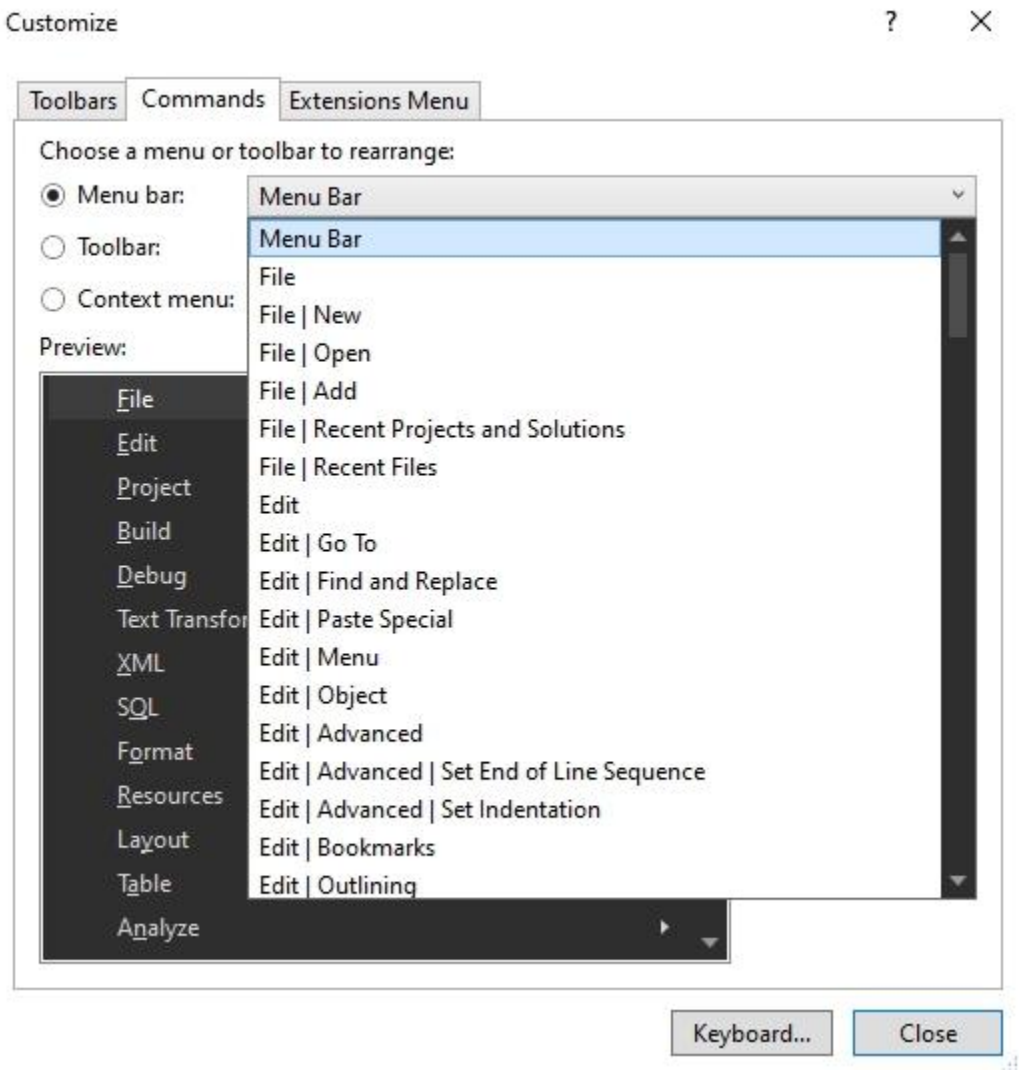
Se observă cum sunt bifate "Build" şi "Standard".
Acestea sunt "barele de unelte" (Toolbars) vizibile acum: de exemplu, în cea "Standard" se află comenzile "Save All", "View Code", View Designer" – care apar sub meniul principal, be această bară de unelte:



Am apăsat pe "Customize":
Aici putem bifa barele de unelte pe care le vrem a fi vizibile:
Observăm că este selectată pagina "Tollbars":

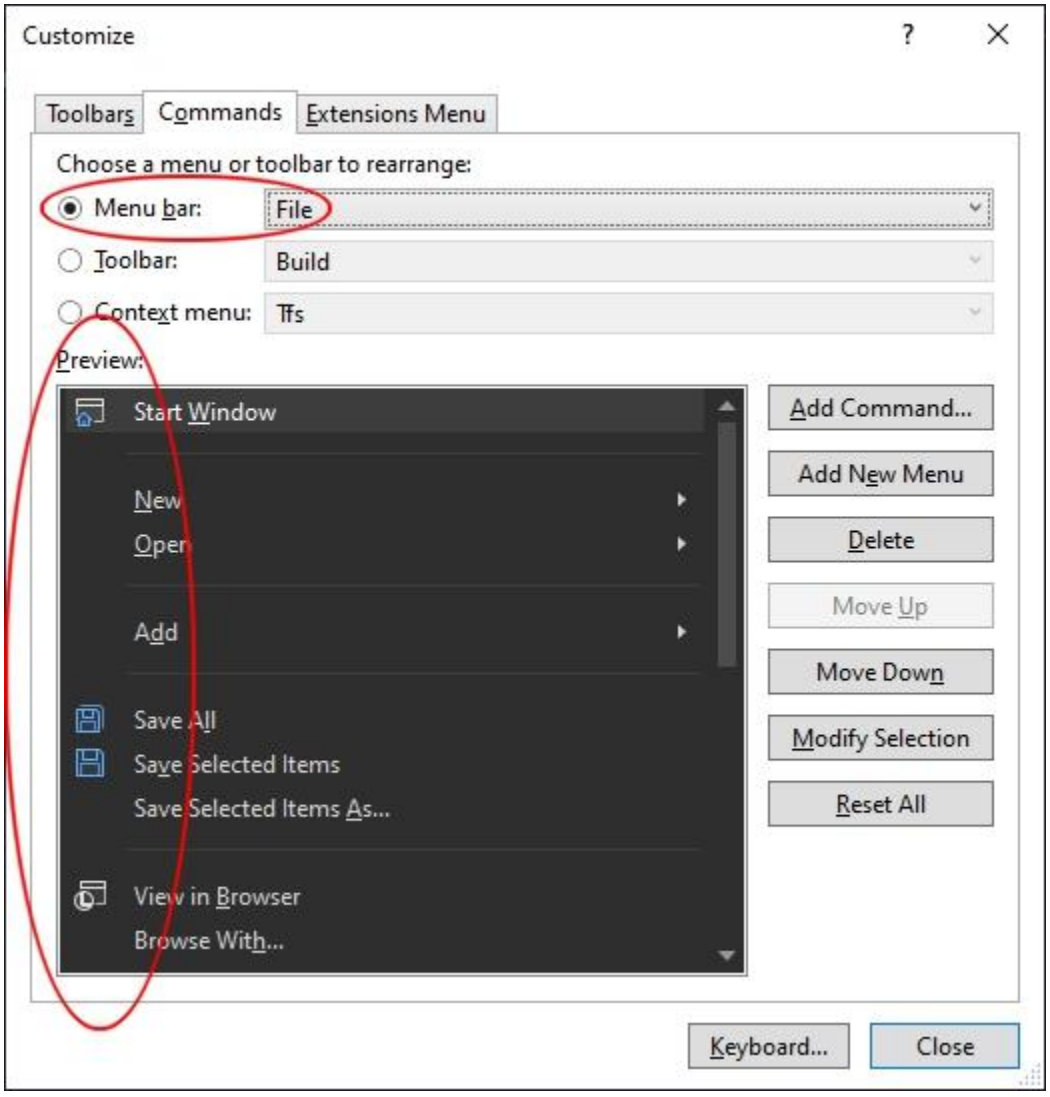


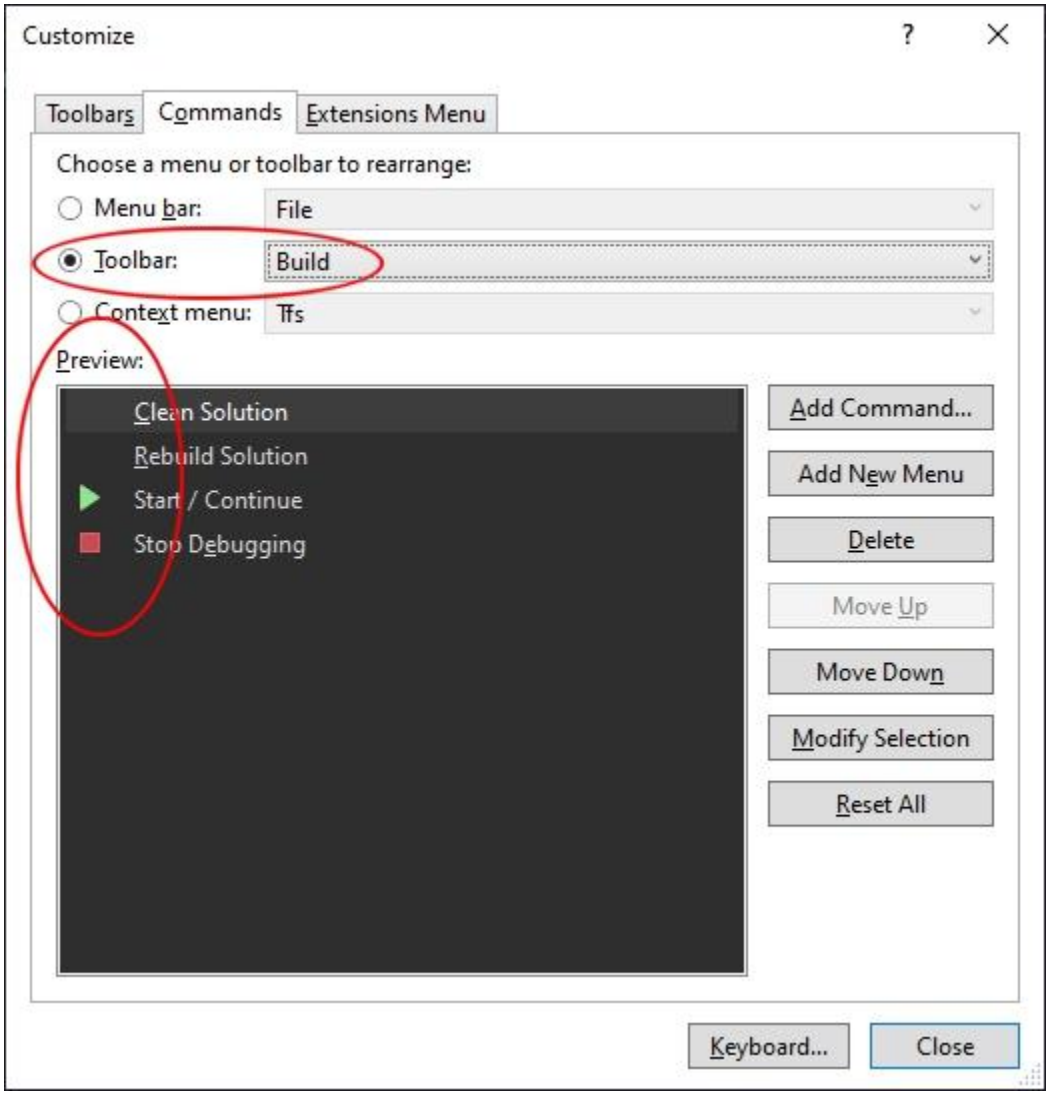
Selectăm pagina "Commands":
Aici putem modifica lista comenzilor disponibile în fiecare meniu şi bară de unelte a Visual Studio:

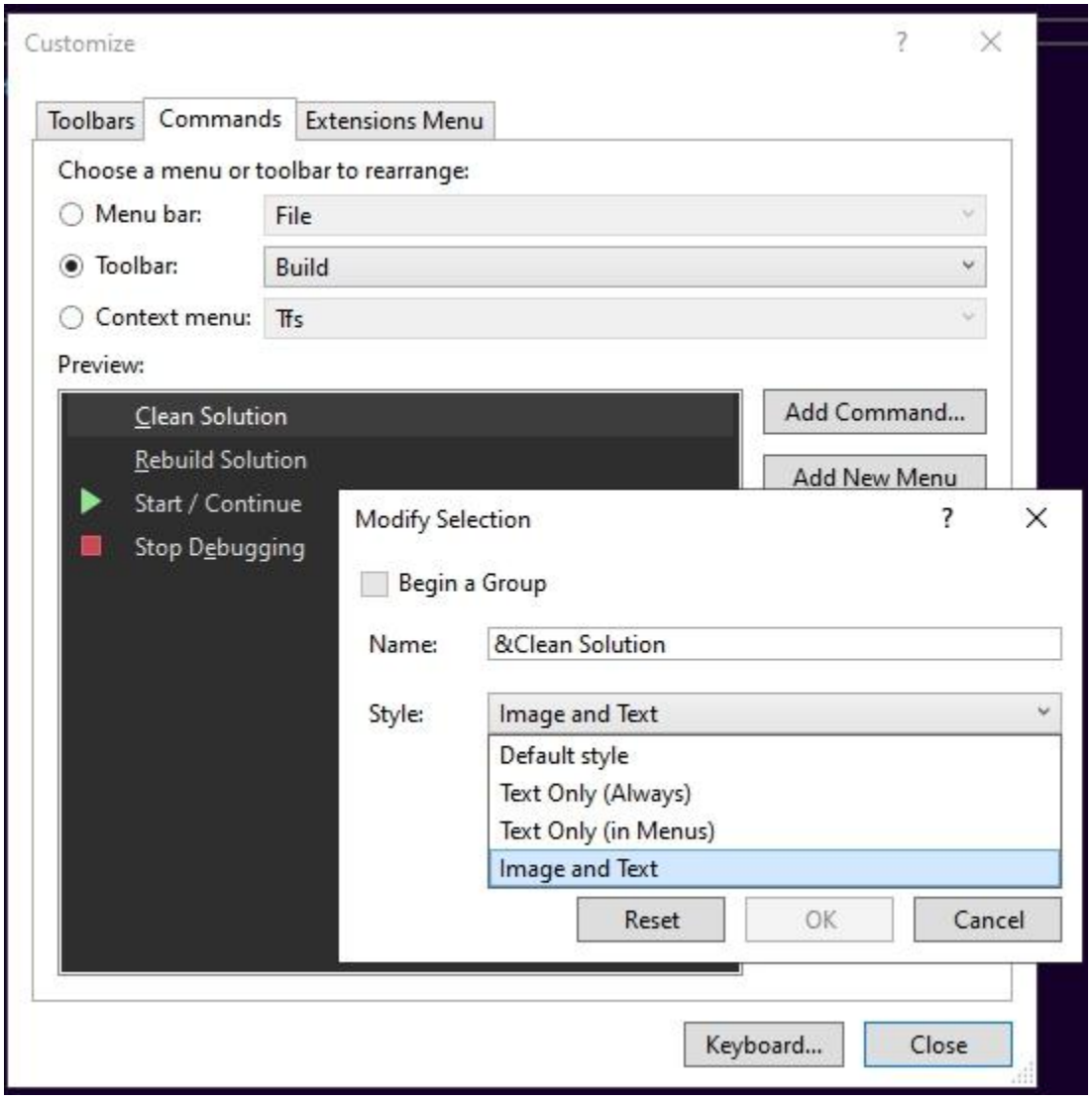


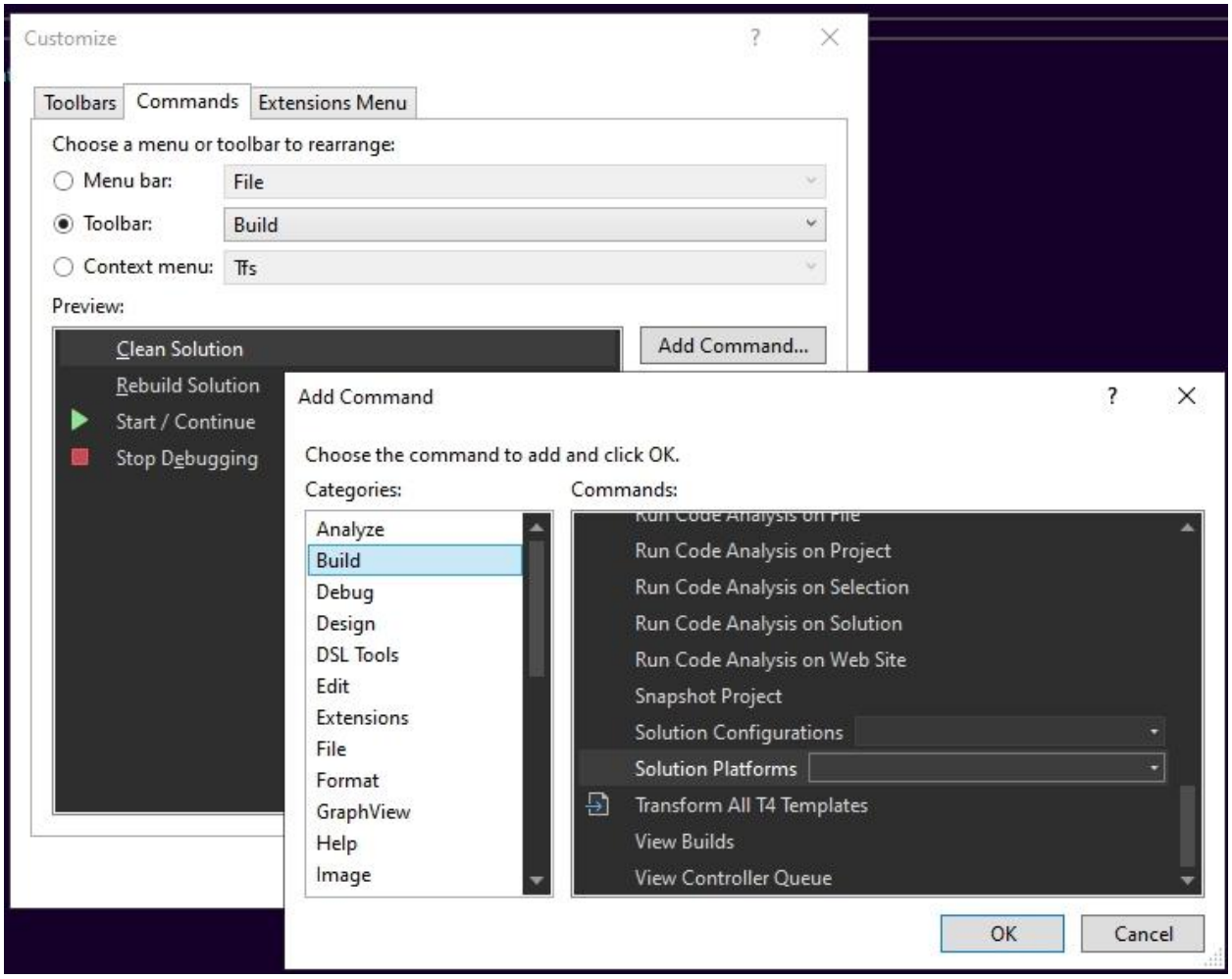
O comandă poate fi modificată - "Modifiy Selection", ştearsă - "Delete", mutată - "Move Up/Down":

Tot aici, comenzi pot fi adăugate – "Add Command" – la meniuri şi bare de unelte:

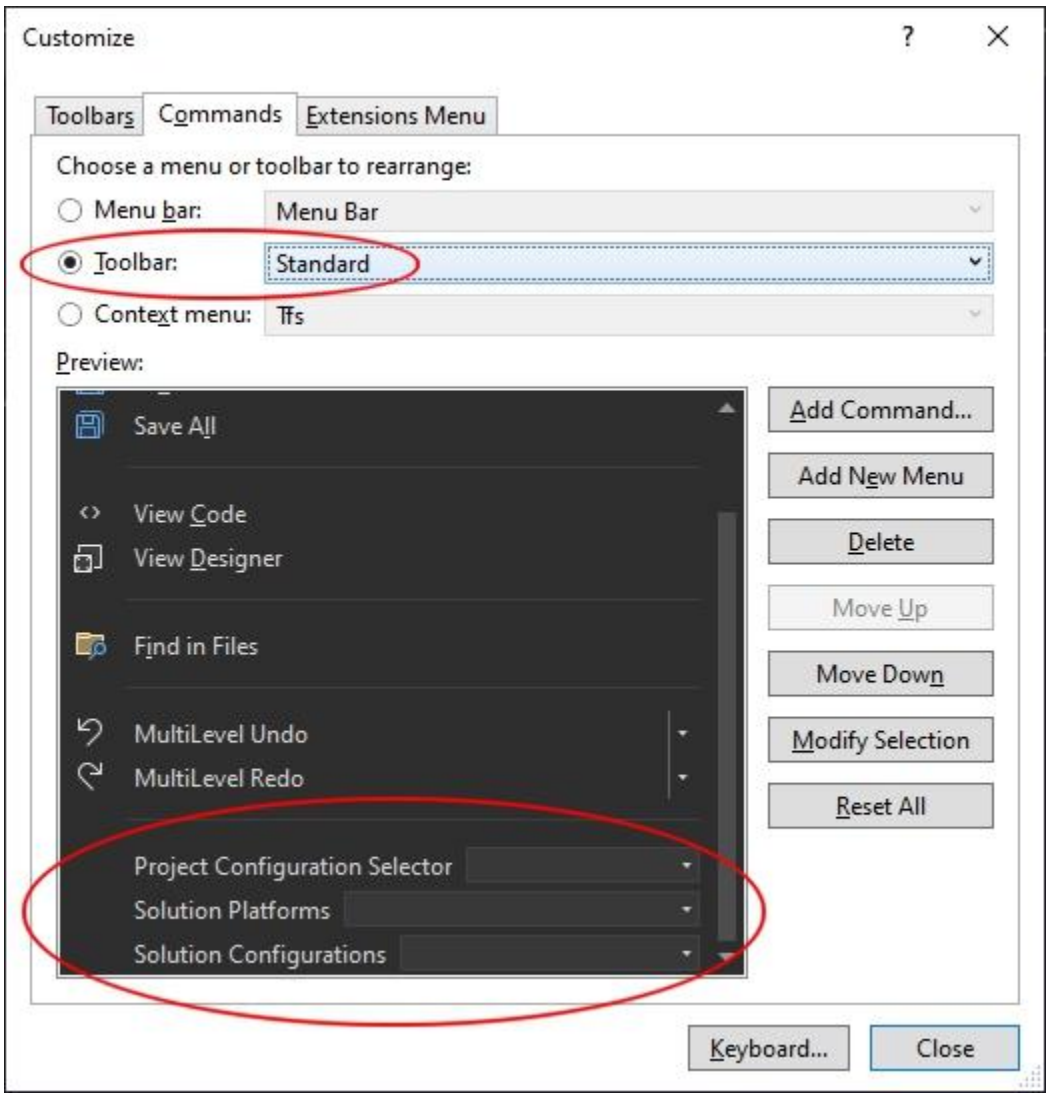


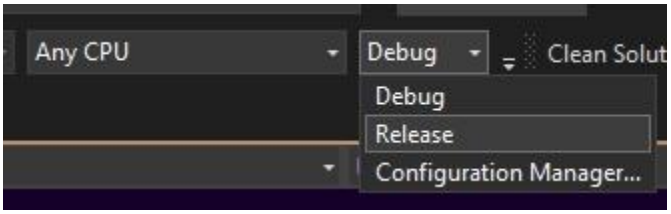

Customize
Toolbars
Commands
Extensions Menu
Choose a menu or toolbar to rearrange:
Menu bar:
File
Toolbar:
Build
Context menu:
Tfs
Preview:
Clean Solution
Rebuild Solution
Start / Continue
Stop Debugging
Add Command...
Add New Menu
Delete
Move Up
Move Down
Modify Selection
Reset All
Keyboard...
Close


Customize
Toolbars
Commands
Extensions Menu
Choose a menu or toolbar to rearrange:
Menu bar:
File
Toolbar:
Build
Context menu:
Tfs
Preview:
Clean Solution
Rebuild Solution
Start / Continue
Stop Debugging
Add Command...
Add New Menu
Modify Selection
Begin a Group
Name:
&Clean Solution
Style:
Image and Text
Default style
Text Only (Always)
Text Only (in Menus)
Image and Text
Reset
OK
Cancel
Keyboard...
Close

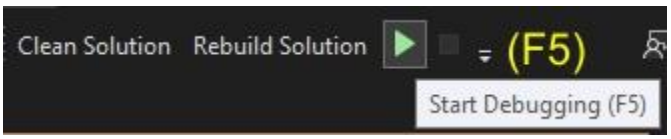

Customize
Toolbars
Commands
Extensions Menu
Choose a menu or toolbar to rearrange:
Menu bar:
File
Toolbar:
Build
Context menu:
Tfs
Preview:
Clean Solution
Rebuild Solution
Start / Continue
Stop Debugging
Add Command...
Add Command
Choose the command to add and click OK.
Categories:
Commands:
Analyze
Build
Debug
Design
DSL Tools
Edit
Extensions
File
Format
GraphView
Help
Image
Run Code Analysis on Project
Run Code Analysis on Selection
Run Code Analysis on Solution
Run Code Analysis on Web Site
Snapshot Project
Solution Configurations
Solution Platforms
Transform All T4 Templates
View Builds
View Controller Queue
OK
Cancel

Versiunile "Debug" şi "Release" ale programului/executabilului sunt în bara de unelte "Standard", cum se vede în imagine:





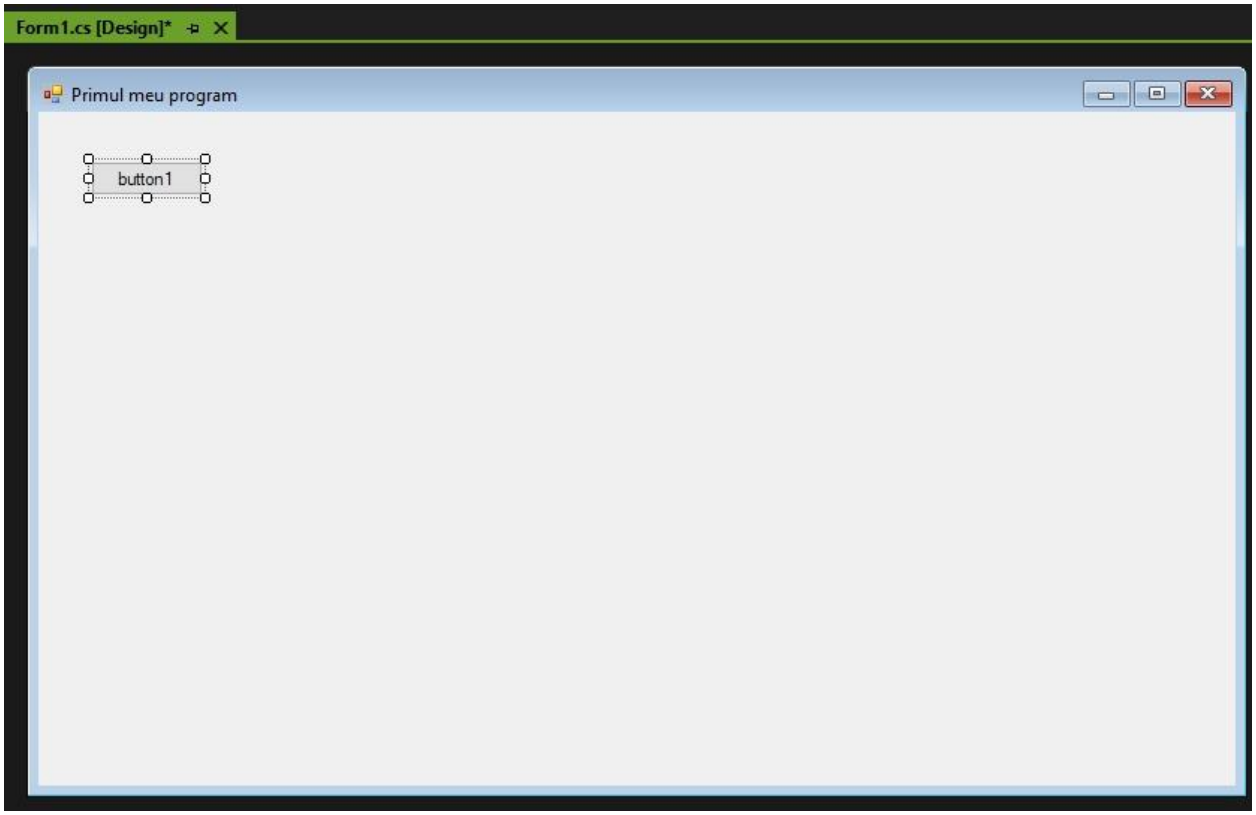
Pictograma care odată apăsată lansează programul în execuţie este în bara de unelte "Build" şi acţionarea sa este echivalentă apăsării tastei F5:



Editor de text,
asemănător cu Notepad

Un Control poate fi plasat pe un Dialog prin click-stânga pe acel Control şi trăgându-l pe Dialog; de asemenea, prin dublu-click-stânga pe acel Control.

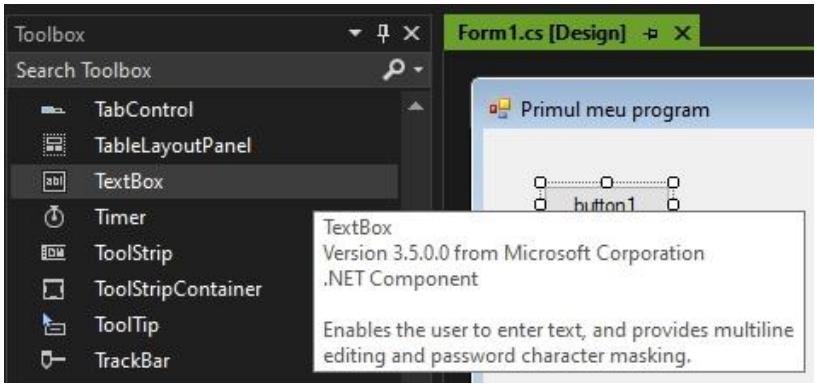
Să plasăm un *Button* pe Dialog:



Text-ul Buttonului este, implicit, "button1". Pentru a-l schimba, selectăm Buttonul, apoi tasta F4 (Proprietăţile Buttonului) unde schimbăm proprietatea *Text* a Buttonului la ce text anume dorim, la fel cum am schimbat-o pentru Dialog.

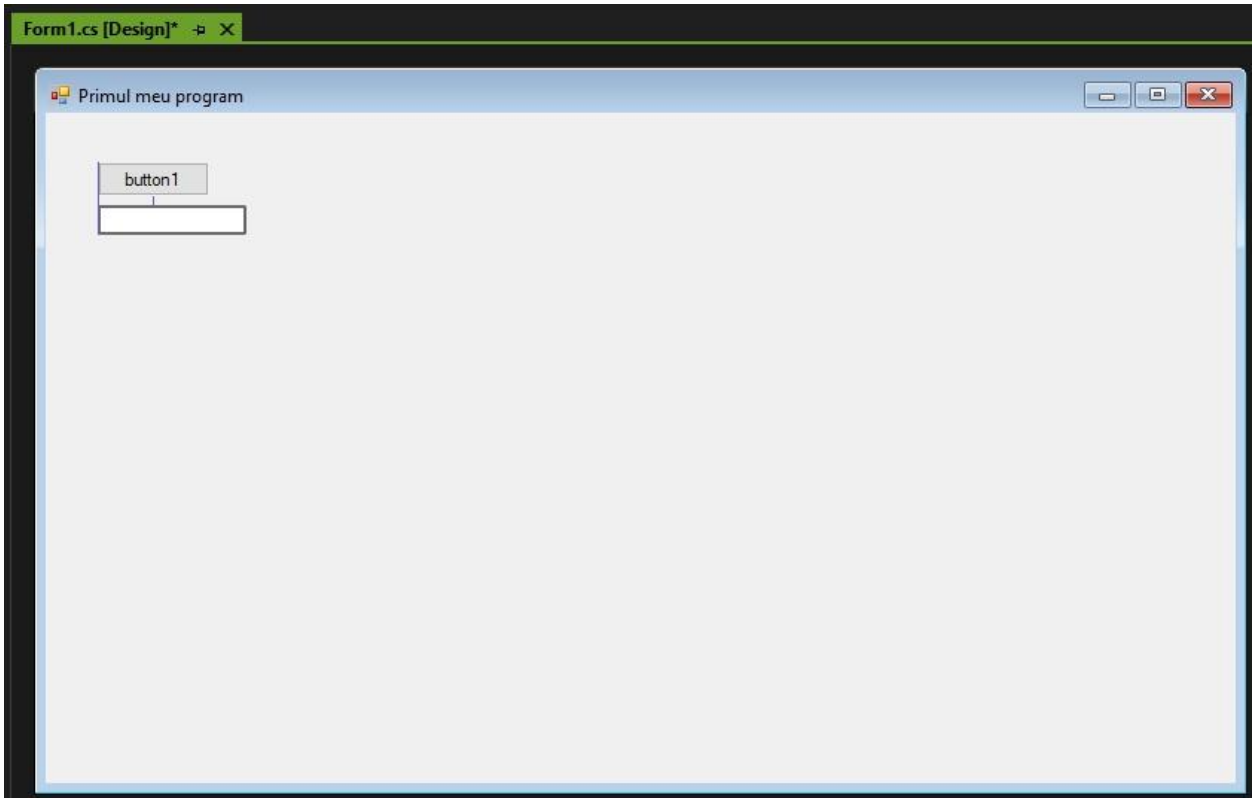
Acum, din Toolbox alegem şi plasăm pe Dialog o casetă de/pentru text, în care utilizatorul va putea scrie ceva.

Acest Control are denumirea "*TextBox*". Derulăm Toolbox-ul până *TextBox* apare în listă şi acum putem trage pe Dialog un astfel de Control:



Observăm cum, dacă ţinem un timp pointerul deasupra lui *TextBox* apare o scurtă descriere a acestui Control ("permite utilizatorului să scrie un text şi oferă posibilitatea de editare pe mai multe randuri/linii şi mascarea ca pentru o parolă a literelor/caracterelor").

Această funcţionalitate a mediului de porgramare este valabilă pentru oricare Control listat în Toolbox, deşi descrierea sumară a unui Control poate lipsi.



Se vede cum am plasat un *TextBox* chiar sub *Button*. Liniile subţiri albastre apar când un Control aflat pe Dialog şi tras cu mouse-ul se află în apropierea altor Controale şi permit programatorului să alinieze/aranjeze noul Control (în raport) cu altele.

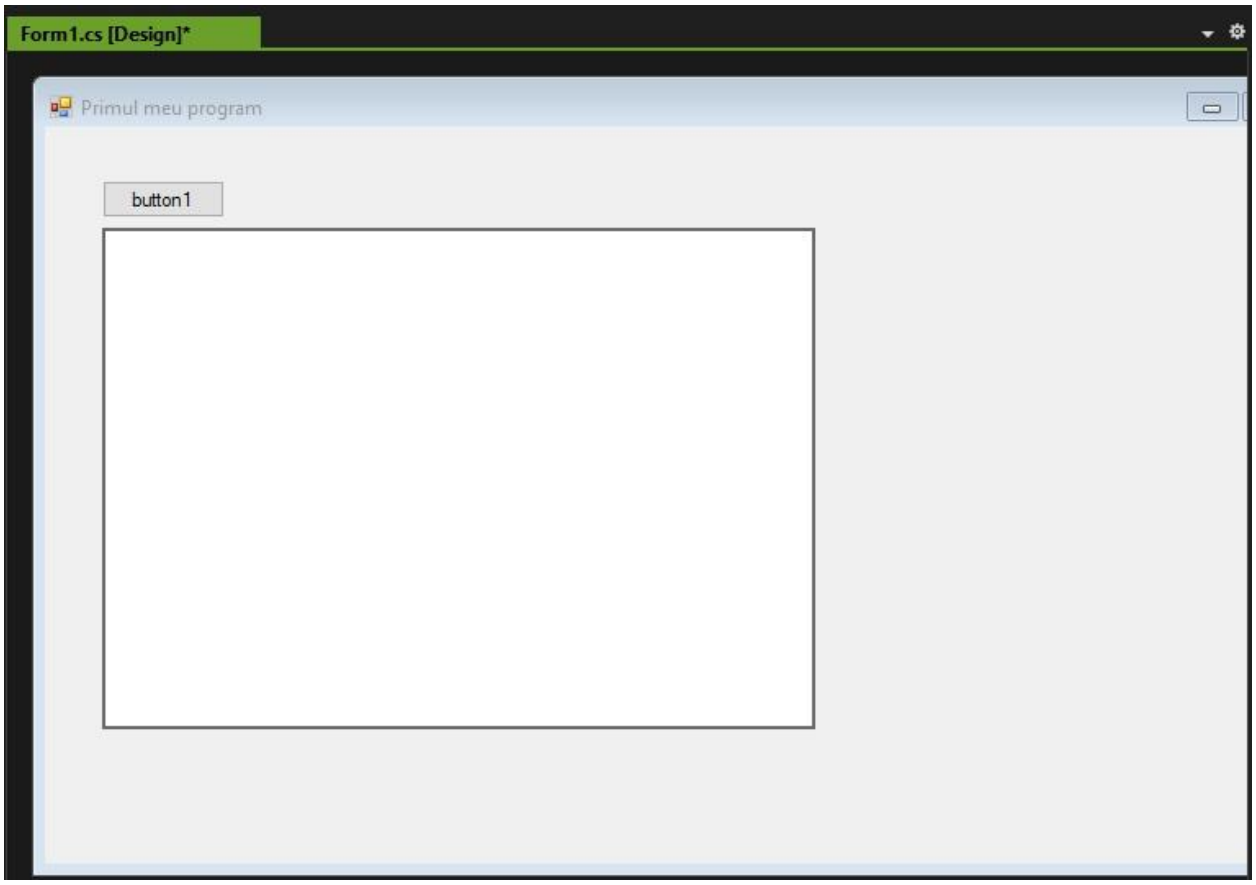
Alinierea Controalelor pe Dialog se face pentru a proiecta o interfaţă cu aspect decent, nu dezordonat – cu Controale plasate de-a valma, fără logică evidentă şi fără a sugera fluxul de urmat de către utilizator.

TextBox-ul tocmai plasat pare a nu fi capabil să conţină vreun text cât de cât lung, scris de către utilizator; aşa că îi modificăm unele Proprietăţi pentru a face din el o casetă de text adecvată scopului programului nostru. Deci, click-stânga pe *TextBox* şi apoi apăsăm tasta F4.

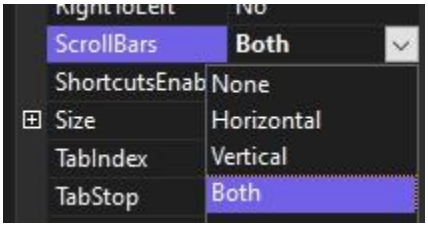
Întâi, stabilim ca *TextBox*-ul să poată conţine text pe mai multe rânduri/linii. Aceasta este Proprietatea "*MultiLine*" pe care, bineînţeles, o schimbăm din *False* în *True* (este o variabilă de tip Boolean, adevărat sau fals, 1 sau 0):



Acum, pentru a putea vedea totuşi ce anume se scrie în acest *TextBox* – casetă care, deşi este capabilă de a afişa text pe mai multe rânduri, acum este minusculă – agăţăm cu poinerul mouse-ului un colţ al acesteia şi tragem de el, încât să o redimensionăm; în imaginea următoare se vede cum am mărit caseta de text trăgând spre dreapta-jos de colţul său din dreapta-jos:



Proprietatea *MultiLine* = *True* pare a nu fi suficientă, întru orientarea utilizatorului, pentru o casetă de text pe mai multe rânduri – aşa că schimbăm şi Proprietatea *ScrollBars* care determină afişarea sau nu a barelor de derulare orizontală şi verticală a textului din casetă; vom stabili această Proprietate la "*Both*" – adică, la nevoie să fie afişate ambele bare de derulare:



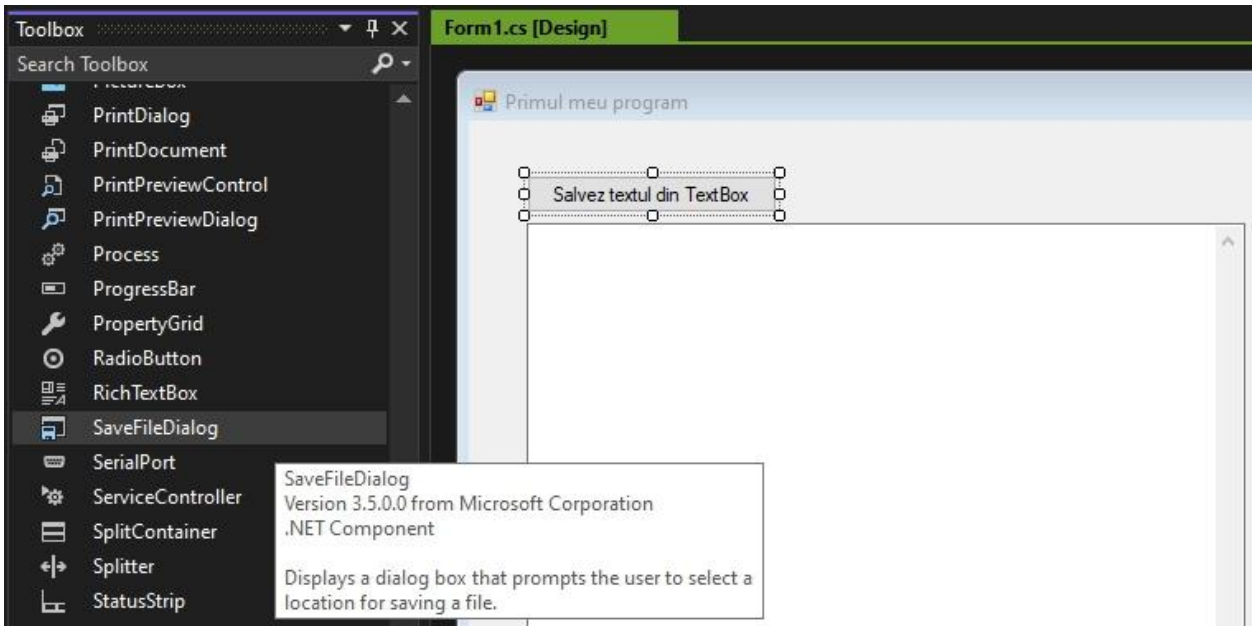
Şi acum, odată ce am pregătit chiar şi cu numai două Controale interfaţa cu utilizatorul, să trecem la ceea ce vrem să facă programul, anume:

- să scriem un text pe care să-l salvăm
- să deschidem un fişier cu text pe care să-l afişăm în *TextBox*

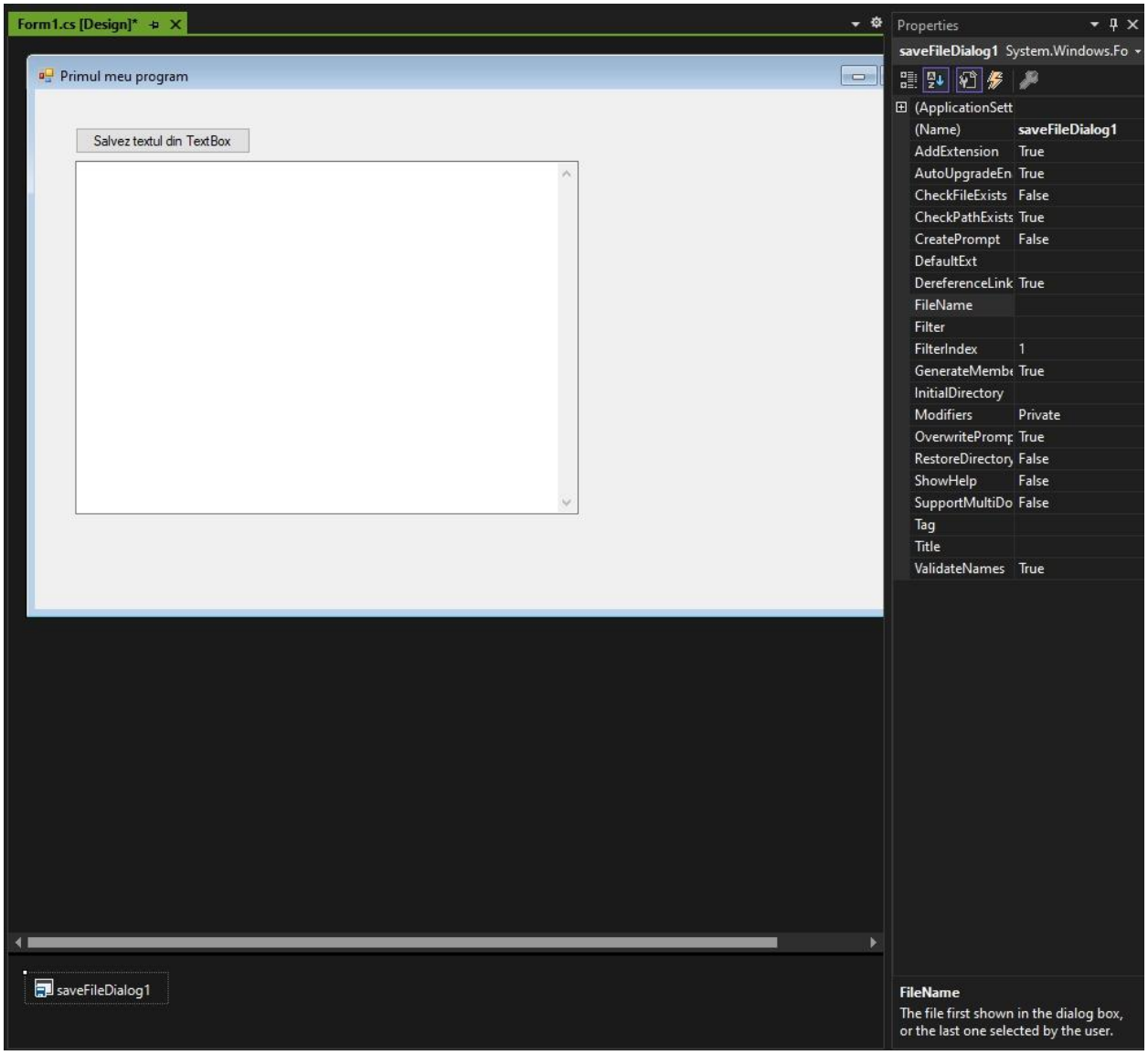
Pentru aceasta nu avem încotro decât să **începem a scrie cod** – codul asociat Controalelor (inclusiv Dialogului *Form1*), codul care înseamnă f**uncţionalitatea** programului, ceea ce face ca programul să aibă un rost.

Între timp am schimbat Text-ul Buttonului la "Salvez textul din TextBox" şi pentru ca acest text să fie vizibil integral pe *Button*, am redimensionat Buttonul la fel cum am făcut cu TextBoxul.

Salvarea a orice (text, imagine ş.a.) într-un fişier se face folosind Controlul *SaveFileDialog* pe care de asemenea îl putem trage din Toolbox pe Dialog:



Acest Control şi de altfel multe altele utile unui program chiar modest (ca *OpenFileDialog*, *Timer*, *ImageList*) nu apare ca un *Button*, de exemplu; nu are un contur anume, ci în mediul de programare apare sub Dialog, astfel:



Ca toate Controalele, însă, *SaveFileDialog* are Proprietăţi care la fel pot fi stabilite selectându-l (click-stânga pe el) şi apoi apăsând tasta F4.

În secţiunea de Proprietăţi ale sale se evidenţiază:

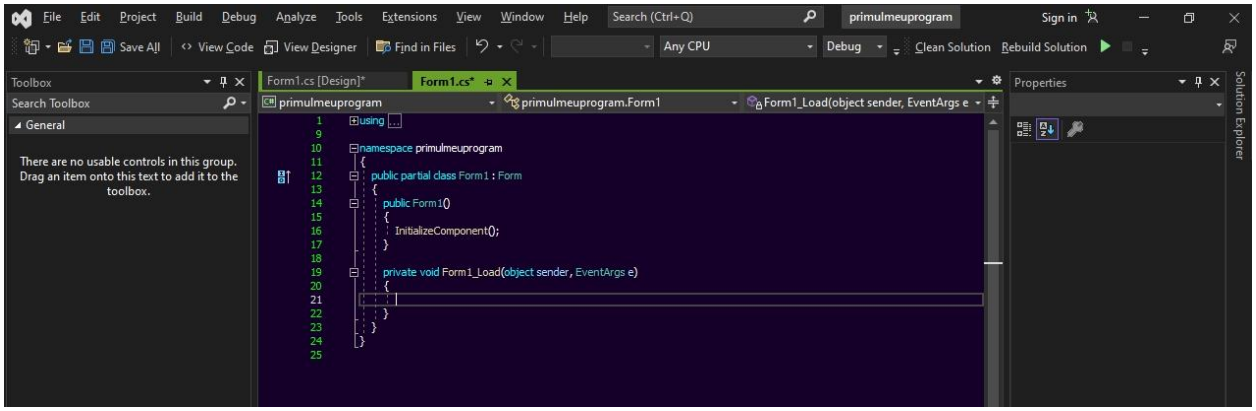
- "*FileName*": acesta va fi denumirea (ori, numele) fişierului de salvat
- "*AddExtension*": utilizatorul nu va fi nevoit să scrie extensia fişierului de salvat, adică în acest caz".txt", ci extensia va fi adăugată automat la numele fişierului
- "*Title*": acesta este textul de pe "bara de sus" a dialogului în care utilizatorului i se va solicita numele fişierului de salvat; poate fi stabilită la, de exemplu, "Selectati locul si numele fisierului pe care vreti sa-l salvati".

- "*Filter*": o Proprietate esenţială a acestui Control. Stabileşte ce anume tip (ca extensie) de fişiere vor putea fi salvate. Este o variabilă de tip *string* şi o vom stabili în cod

Deci începem să **scriem** programul.

Pentru a vedea codul asociat Dialogului pe care tocmai am plasat un *Button*, un *TextBox* şi un *SaveFileDialog*:
- dublu-click pe Dialog, ***sau***
- click pe Dialog şi se apasă tasta F7

rezultatul fiind acelaşi, anume afişarea codului asociat Dialogului:



Dacă ne speriem, ne şi calmăm fiindcă nu e nimic de speriat şi începem a inspecta întru înţelegere ceea ce tocmai a afişat mediul de porgramare.

În stânga se vede binecunoscutul deja Toolbox. Decât că, nemaiaflându-ne în modul Design al Dialogului, ci în modul de scriere a codului, Controalele din Toolbox sunt invizibile: nu avem ce Controale să adăugăm la/să tragem peste un text (cod).

În dreapta se vede binecunoscuta deja secţiune a Proprietăţilor, cu aceeaşi menţiune. Nu avem ce Proprietăţi să stabilim codului, care până una-alta este o litanie de text care va fi interpretat de compilatorul mediului de programare şi transformat într-un program executabil, aşa cum de altfel vrem.

Între cele două secţiuni se vede o porţiune unde putem scrie **codul propriu-zis** al programului, ori unde **scriem programul**.

Pe primul rând scrie "**using**...", cuvânt precedat de semnul "+". Acest semn denotă aici "un fel de" *Regiune de cod* - ne amintim din curs că secţiuni de cod pot fi grupate între directivele #region şi #endregion; apăsând pe semnul "+" se poate vedea secţiunea de cod ascunsă – derulate Referinţele la porţiuni din platforma .NET care, implicit, sunt inserate de către mediul de programare într-o aplicaţie de tip Desktop aşa cum am ales la început (Referinţe sau nume de spaţii depinzând de versiunea platformei .NET a aplicaţiei); membrii lor, de exemplu obiectele de tip *StringBuilder* din spaţiul de nume (*namespace*-ul) "System.Text" vor fi accesibili scriind aici, în cod, direct numele lor fără a le "califica" - de exemplu, o variabilă de tip *StringBuilder* poate fi acum declarată direct, *StringBuilder sb= new StringBuilder()*, în loc de *System.Text.StringBuilder sb= new StringBuilder()*, deşi se poate face şi asta la fel de bine – ambele declaraţii având acelaşi efect.

Unele Referinţe din această listă pot fi şterse, nefiind accesate; însă chiar acum nu ştim exact care, aşa că lăsăm lista în pace şi trecem la liniile următoare.

Linia "**namespace primulmeuprogram**" denotă aceea, că tot ce se întâmplă mai jos, după prima acoladă deschisă până la ultima acoladă închisă este asociat spaţiului de nume "primulmeuprogram". Adică, trecând mai jos:

Linia "**public partial class Form1: Form**" înseamnă că Dialogul proiectat adneauri este declarat ca o clasă cu denumirea Form1, de tipul Form – care "moşteneşte" toate atributele, Proprietăţile, membrii, Evenimentele unei clase de tipul Form.

Clasa aceasta este înăuntrul spaţiului de nume declarat mai sus, deci tot ce se petrece cu Dialogul va fi cunoscut numai în interiorul acestuia.

Mai departe se află metoda scrisă automat de mediul de programare prin care se iniţializează Dialogul, la fel ca un obiect din clasa cutare:

public Form1()

cu metoda unică "*InitializeComponent()*"

Urmează linia de cod executată la pornirea programului, la afişarea Dialogului denumit aşadar "Form1":

private void Form1_Load(object sender, EventArgs e)

în această metodă se poate scrie cod care trebuie executat concomitent cu afişarea Dialogului, la Evenimentul "*Load*"; argumentele "sender" şi "EventArgs" nu ne interesează (de cele mai multe ori).

Şi exact asta vom face – vom scrie cod. La urma urmei, de-asta ne ostenim, vrem să încropim un program funcţional.

După acolada de deschidere a acestei metode:

- stabilim Filtrul de extensii potrivite Controlului SaveFileDialog, anume fişiere cu extensia ".txt":

saveFileDialog1.Filter= "Fisiere text (*.txt)Ţ*.txt";

Citim astfel *string*-ul din care constă valoarea acestei Proprietăţi:

- Utilizatorul este interesat numai de ceea ce scrie înainte de caracterul "Ţ" – este anunţat că fişierele accesibile implicit cu acest *SaveFileDialog* sunt "Fisiere text" a căror extensie este .txt (*.txt înseamnă toate fişierele cu extensia txt din directorul/folderul accesat)
- şi ceea ce interesează numai SaveFileDialogul se află după caracterul "Ţ", anume i se spune să propună numai extensia .txt

Proprietatea *Filter* poate specifica mai multe extensii ale fişierelor de salvat, caz în care *string*-ul din care constă arată aşa – exemplu pentru două extensii; se observă poziţia caracterului separator "Ţ":

"Fisiere text (*.txt)Ţ*.txtŢFisiere imagine JPG (*.jpg)Ţ*.jpg";

Această instrucţiune care stabileşte extensiile fişierelor accesibile implicit cu SaveFileDialogul având numele "saveFileDialog1" va fi executată la afişarea Dialogului, la pornirea programului.

Se observă numărul "1" din numele "Form1" şi "saveFileDialog1". Pe măsură ce la program se adaugă Controale de acelaşi fel cu al unora existente, acelea *nefiind redenumite de la aceste nume conţinând un număr* (tot din secţiunea Proprietăţi), numărul este incrementat cu o unitate, deci următorul dialog se va numi implicit "Form2" şamd.

Aşadar, până acum am stabilit ce fişiere anume, ca extensie – respectiv, fişiere cu extensia .txt, vor putea fi salvate de către utilizator

Deocamdată nu mai avem ce face pentru momentul afişării Dialogului, aşa că trecem la ceea ce se va întâmpla efectiv în program.

Aşa că ne întoarcem în modul Design, de proiectare a Dialogului: click în stânga-sus, pe "Form1.cs îDesignş".

Mai vedem particula ".cs", după numele Dialogului (clasei) *Form1*; aceasta reprezintă extensia (.cs, de la CSharp) fişierelor care conţin interfaţa şi codul programului.

Înapoi în modul Design fiind, dublu-click pe *Button*, pentru a scrie cod de executat când Buttonul este apăsat.

Un *Button* poate fi "acţionat" astfel:

- click-stânga, adică: *deasupra Buttonului fiind pointerul mouse-ului*: 1) se apasă butonul stâng al mouse-ului, 2) se eliberează butonul stâng al mouse-ului. Evenimentul *Click* este declanşat *numai* dacă butonul stâng este a) apăsat şi 2) eliberat *deasupra Controlului*
- plasând focusul pe Control, ori selectându-l, apoi apăsând tasta Enter sau tasta Space (pentru un caracter "blank"-spaţiu liber)

După dublu-click pe *Button* suntem trimişi înapoi în modul de scriere a codului şi vedem cum mediul de programare a inserat automat o metodă care trebuie să trateze evenimentul *Click* asociat acestui *Button*:

private void button1_Click(object sender, EventArgs e)

Se observă cum, deşi am apăsat de două ori pe Buttonul tras pe *Form1* (la fel ca pe Dialog, mai devreme, pentru a accesa codul asociat), evenimentul din cod este ***Click***, nu *DoubleClick* sau altul.

Aceasta, fiindcă ***dublu-click pe un Control plasat pe un Dialog trimite programatorul la evenimentul implicit asociat acelui Control***. De exemplu, dublu-click pe un *TextBox* generează automat sau ne conduce la codul Evenimentului "*TextChanged*" al acelui *TextBox*. Deci, deşi tot de dublu-click e vorba, pe un Dialog, aceasta nu are legătură cu Evenimentul *DoubleClick* sau similar al acelui Control.

Deci tocmai am propus ca Buttonul să facă o chestie atunci când e apăsat. Ce altceva mai bun poate face, dat fiind textul pe care l-am scris pe el in secţiunea de Proprietăţi, decât să salveze într-un fişier textul din *TextBox*?..



Astfel, ca orice programator care ştie ce vrea de la un *Button* anume, n-avem încotro decât să scriem codul pentru salvarea textului într-un fişier – care fişier va fi accesibil cu Controlul saveFileDialog1.

Pentru aceasta trebuie să instruim programul că **vom lucra cu fişiere**.

Procedurile pentru lucrul cu fişiere se află în spaţiul de nume **System.IO.**
(*IO = InputOutput*)

Aşadar, suntem curioşi dacă acest spaţiu de nume apare în enumerarea de Referinţe începută pe prima linie a programului, cea cu "using...".

Şi constatăm că nu este acolo.

Aşa că, pentru a nu accesa membrii săi "calificându-i" tot scriind System.IO..., ci direct, prin numele lor, adăugăm o linie la cele existente, anume:

using System.IO;



Astfel de "artificii" care în fapt înlesnesc scrierea codului sunt obişnuite şi se învaţă repede care anume spaţii de nume conţin anumiţi membri (clase, Evenimente, metode). De exemplu, tot ce ţine de lucrul cu text "simplu" se află în spaţiul de nume System.Text; tot ce ţine de lucrul cu colecţii/dicţionare de obiecte se află în spaţiul de nume System.Collections; tot ce ţine de lucrul cu imagini se află în spaţiul de nume System.Drawing – deci, o instrucţiune *using spatiuldenumecutare;* înaintea porţiunii de cod unde vom solicita membrul cutare ne scuteşte de la a scrie toată calea spre acel membru, în stilul *spatiuldenume.numelemembruluicutare şamd*.

Textul dintr-un *TextBox*, text neformatat, aşa cum apare de exemplu în *Notepad*, poate fi salvat într-un fişier iniţializat ca *StreamWriter*.

Denumirea acestui obiect, ca de altfel a majorităţii obiectelor, spaţiilor de nume, Evenimentelor, Proprietăţilor, membrilor în general ai unui spaţiu de nume este sugestivă - în limba engleză - astfel încât programatorul măcar să bănuiască despre ce anume este vorba şi ce anume caută.

De fapt, când vedem particula *Stream*, gândul poate duce imediat la ceva continuu, un şir de lucruri, ceva care *curge*, care e *transportat* între două locuri. Astfel, *StreamWriter* se referă la un şir de octeţi (sau de biţi, 1 octet = 1 byte = 8 biţi, 1 kilobyte = 1024 bytes, 1 Megabyte = 1024 kilobytes, 1 Gigabyte = 1024 Megabytes şamd.) din care constă un fişier. La fel, obiectul *StreamReader*.

Aceste două obiecte vor fi folosite în programul nostru:

- o variabilă de tip *StreamWriter* pentru a scrie text într-un fişier
- o variabilă de tip *StreamReader* pentru a citi textul dintr-un fişier

Să începem a scrie codul de salvare a textului din TextBoxul de pe Dialogul *Form1*.

În metoda asociată Evenimentului *Click* al Buttonului scriem aceste instrucţiuni:

```
private void button1_Click(object sender, EventArgs e)
{
        StreamWriter sw = null;
        try
        {
                if (saveFileDialog1.ShowDialog() != DialogResult.Cancel)
                {
                        sw = new StreamWriter(saveFileDialog1.FileName);
                        sw.Write(textBox1.Text.Trim());
                }
        }
        catch (Exception ex)
        {
                MessageBox.Show("Eroare:\r\n\r\n" + ex.Message, "EROARE");
        }
        finally
        {
                if (sw != null)
                {
                        sw.Close();
                        sw.Dispose();
                }
        }
}
```

Instrucţiunile de mai sus salvează efectiv textul din TextBox într-un fişier. Să citim fiecare linie.

StreamWriter sw = null;
Am iniţializat variabila *sw* la valoarea *null*. Deşi aparent aşa ceva nu are rost, compilatorul de cod al versiunii cu care lucrăm de Visual Studio (2022 Community) solicită iniţializarea explicită a unei astfel de variabile/obiect.

try
Orice operaţiune cu fişiere trebuie inclusă într-un bloc *try-catch*(*-finally*).

if (saveFileDialog1.ShowDialog() != DialogResult.Cancel)
- apăsând butonul, va fi afişat (cu metoda *ShowDialog* asociată Controlului de tip *SaveFileDialog*) dialogul (o *Form*-ă în sine, binecunoscută din orice program *Windows* care manipulează fişiere) de salvare a unui fişier.
- dacă răspunsul utilzatorului la acest dialog nu este apăsarea butonului "*Cancel (Anulare)*", adică nu mai vrea să salveze textul într-un fişier, atunci:

sw = new StreamWriter(saveFileDialog1.FileName);
Atribuim o valoare utilă variabilei *sw*, pentru a o folosi mai departe (valoarea iniţială *null* fiind inutilă mersului programului în sensul dorit acum)

sw.Write(textBox1.Text.Trim());
Am apelat metoda *Write* a lui *StreamWriter* pentru a *scrie* textul (Proprietatea *Text*) din TextBoxul de pe Dialog, denumit *textBox1*.
Ce apare ciudat după proprietatea *Text* este metoda *Trim()*. Aceasta face încât textul să fie trunchiat până la spaţiile libere de la sfârşitul şi de la începutul său, dacă există astfel de spaţii libere (blank-uri) care să preceadă şi/sau să urmeze textului respectiv. De exemplu, Trim() aplicată textului " un text cu spaţii libere înainte şi după " va întoarce numai textul "un text cu spaţii libere înainte şi după".

catch (Exception ex)
Această instrucţiune din blocul *try-catch* interceptrează o eroare generică (necunoscută) survenind în decursul executării instrucţiunilor din secvenţa *try*, adică în timpul plasării (salvării) textului din TextBox în fişier. Mesajul de eroare va fi afişat cu instrucţiunea următoare:

MessageBox.Show("Eroare:\r\n\r\n" + ex.Message, "EROARE");
Secvenţa "\r\n\r\n" reprezintă două linii goale (pe care nu se află niciun text).

finally
Închidem blocul *try-catch* cu instrucţiuni de finalizare a operaţiunilor din secvenţele *try* şi *catch*; orice s-ar întâmpla în unul din acestea sau în ambele, codul din secvenţa *finally* va fi executat oricum. Ce anume exact se face în blocul *finally* este închiderea fişierului în care a fost salvat textul.
Un fişier deschis trebuie închis. Dacă nu este închis va fi inaccesibil pentru scriere şi probabil pentru citire (pentru deschidere).

if (sw != null)
Verificăm că avem ce fişier să închidem (sau să facem altceva cu el)

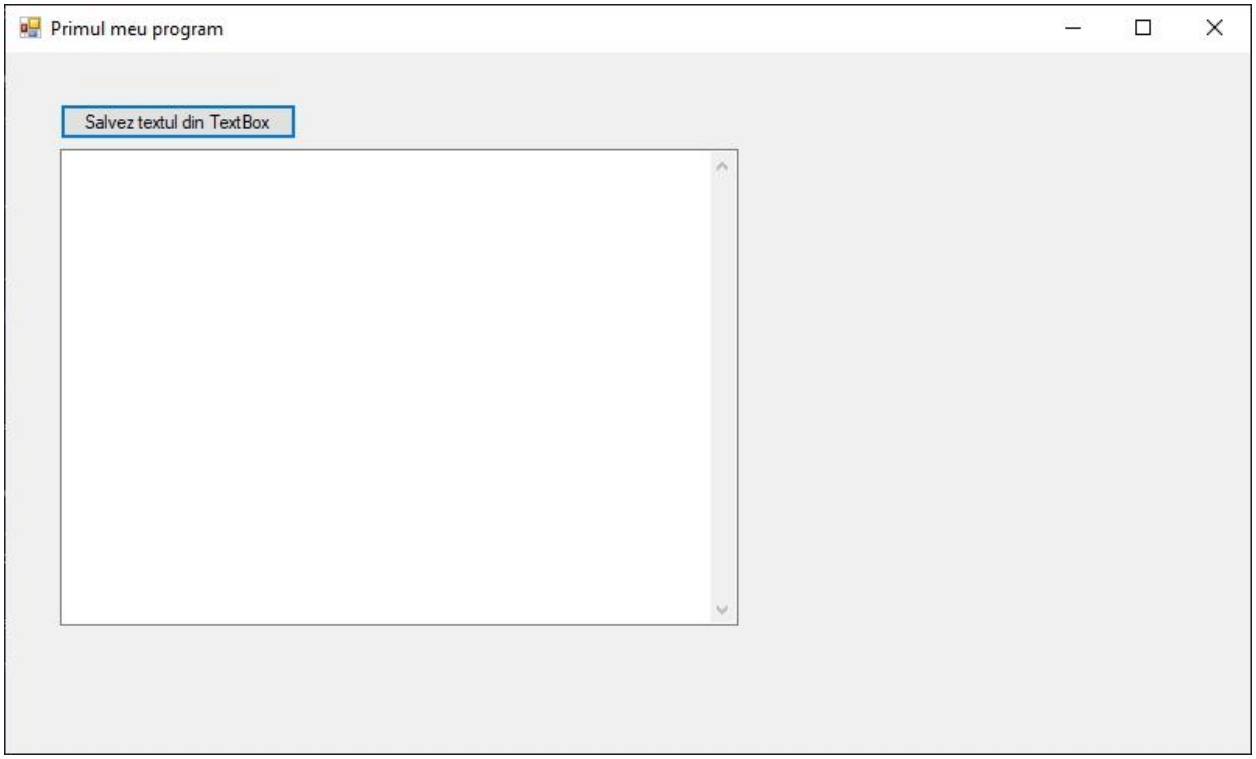
sw.Close();
sw.Dispose();
Închidem şi distrugem obiectul/variabila *sw*, fiindcă nu mai avem nevoie, orice s-ar fi întâmplat.

Acum e timpul să testăm programul. Oare *merge*? Oare face *ceea ce vrem*?

Un program şi în general un computer face ce i se spune, nu ce doreşte utilzatorul

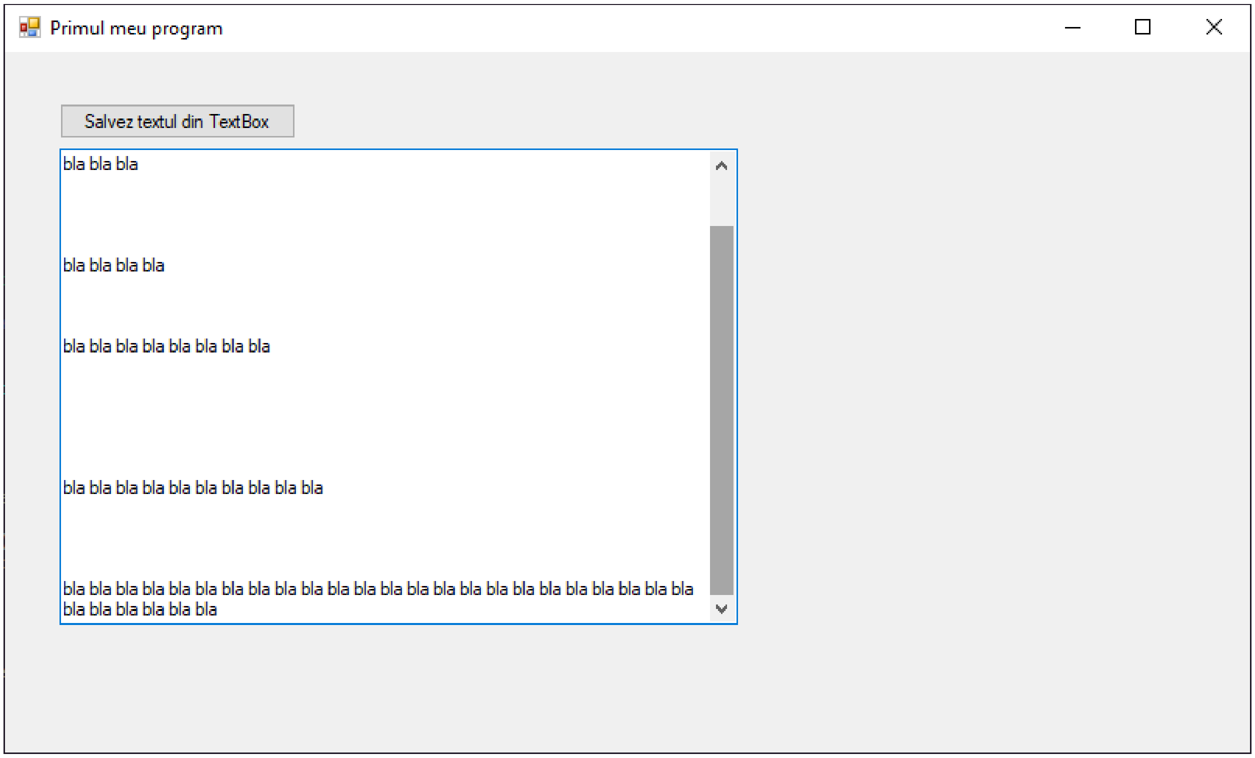
Pentru a lansa programul nostru, în Visual Studio apăsăm tasta F5 – fie din modul Design al Dialogului, fie din modul de scriere a codului, nu contează.

Rezultatul:



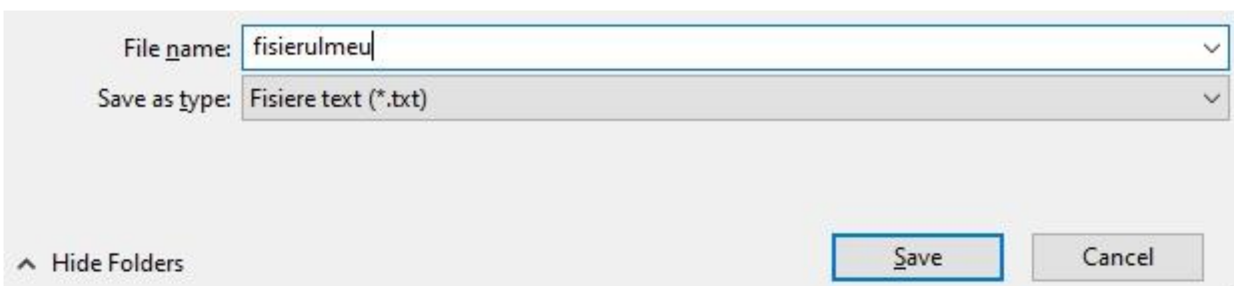
Deci acesta este Primul Dialog (singurul, de altfel, Dialogul de pornire) al Primului Nostru Program.

Scriem ceva în TextBox...



... şi dacă textul are mai multe linii decât pot fi afişate deodată în *TextBox*, apare bara de derulare verticală (în imaginea precedentă, bara este în dreapta; poziţia sa depinde de sensul în care se scrie în *TextBox* – anume, de la stânga la dreapta sau de la dreapta la stânga şi aici s-a scris de la stânga la dreapta).

Încercăm să salvăm textul, aşa că apăsăm Buttonul. Iată ce se întâmplă:



Este afişat dialogul de salvare a unui fişier, dialog al cărui titlu (pe "bara de sus") este implicit "*Save As*" – acest titlu poate fi schimbat în secţiunea Properties.

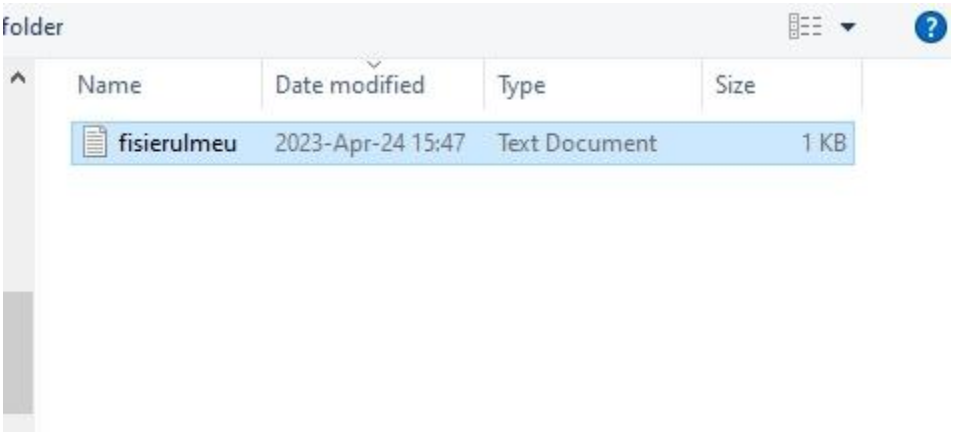
Propunem un nume (o denumire) pentru fişierul de salvat cu textul din *TextBox*, în imagine numele fiind "fisierulmeu".

Observăm cum în caseta "*Save as type*" scrie ceea ce am specificat în cod, anume o parte din Proprietatea *Filter* a acestui dialog – anume, "Fisiere text (*.txt).

Apăsăm butonul "*Save*", programul va încerca să plaseze textul în fişierul denumit cum am dorit, iar dialogul se va închide.

Dacă nu a fost niciun mesaj de EROARE putem presupune că textul a fost salvat în acel fişier.

Dar nu ştim dacă aşa este, aşa că verificăm existenţa acelui fişier:
- fie lansând programul "*My Computer*" (sau *"Windows Explorer*") şi navigând în directorul (folderul) unde am salvat fişierul
- fie apăsând iar Buttonul şi inspectând acel director, dacă într-adevăr fişierul se află acolo



Deducem din aceste observaţii ***că programul a funcţionat***:
- în directorul acela există un fişier denumit cum am specificat
- data ultimei modificări a fişierului este recentă, apropiată de ziua-ora-minutul când am rulat programul
- dimensiunea fişierului nu este 0, ceea ce înseamnă că fişierul conţine ceva şi că a fost închis

Dar nu ştim ce anume se află în fişier. Desigur că putem face click-dreapta pe el pentru a-l deschide cu Notepad (de exemplu), dar vrem să urmăm calea mai dificilă, accesându-i conţinutul chiar din programul nostru.

Pentru aceasta, închidem programul ca pe orice program care are pe "bara de sus", de obicei în colţul din dreapta-sus, acel X care odată apăsat închide Dialogul pe care se află. Programul nostru, constând într-un singur Dialog (*Form1*), va fi închis odată cu închiderea acestui unic Dialog.

Deci, vrem să citim conţinutul fişierului.
Pentru asta avem evoie să folosim un obiect de tipul *StreamReader*, menţionat mai devreme.
Acest obiect, însă, nu este suficient; avem nevoie şi de un Control de tipul *OpenFileDialog* - care face exact ce sugerează denumirea sa, foloseşte la deschiderea (accesarea) fişierelor.

Înapoi în modul Design, tragem din Toolbox pe Dialogul *Form1* un control de tipul *OpenFileDialog*. La fel ca *SaveFileDialog*, nu are un contur vizibil pe interfaţă şi este plasat jos, lângă controlul "saveFileDialog1".

Pe Dialog tragem încă un *Button* –pe care scriem eventual "Deschid un fisier-text".

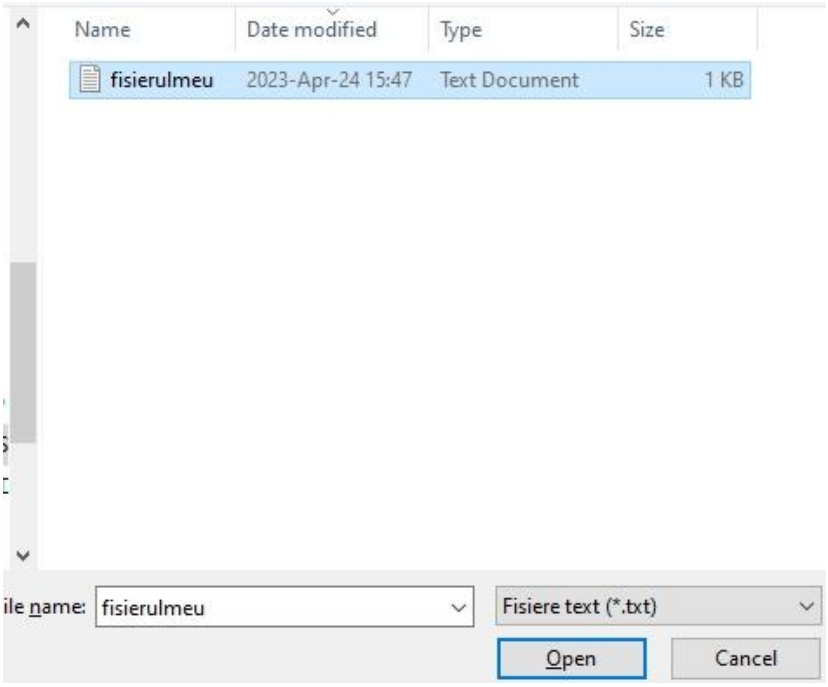
Dublu-click pe acest *Button*, ceea ce desigur că ne duce în modul de scriere a codului, în procedura Evenimentului *Click* asociat *numai* acestui *Button*; scriem instrucţiunile prezentate aici:

```
private void button2_Click(object sender, EventArgs e)
{
        openFileDialog1.Filter = "Fisiere text (*.txt)|*.txt";
        StreamReader sr = null;
        try
        {
                if (openFileDialog1.ShowDialog() != DialogResult.Cancel)
                {
                        sr = new StreamReader(openFileDialog1.FileName);
                        textBox1.Text = sr.ReadToEnd();
                }
        }
        catch (Exception ex)
        {
                MessageBox.Show("Eroare:\r\n\r\n" + ex.Message, "EROARE");
        }
        finally
        {
                if (sr != null)
                {
                        sr.Close();
                        sr.Dispose();
                }
        }
}
```

Am aplicat aceeaşi metodologie ca în cazul Buttonului prin care am salvat textul în fişier.

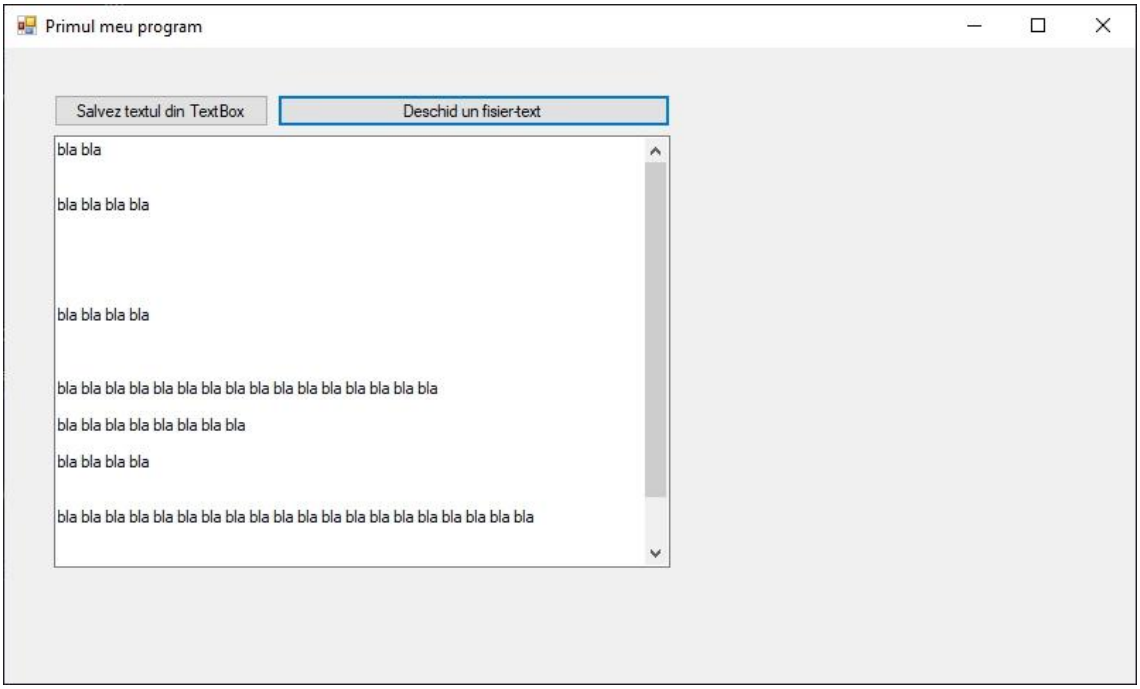
Fiecare instrucţiune este perechea sau opusul celei corespunzătoare din procedura de salvare a textului. Astfel, folosim un *StreamReader* în loc de *StreamWriter*, iar conţinutul fişierului este citit cu metoda *ReadToEnd()*.

Lansăm iarăşi programul, apăsând tasta F5, pentru a verifica procedura tocmai scrisă şi care trebuie să preia textul din fişierul selectat în *openFileDialog1* şi să-l plaseze în *textBox1*.



Selectăm fişierul, apăsăm butonul "*Open*" şi sperăm.

Rezultatul la care am sperat:



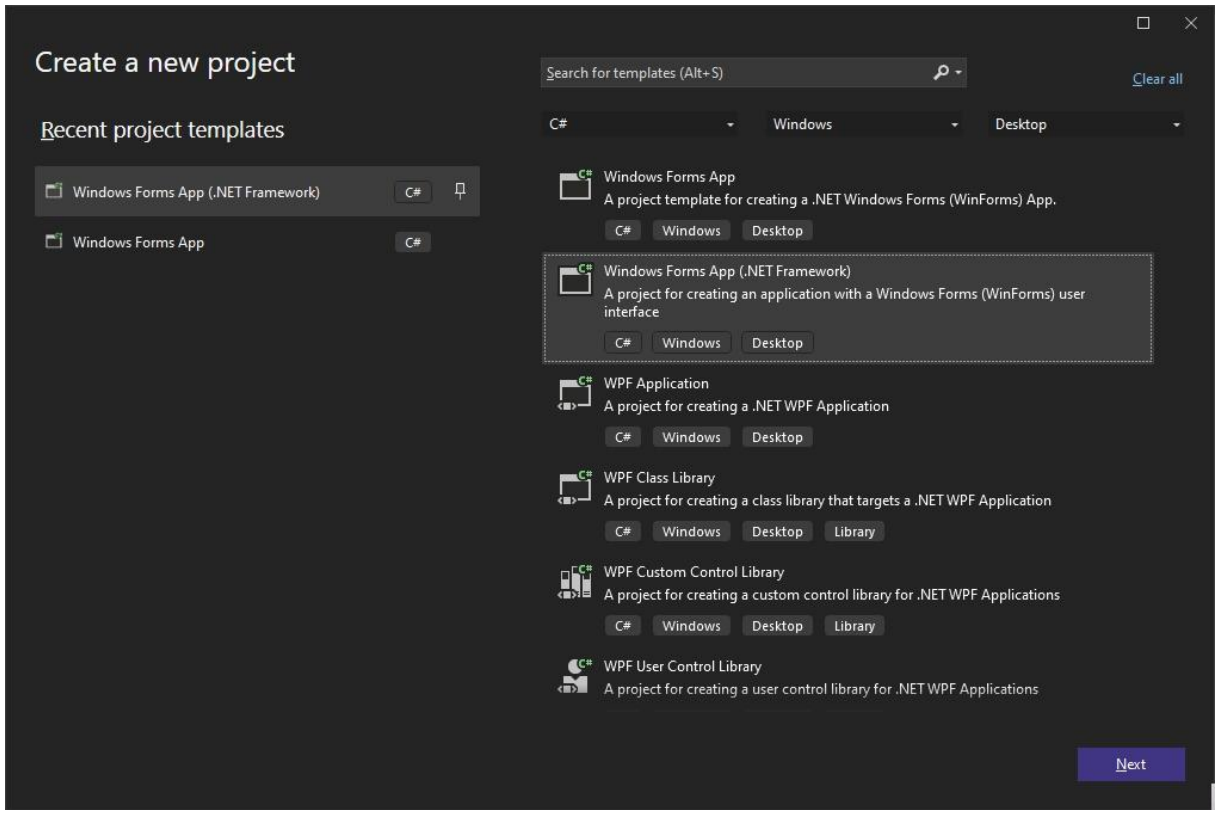
Am avut noroc, nu am greşit nimic şi textul din fişier este acum afişat în TextBox.

Cronometru

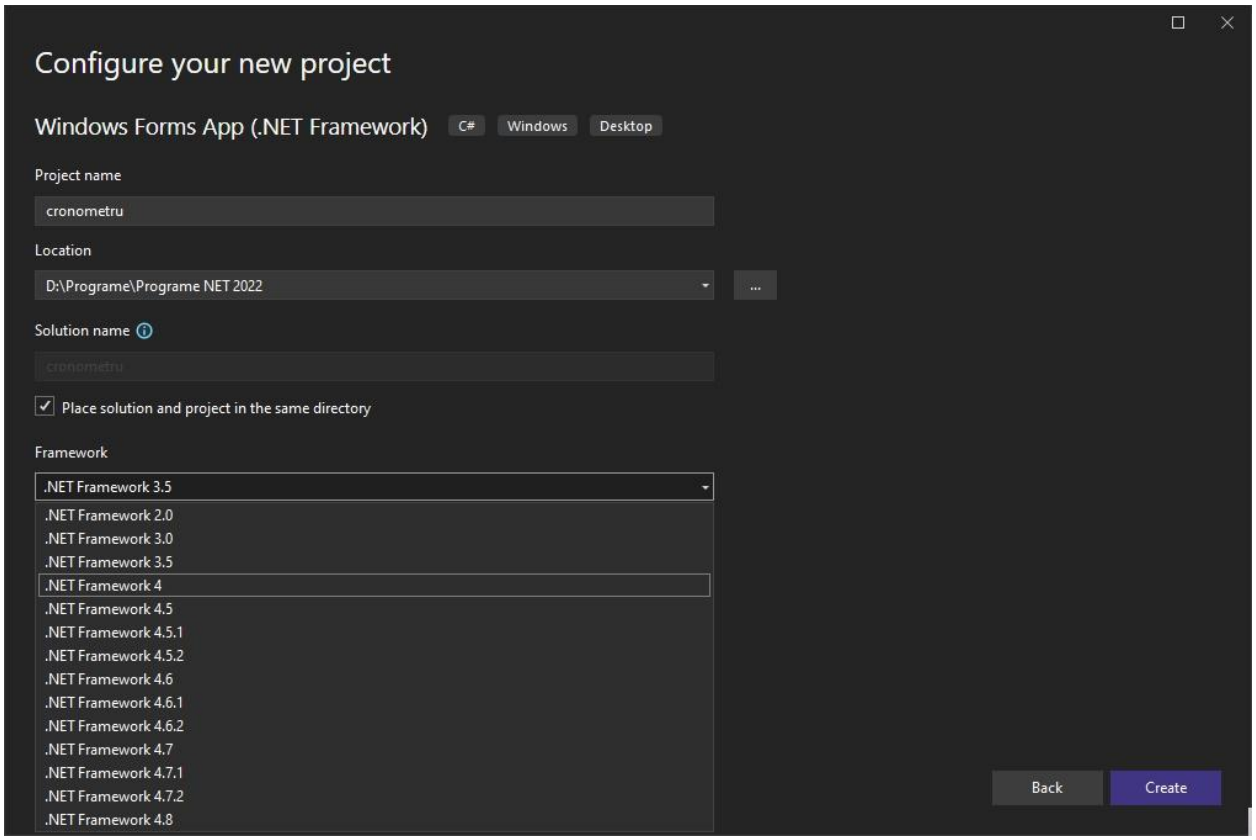
Se presupune că până acum avem cunoştinţe generale de navigare prin interfaţa Visual Studio – anume, cum să creăm un Proiect nou, cum să plasăm Controale pe un Dialog (pe o *Form*-ă) trăgându-le din Toolbox, cum să stabilim Proprietăţi ale Controalelor (click pe acel Control şi apoi tasta F4) şi cum să trecem din modul Design (Proiectare a interfeţei cu utilizatorul programului nostru) în modul de scriere a codului (respectiv, cu *Shift-F7* şi *F7*).

Vom construi un program simplu care:
- conţine un cronometru care:
- la anumite intervale afişează data şi ora curente
- la anumite minute schimbă textul afişat pe "bara de sus" a Dialogului pe care se află

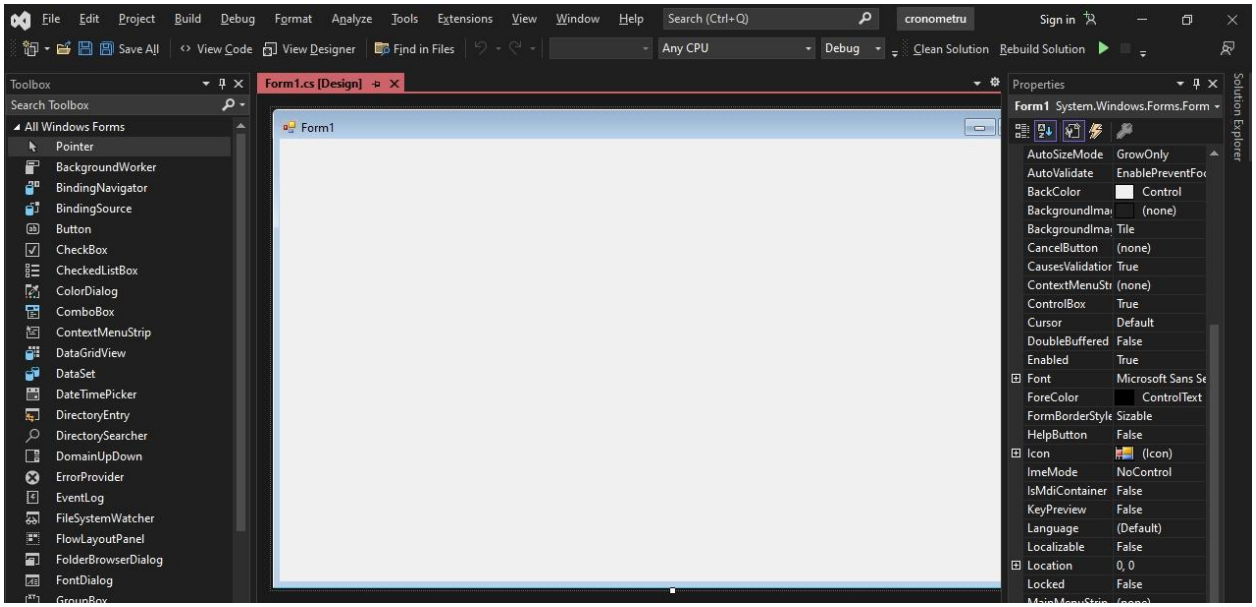
Să începem.
File -> New... Project...
Selectăm cele evidenţiate în imaginea următoare: anume, aplicaţie *desktop* ("Windows Forms App").



Next.
Dăm un nume sugestiv programului, selectăm versiunea .NET, de exemplu 4, apoi click pe "*Create*":



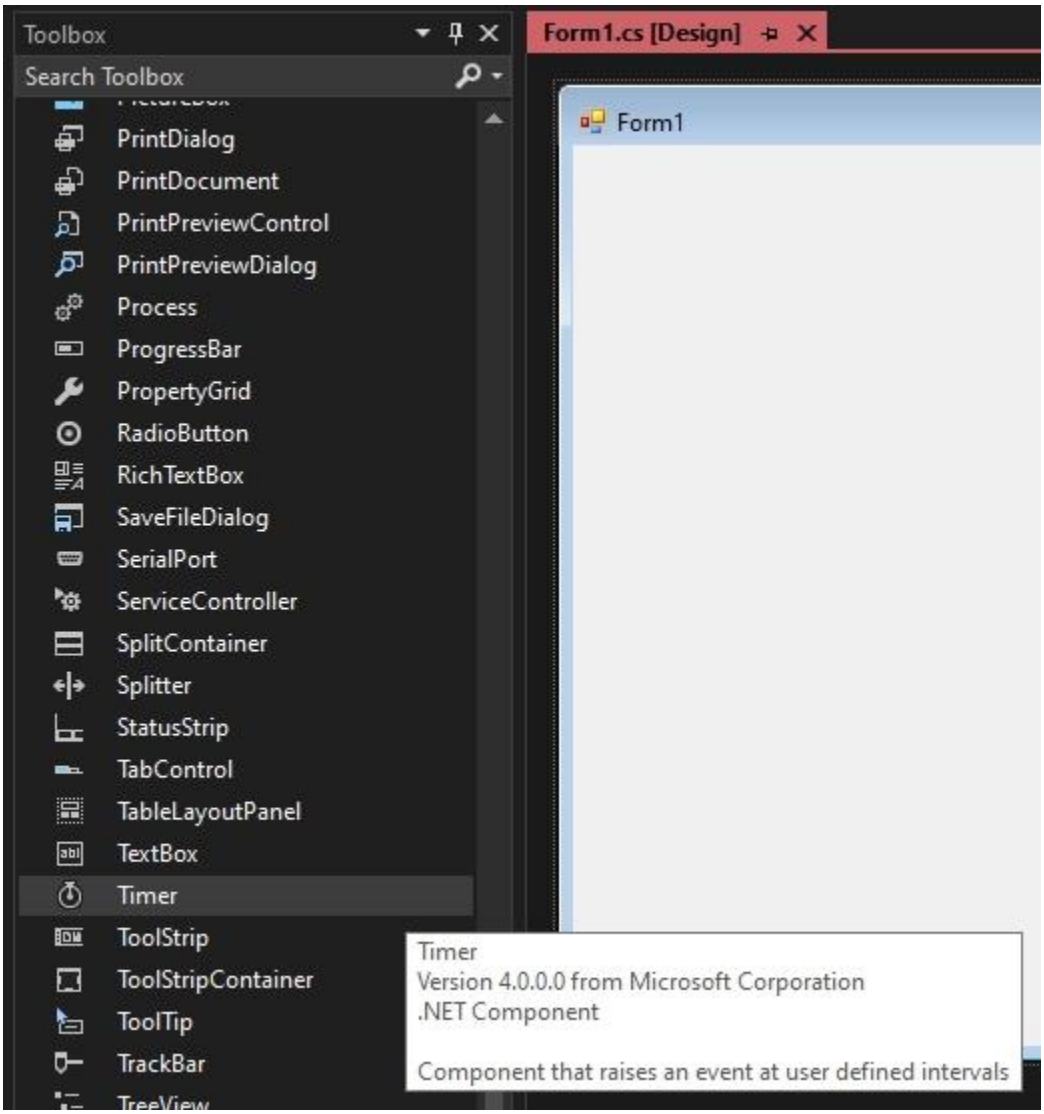
Rezultatul, un Dialog (o *Form*-ă) pe care de-acum putem plasa Controale încât să facem programul nostru funcţional şi totodată plăcut pentru utilizator.



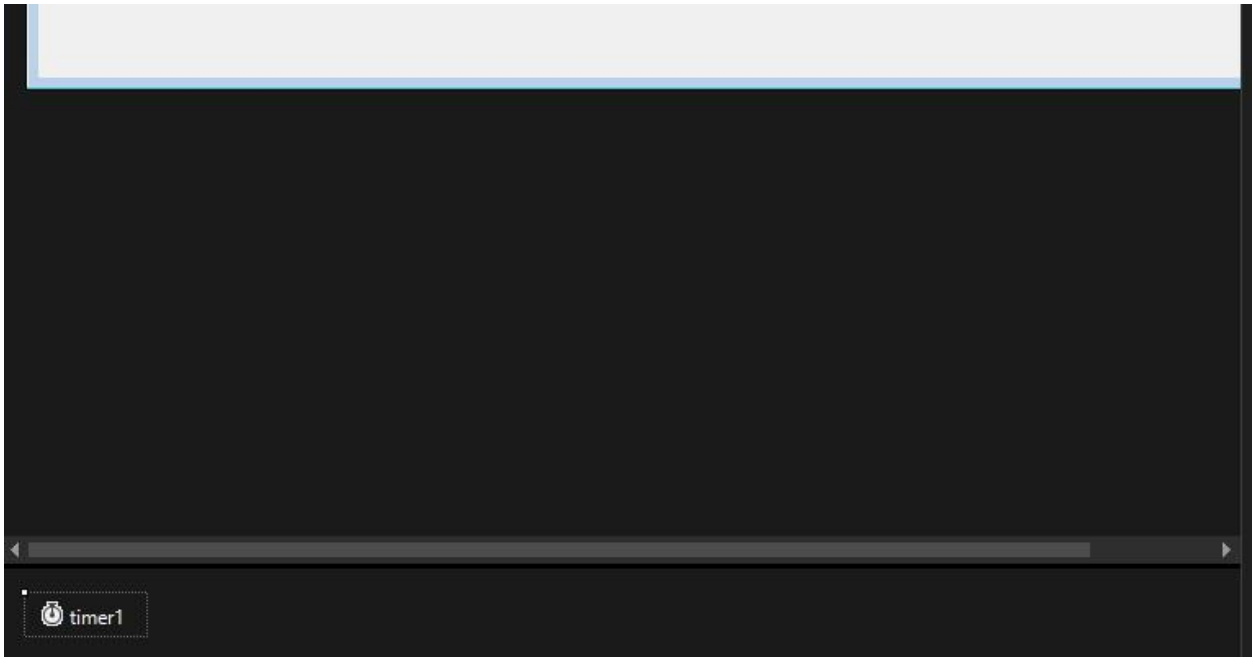
În stânga Dialogului ar trebui să fie Toolboxul cu Controale – dacă nu este vizibilă nicăieri pe ecran, din intrarea de meniu "View" se selecteaza "Toolbox" şi apoi se apasă pe "acul" acesteia din dreapta sa-sus încât să fie vertical - astfel, Toolboxul va rămâne pe ecran tot timpul, până la închiderea Visual Studio.

În dreapta Dialogului ar trebui să fie secţiunea Properties, unde pot fi stabilite Proprietăţi ale Controalelor şi ale Dialogurilor programului nostru. Dacă nu este vizibilă, atunci click pe Dialog şi apoi tasta F4, procedând la fel cu "acul" din dreapta-sus a acestei secţiuni ca în cazul Toolbox, pentru a rămâne vizibilă permanent.

Din Toolbox tragem pe Dialog un Control de tipul Timer:



Observăm cum Timerul nu apare pe Dialog (pe interfaţa cu utilizatorul, aşadar), ci undeva sub acesta. Timer este un Control care nu are un contur anume, pentru ca utilizatorul să poată interacţiona cu el, ci numai funcţionalitate stabilită prin Proprietăţile sale şi prin cod:



Click pe timer1 pentru a-i stabili câteva Proprietăţi esenţiale pentru ca Timerul acesta să aibă un rost.

În secţiunea Properties pentru orice *Timer*, atenţie mare trebuie acordă la acestea:

- *Enabled*. Specifică dacă Timerul este activ, adică dacă măsoară trecerea timpului. Este o variabilă de tip Boolean (adevărat sau fals) şi implicit are, după cum se vede, valoarea *False*; o schimbăm la True: vrem ca Timerul să măsoare curgerea timpului.
- *Interval*. Specifică numărul de ***milisecunde*** care trebuie să treacă între două operaţiuni succesive pe care acest Timer le apelează/iniţiază. De exemplu, pentru ca un anumit lucru să fie făcut la fiecare 2 minute, valoarea lui *Interval* este 2 minute x 60 secunde x 1000 milisecunde = 120000. Pentru 1 secundă, valoarea este 1000; pentru 30 secunde este 30 x 1000 = 30000; pentru 1 minut (60 secunde), valoarea este 60 x 1000 milisecunde = 60000, pentru o oră, valoarea este 60 minute x 60000 milisecunde = 3600000.

Acum stabilim *Intervalul* la 1000, adică 1000 milisecunde = 1 secundă şi specificăm ce anume trebuie să facă Timerul la fiecare acest număr de milisecunde.

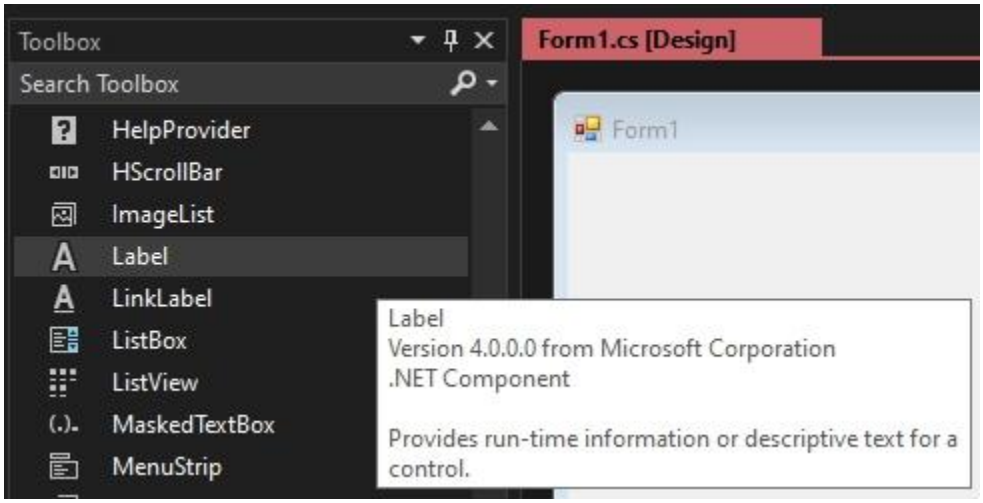
Anume, trebuie ca periodic să fie afişate **data şi ora curente, cele ale sistemului de operare *Windows* pe computerul pe care va rula acest program**.

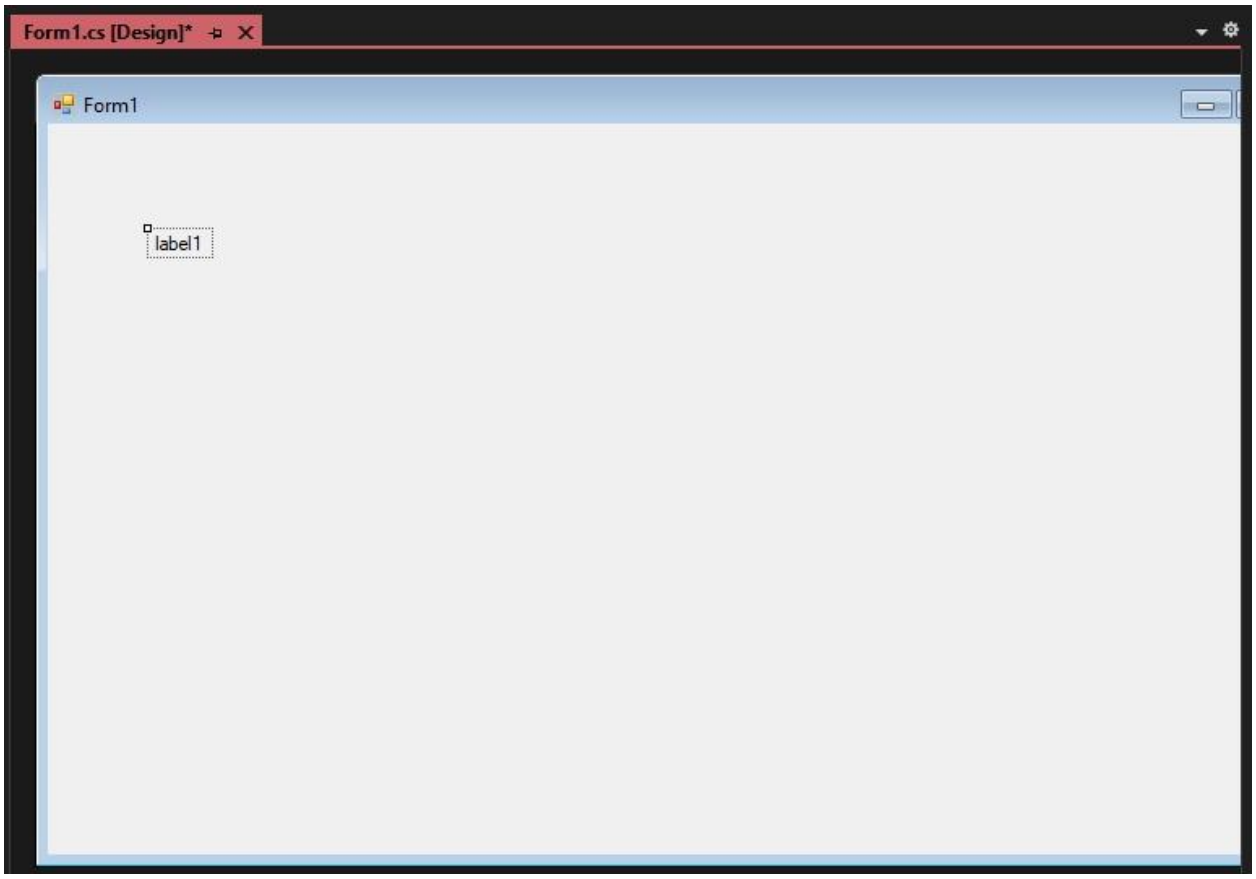
Pentru aceasta, sigur că trebuie să punem pe interfaţă un Control unde acel text să fie afişat.

Controalele comune, adecvate afişării de text sunt:

- TextBox
- Label

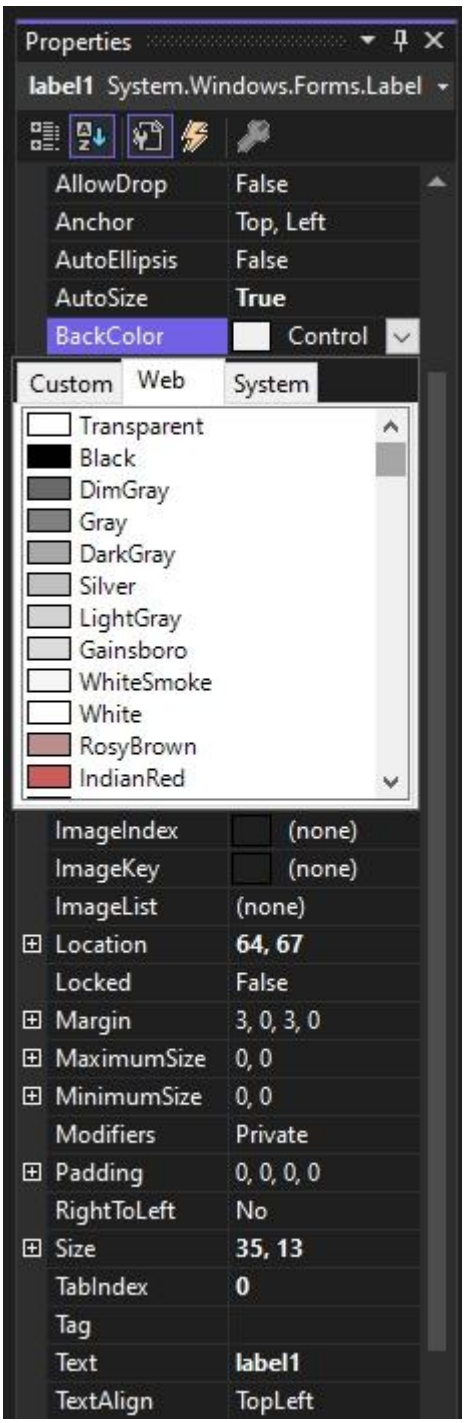
Dintre acestea, *TextBox* îi oferă utilizatorului posibilitatea de a *edita* (*schimba manual*) textul afişat.
Însă, pentru scopul *numai al afişării unui text*, Controlul *TextBox* nu este adecvat: utilizatorul nu va schimba data şi ora curente scriindu-le din tastatură.
Astfel că folosim un Control de tip *Label*, adică o etichetă simplă:





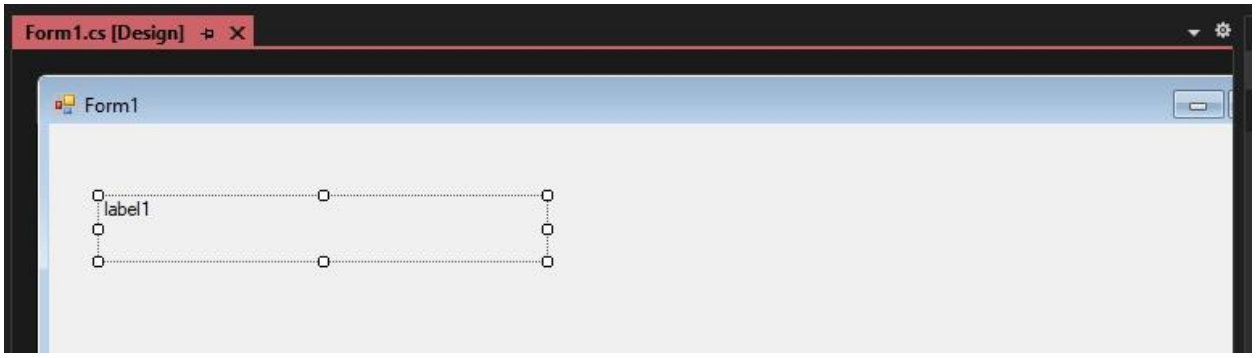
Implicit, eticheta este micuţă şi parcă nu are ce căuta pe ditamai *Form*-a. Ori mărim eticheta, ori micşorăm *Form*-a. Nu-i facem în ciudă utilizatorului, încât să-i fugă ochii pe interfaţă în căutarea a ceva care trebuie să fie foarte uşor de găsit.
Tot implicit, ca de altfel pentru orice Control nou adăugat, Textul afişat este numele acestui Control după ucm este stabilit de mediul de programare – anume, "label1". Întrucât aceasta va fi singura etichetă de pe acest Dialog şi astfel vom şti la ce ne referim când vom scrie cod, putem să nu-i schimbăm numele, de exemplu la "eticheta_dataora".

Click pe etichetă, tasta F4, schimbăm Proprietăţi:

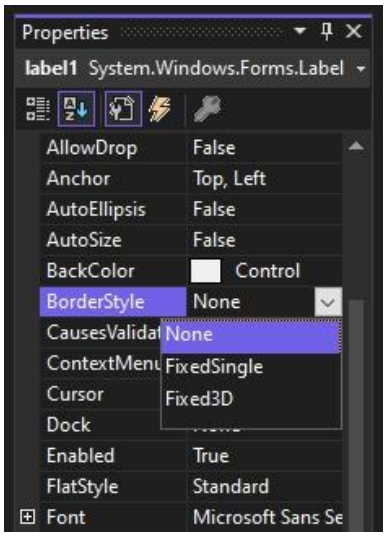


Proprietăţi în general utile (pentru afişarea unui text – acesta fiind, de fapt, scopul primar) ale unui Control de tip *Label* sunt:

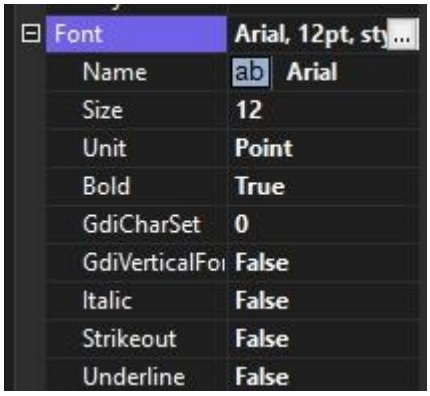
- *BackColor*, culoarea de fundal
- *ForeColor*, culoarea textului. Pentru a nu avea o interfaţă alandala, pentru a nu deranja ochii şi simţul estetic, aceste două culori trebuie să fie stabilite judicios, de exemplu nu text roşu pe fundal verde şi nu text alb pe fundal galben
- *Text*, textul propriu-zis afişat *implicit* (cel puţin la pornirea programului) pe etichetă
- *AutoSize* specifică dacă dimensiunea etichetei va fi stabilită automat la rularea programului, încât textul conţinut să afişfie afişat integral – deci, *dimensionare Automată la cea a textului*.
- *Size* specifică dimensiunea (înălţime, lăţime) a etichetei; este o proprietate comună multor Controale având un contur anume, adică vizibile pe interfaţă. Dacă însă *AutoSize* este *True*, degeaba se stabileşte *Size* şi degeaba tragem de colţurile etichetei, pentru a o dimensiona după plac. Totuşi, vrem să aibă un anumit aspect, aşa că schimbăm *AutoSize* la False şi mărim eticheta încât să cuprindă integral data şi ora curente fără a recurge la *AutoSize*, cam ca în imaginea următoare:



Dintre Proprietăţile foarte folositoare ale unui Control de tip *Label* se mai remarcă:







Am modificat:

- *BorderStyle*, la "Fixed3D"
- *TextAlign*, la "*MiddleCenter*" – textul va fi afişat centrat pe orizontală şi pe verticală, în raport cu marginile (*borders*) etichetei
- *Font*-ul, la Arial, 12pt, Bold

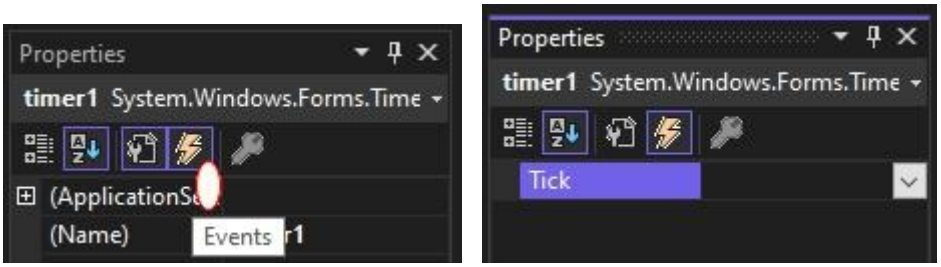
Acum eticheta arată mai decent: are margini, deci conturul său va fi văzut atât de către utilizator, ci şi de către noi, cei care proiectăm interfaţa.

Odată ce am rezolvat cu interfaţa, să începem a scrie codul pentru afişarea în etichetă a datei şi orei curente.

Astfel, trebuie să asociem această operaţiune momentului în care *Intervalul* lui timer1 a ajuns la capăt.

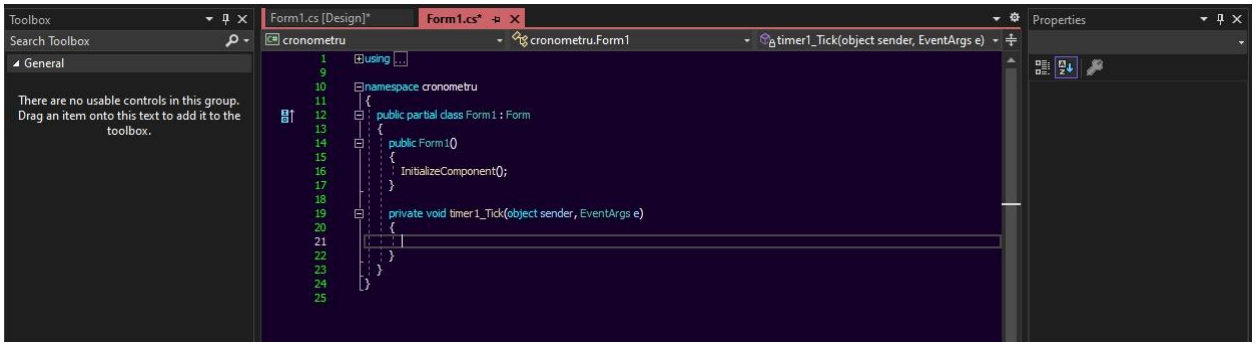
Pentru aceasta, selectăm Timerul: click-stânga, apoi ne uităm iarăşi la secţiunea Properties:
Fiindcă acolo sunt afişate şi Evenimentele asociate Timerului,
dintre care vom putea selecta Evenimentul asociat trecerii acelui *Interval.*

Această funcţionalitate este disponibilă pentru toate Controalele de pe un Dialog.



Observăm că Timerul nostru, *timer1*, are asociat numai un Eveniment – anume, *Tick* – când ceasul va tic-ăi, atunci se va întâmpla cutare – se va executa o instrucţiune.
Exact asta dorim – ca atunci când *timer1* va tic-ăi periodic la termnarea acelui *Interval* stabilit deja, pe eticheta *label1* să fie afişate data şi ora curente.

Pentru a solicita ajutor mediului de programare pentru crearea *metodei* Evenimentului *Tick*, dublu-click pe "Tick" – ceea ce ne trimite în modul de scriere a codului, unde acea metodă a fost astfel generată:



De-acum putem scrie codul pentru afişarea datei şi orei în *label1*, afişate la fiecare *Tick*-ăit al cronometrului *timer1*.

Scriem ca în imaginea următoare:

```
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Text = DateTime.Now.ToString("dddd, dd MMMM yyyy, HH:mm:ss");
}
```

Am instruit programul ca:

- la fiecare *Tick*-ăit al lui *timer1*, deci în fiecare secundă (*Interval* = 1000 milisecunde) textul afişat pe eticheta *label1* să fie:
- data curentă (un obiect *DateTime*) formatată (.*ToString*) astfel:
- numele în cuvinte ("dddd") al zilei *aşa cum se numeşte în limba sistemului de operare pe computeruil unde rulează programul* ("Friday" sau "vineri" şamd.), urmat de numărul în lună al zilei ("dd"), de numele în cuvinte ("MMMM") al lunii (cu aceeaşi observaţie ca pentru numele zilei), de numele complet al anului ("yyyy"), de oră ("HH"), minute ("mm") şi secunde ("ss").

Şi acum, în sfârşit, lansăm în execuţie (pornim) programul nostru. Apăsăm tasta F5 şi sperăm la succes.



Minunat. Programul funcţionează şi chiar face ce am vrut: vedem cum secundă de secundă, la fiecare 1000 milisecunde (*Intervalul* la care deci *Tick*-äie periodic cronometrul *timer1*), pe eticheta *label1* sunt afişate data şi ora curente, *ale computerului pe care rulează programul*.

În continuare, să facem încât la numai ***anumite*** minute sau secunde (deci, nu la fiecare *Tick*-ăit/*Interval*) să fie executată o operaţiune.

În acest exemplu, ne referim la un număr de secunde care este divizibil cu 10.
Adică, să se execute o operaţiune de fiecare dată când numărul secundelor este: 0, 10, 20, 30, 40, 50.

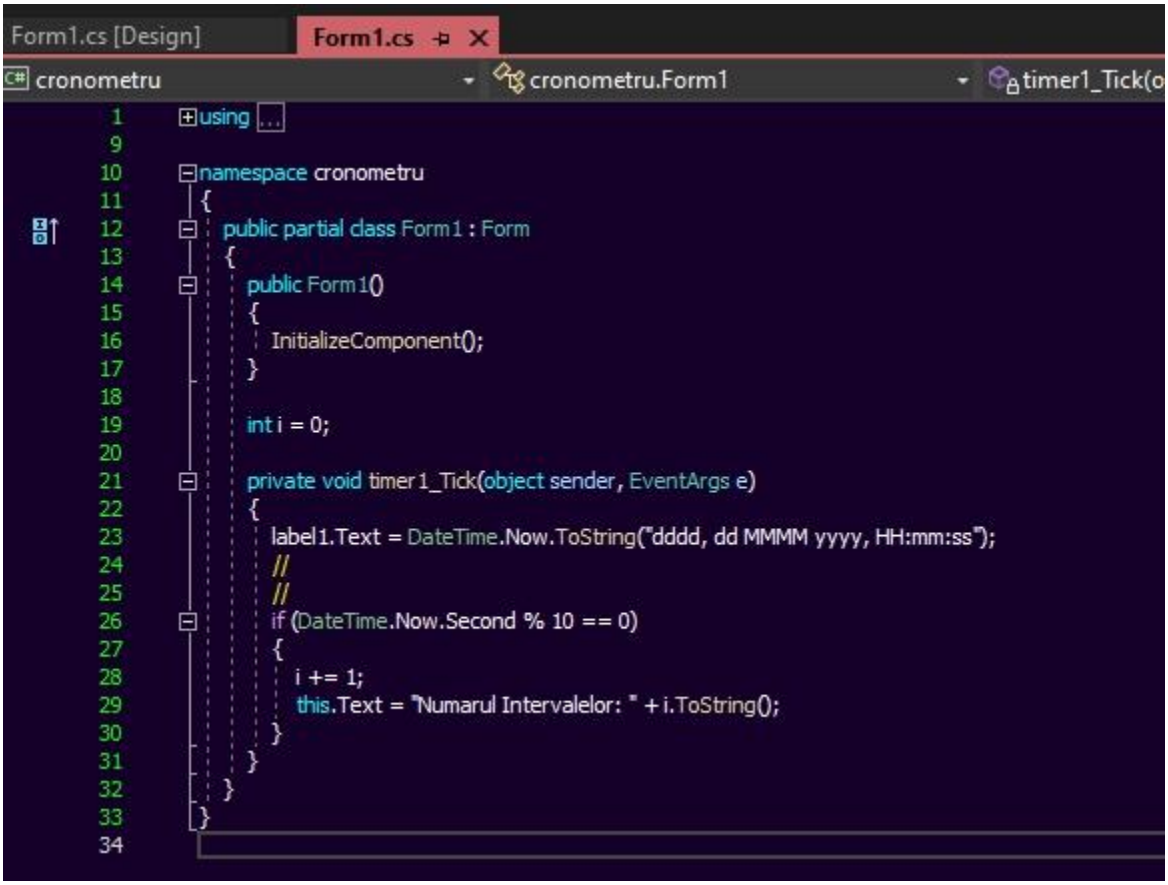
Vom schimba Textul (titlul) *Form*-ei (Dialogului), la:
"Numarul intervalelor: " urmat de 1, 2, 3 şamd – număr incrementat cu o unitate de fiecare dată când cel al secundelor este divizibil cu 10.

Pentru aceasta, în cod adăugăm linia:

```
int i= 0;
```

Aceasta (i) este o variabilă pe care o declarăm iniţial cu valoarea 0 şi care aici semnifică *numărul intervalelor de 10 secunde trecute de la începerea cronometrării*.
Desigur, cronometrul poate porni (odată cu programul) la secunda 9 sau 16, caz în care numaidecât se ajunge la o secundă (10, respectiv 20) divizibilă cu 10; ceea ce înseamnă că variabila *i* va fi incrementată cu 1 şi când nu dorim (deci nu numai după fiecare 10 secunde) – dar deocamdată ne mulţumim cu asta.

După ce scriem cele necesare codul va arăta astfel:



Am scris un bloc *if* după două *linii de comentariu*: după două slash-uri directe (//) putem scrie observaţii la cod – orice cuvinte, fiindcă nu vor fi interpretate drept cod:

//aceasta este o linie unde explic ce voi face in continuare
//asta e inca o linie de comentarii la codul de aici sau din alta parte a programului sau despre altceva

Instrucţiunile fiind citite astfel:

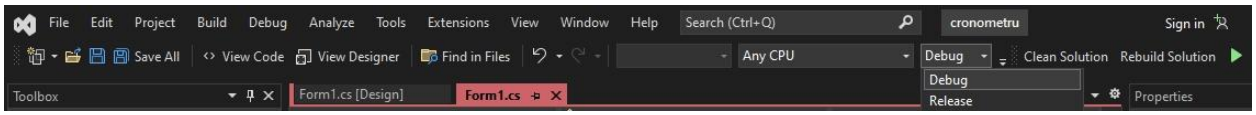
Dacă (numărul secundelor din data-ora (*DateTime*) curentă (DateTime.Now.Second) este divizibil *modulo* cu 10, deci restul = 0)
atunci
valoarea variabilei *i* declarate mai sus se incrementează cu 1 (cu o unitate)
apoi se stabileşte Textul (titlul) acestei (*this*) *Form*-e la "Numarul intervalelor: " urmat de valoarea transformată la text a variabilei i

Mai departe, dacă vrem putem închide programul pentru a schimba Proprietăţi ale *label1* ca BackColor, ForeColor, Size, sau *Intervalul* lui *timer1* şi lansăm iar programul. Repetăm până ne convine rezultatul.

Proprietăţile Controalelor nu pot fi modificate în timpul rulării programului.
Cel mult, unele pot fi modificate – dar schimbările nu vor fi reflectate în programul deja rulând.

Acum că avem deja două programe funcţionale, e cazul să ştim şi ***unde anume se află*** în computer, ca fişiere executabile care pot fi transferate, plasate pe alte computere.

Ne uităm la bara de unelte a Visual Studio:



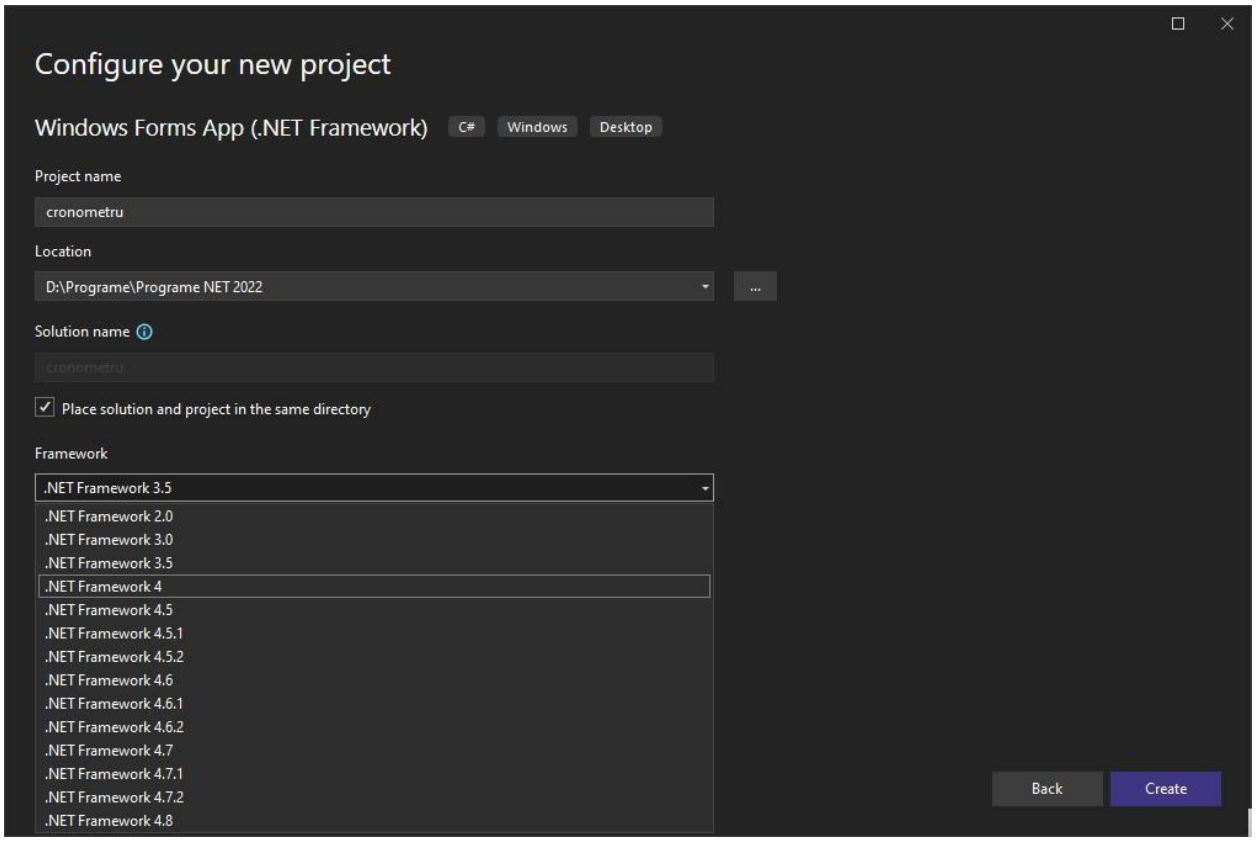
Pictogramele şi textele afişate aici, de exmeplu "Save All", "View Code", "Find in Files", "Any CPU", pot arăta altfel, sau pot fi dispuse altfel decât în imaginea precedentă.

Totuşi, trebuie să existe caseta-listă unde scrie "Debug"

Aceasta se referă la tipul executabilului creat de Visual Studio. Implicit, versiunea este "Debug" şi poate fi schimbată la "Release", după care programul trebuie lansat iarăşi pentru a fi creat acel executabil. Sau invers, de la "Release" la "Debug". Diferenţele între *Debug* şi *Release* ţin de alcătuirea fişierului executabil care în cazul programului nostru se numeşte "cronometru.exe".

Locul unde acest executabil (deci, fişierul programului pe care tocmai l-am construit şi rulat, executabilul care poate fi copiat, duplicat, şters, redenumit ca orice fişier) depinde de versiunea, *Debug*/*Release* a programului, fiindcă fiecare din acestea două beneficiaz a de propriul director/folder.

Ne amintim, dacă am uitat, unde anume am creat Proiectul acestui program:

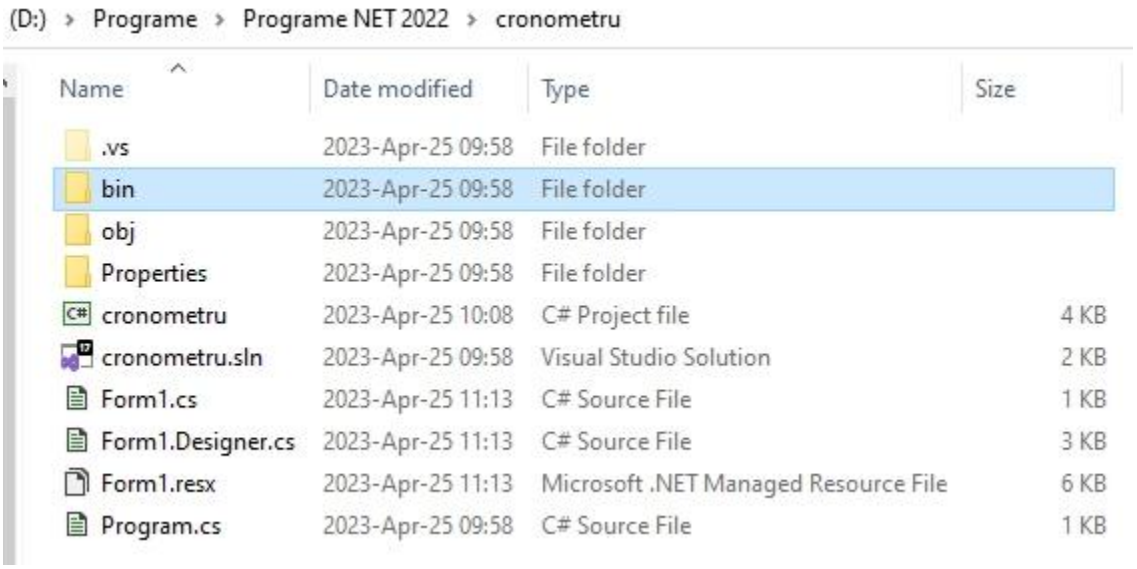


L-am creat în "Location:"
D:ţProgrameţPrograme NET 2022

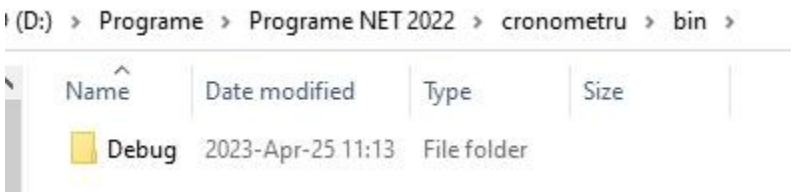
Ei bine, navigăm la acel director/folder şi căutăm executabilul "cronometru.exe":



Din directorul denumit în acelaşi fel cu programul nostru şi unde se află toate componentele programului ne interesează subdirectorul "bin:

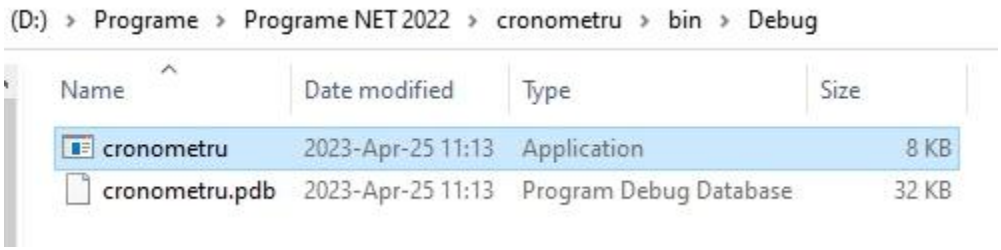


În acest director, "bin", se află subdirectoarele corespunzătoare celor două verisuni de executabil – *Debug* şi *Release*:

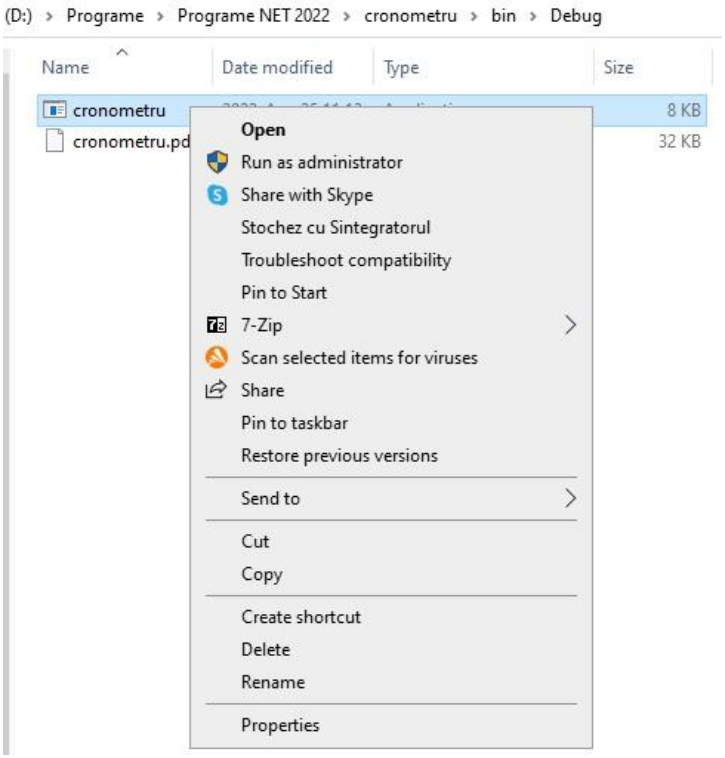


În cazul acesta, aici găsim numai subdirectorul "Debug" – fiindcă am construit numai versiunea *Debug*, adică neselectând "*Release*" în acea casetă cu listă de pe bara de unelte a Visual Studio, menţionată mai sus şi relansând programul în acea versiune.

Iar în "Debug" iată şi programul nostru, ca fişier executabil:



De acum, cu acest fişier executabil putem face orice se poate face cu oricare alt fişier (de acest tip): click-dreapta pe denumirea sa:



Alături de fişierul executabil (cel cu extensia .exe) există unul denumit la fel, dar având extensia ".pdb". Acesta este un fişier conţinând simboluri pentru depanarea programului; nu este necesar rulării executabilului, acesta având nevoie numai de prezenţa pe computerul unde rulează a versiunii platformei .NET în acord cu care a fost construit.

Totuşi, menţiunea dinainte referitoare la prezenţa numai a platformei .NET este adev ărată numai în cazul programelor care nu folosesc module create de terţi şi/sau diverse resurse (informaţii diverse, de exemplu imagini) apelate în timpul rulării dar inexistente sau inaccesibile.

Evaluarea unor variabile numerice şi interacţiunea lor cu valori de tip text

Va fi creat un program unde vor fi puse în corespondenţă numere (variabile de tip numeric) şi texte (variabile de tipul *string*).
În funcţie de o valoare numerică se va determina care text anume va fi preluat şi folosit mai departe.

Anume, două *tablouri* (*Arrays*) – unul populat de numere şi celălalt de texte, vor fi parcurse încât să extragem informaţie referitoare la relaţia dintre ele.
Ambele tablouri vor avea o singură dimensiune – deci, valori înşiruite una după alta, de exemplu
"cald", "frig", "caldut", "racoare", "agreabil", "tolerabil".

Observăm particula "da_" dinaintea denumirii variabilelor pe care le declarăm.
Aceasta este o *convenţie de denumire a variabilelor*; fiecare programator îşi poate proiecta propriul set de convenţii care să ajute la identificarea *tipului* entităţilor dintr-un program, de exemplu:

- *Form*-ele (*Dialog*-urile) ar fi denumite după modelul: "forma_numeleformei"
- variabilele de tip *string* ar fi denumite după modelul "str_numelevariabilei"
- variabilele numerice de tip *double* ar fi denumite după modelul "dn_numelevariabile"
- variabilele de tip NameValueCollection ar fi denumite după modelul "nvc_numelevariabilei"
- şi tot aşa
- constantele de diverse tipuri ar fi denumite similar

Aici, tabloul "numeric" va fi declarat/iniţializat astfel ("da_" sugereaza "*doublearray*_numitastfel"):

doubleîş da_temperaturi= new doubleîş Î1, 2, 3, 4.39, 5.8, 6, 7.26, 8, 9.12, 10Ş;

Tabloul "de *string*-uri" va fi acesta:

stringîş sa_parere= new stringîş Î"frig", "racoare", "tolerabil", "caldut", "cald"Ş;

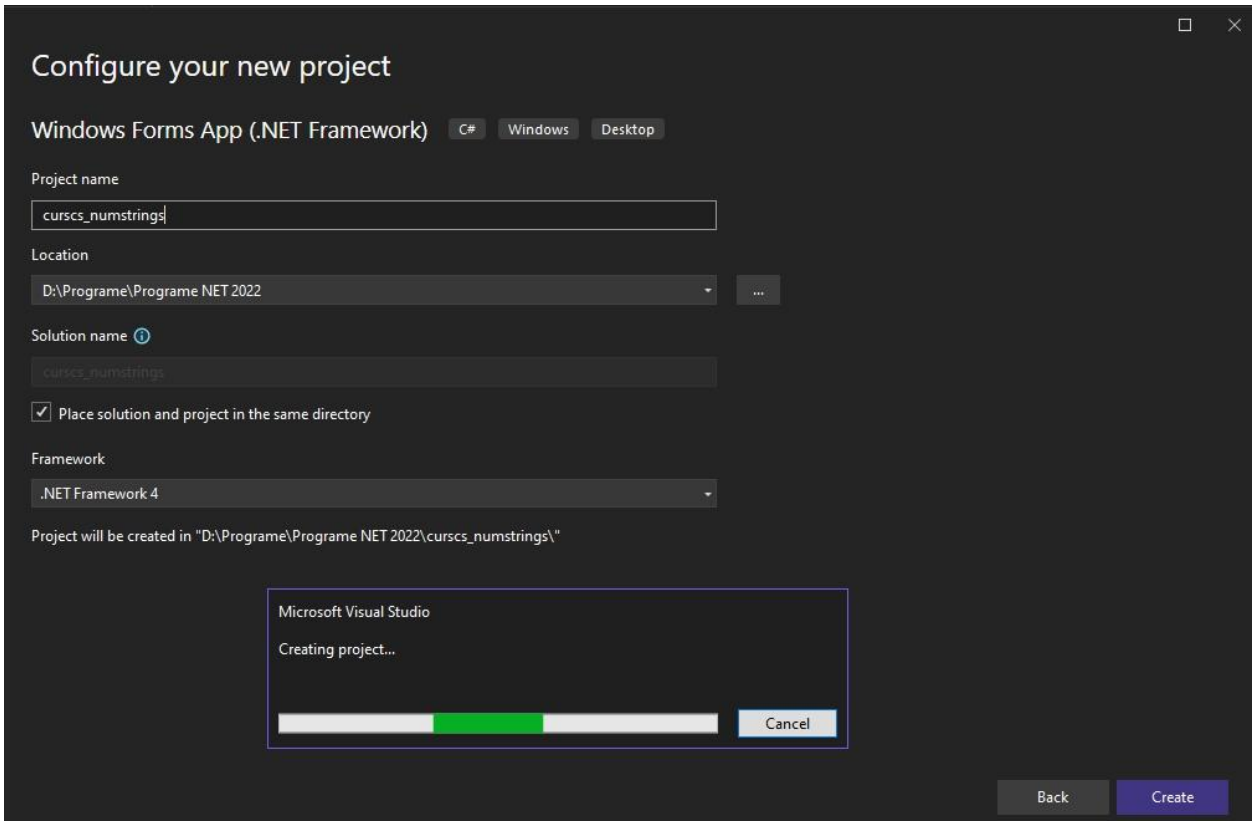
Se observă că tablourile nu au acelaşi *număr de elemente*:

- tabloul "numeric" are 10 elemente
- cel "de *string*-uri" are 5 elemente

Pentru scopul programului, nu contează câte elemente are fiecare tablou.

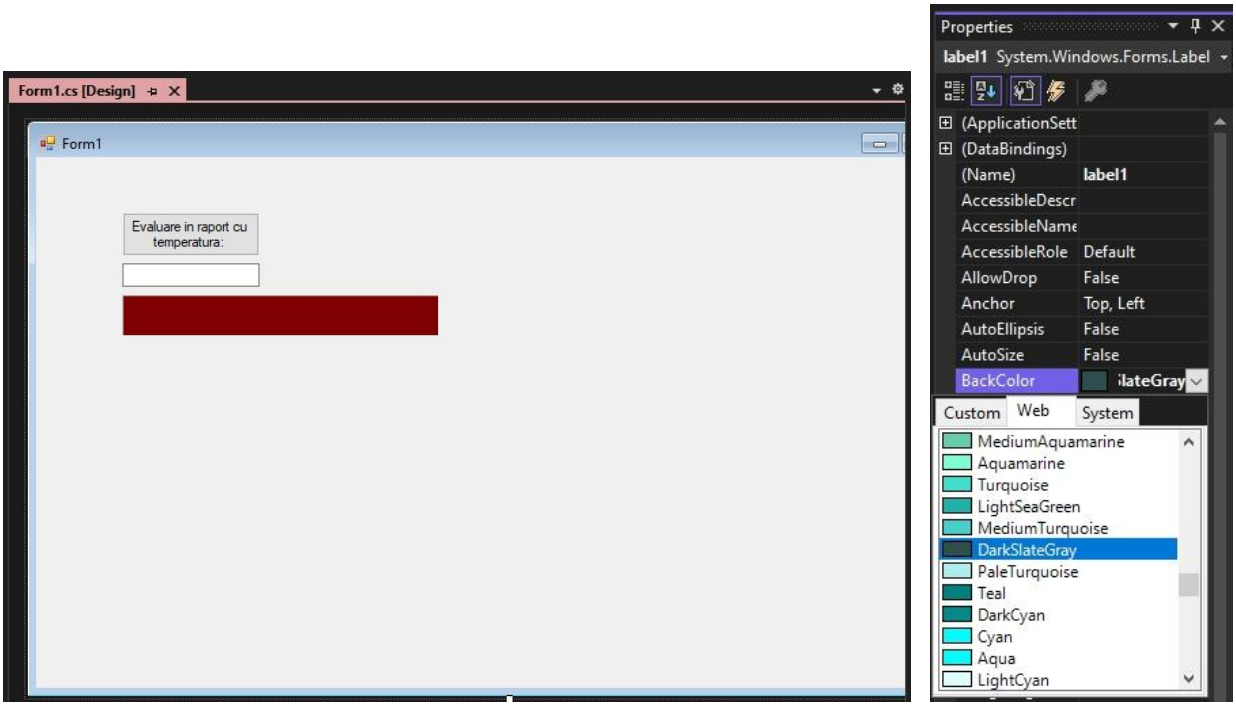
Să începem:

Pornim Visual Studio, *File -> New... Project...*



Pe Dialogul "Form1" creat implicit tragem câteva din Toolbox aceste Controale:

- un buton (*Button*) – denumit implicit "button1"
- o etichetă (*Label*) – denumită implicit "label1" şi căreia în acest exemplu i-am modificat proprietatea "BackColor" la "DarkRed" sau "DarkSlateGray", "ForeColor" la "White" (oricum, culori cât de cât contrastante care astfel să asigure lizibilitatea textului)şi "AutoSize" la "False", după care am redimensionat-o trăgând de colţul din dreapta-jos al său
- o casetă de editare a textului (*TextBox*) – denumit implicit "textBox1" şi căruia i-am modificat Proprietăţile: "TextAlign" la "Center" şi "Font" cu "Size" 12 şi "Bold" True:



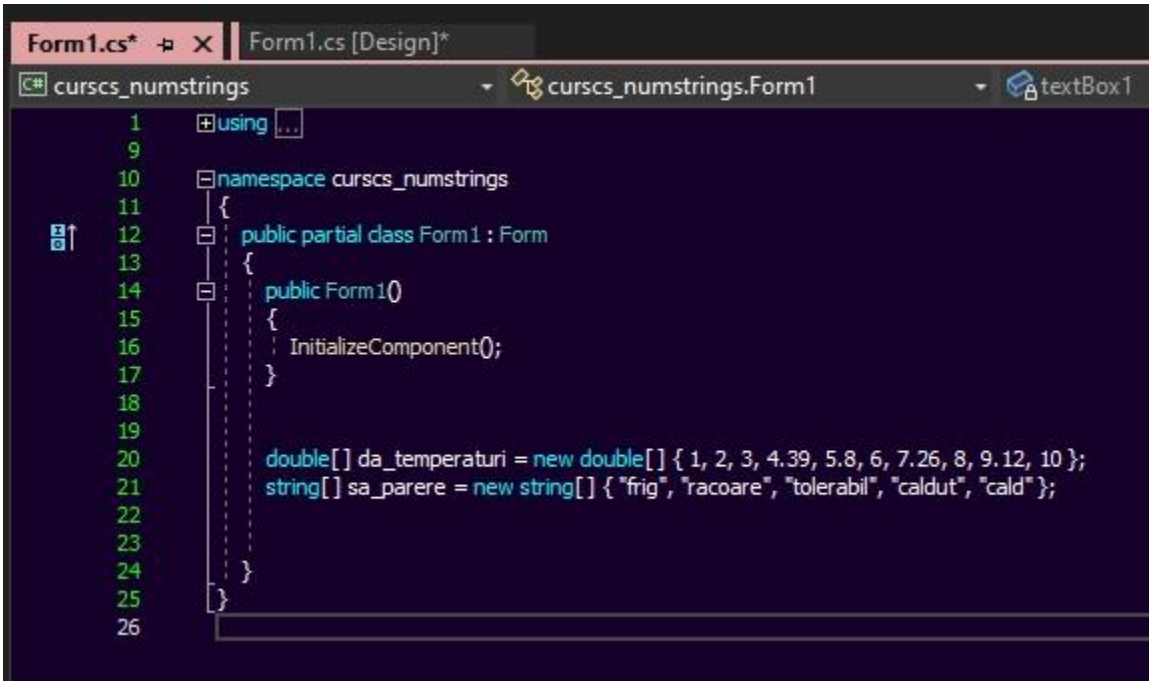
Am redimensionat Controalele după bunul plac.
Ar trebui deja să ştim cum să proiectăm o interfaţă decentă, nu extraordinară, ci una care să nu forţeze utilizatorul să caute un Control care ar trebui să fie plasat într-un loc accesibil fără dificultate.

Să scriem codul programului.

Click pe *Form1* şi apoi pe tasta F7:

Suntem trimişi în modul de scriere a codului in Visual Studio.

Scriem declaraţiile celor două variabile-tablou:



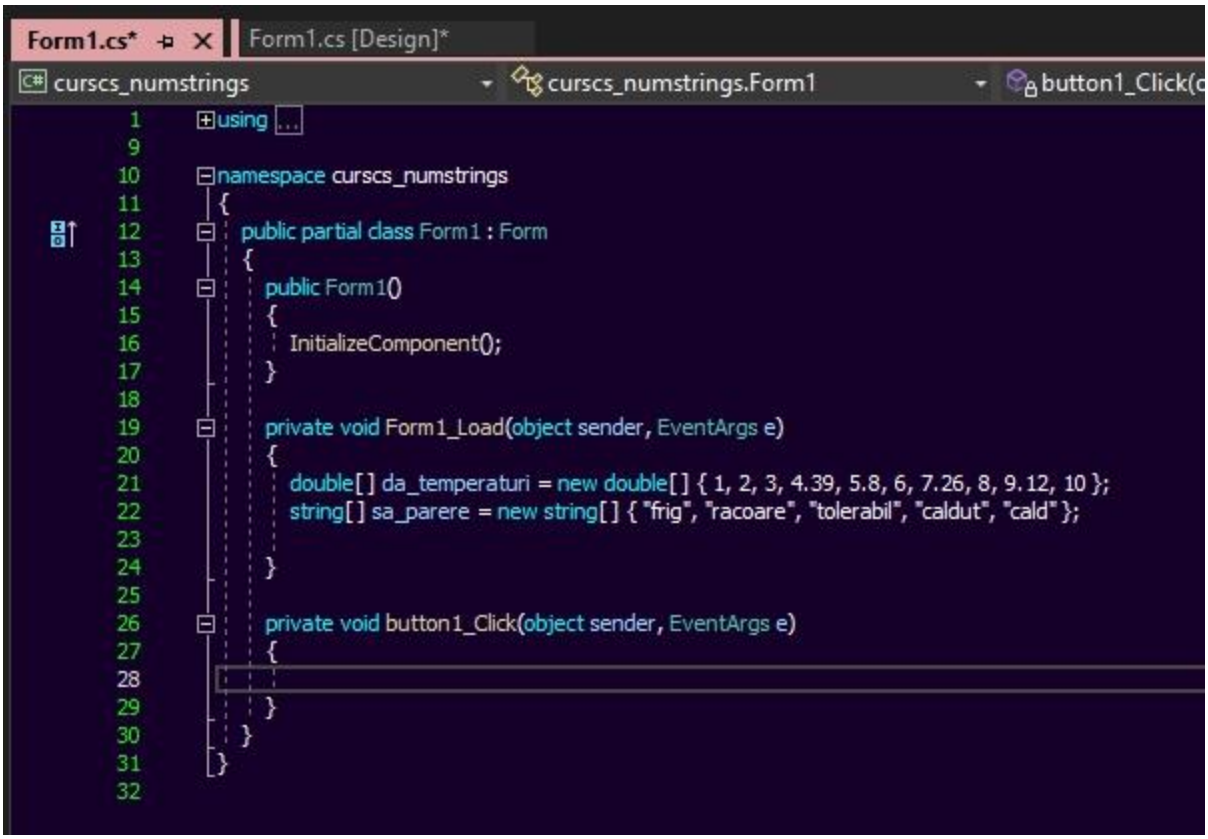
Pe *Form1* am plasat acel *TextBox* ("textBox1") în care vom scrie un număr în funcţie de care vom prelua un text din tabloul tocmai declarat.

Pentru a efectua acea evaluare a numărului scris în caseta de text trebuie să apăsăm butonul "button1".

Deci trebuie să creăm Evenimentul *Click* asociat lui "button1":

Pentru asta, desigur, ne întoarcem în modul Design, de proiectare a Dialogului.

Dublu-click pe buton, ceea ce ne trimite înapoi în modul de scriere a codului, direct în metoda asociată Evenimentului *Click* al lui "button1":



Ceea ce vrem să se întâmple după ce apăsăm (*Click*) pe butonul "button1" este:

- evaluarea numărului scris în caseta "textBox1"
- asocierea acelui număr cu un element din tabloul "de *string*-uri" pentru
- afişarea acelui *string*

Scriem acestea:

```
if (textBox1.Text.Trim() != "")
Î
        string snum = textBox1.Text.Trim();
        double dnum = double.Parse(snum);
        //in instructiunea deasupra acestui comentariu,
        //daca textul nu poate fi interpretat (citit) si transformat de compilator
        //ca/in valoare numerica, atunci programul se intrerupe
        if (
        (dnum >= 1) & (dnum <= 5)
        )
        Î
                label1.Text = sa_parereî0ş;
                //frig
        Ş
        else if (
                (dnum > 5) & (dnum <= 10)
                )
        Î
                label1.Text = sa_parereî2ş;
                //tolerabil
        Ş
Ş
```

Citim aceste linii:

if (textBox1.Text.Trim() != "")
Dacă textul fără spaţiile goale ***de la începutul şi de la sfârşitul său*** nu este un text nul (deci în caseta de editare a textului se află fie şi numai 1 caracter diferit de spaţiul *blank*), atunci

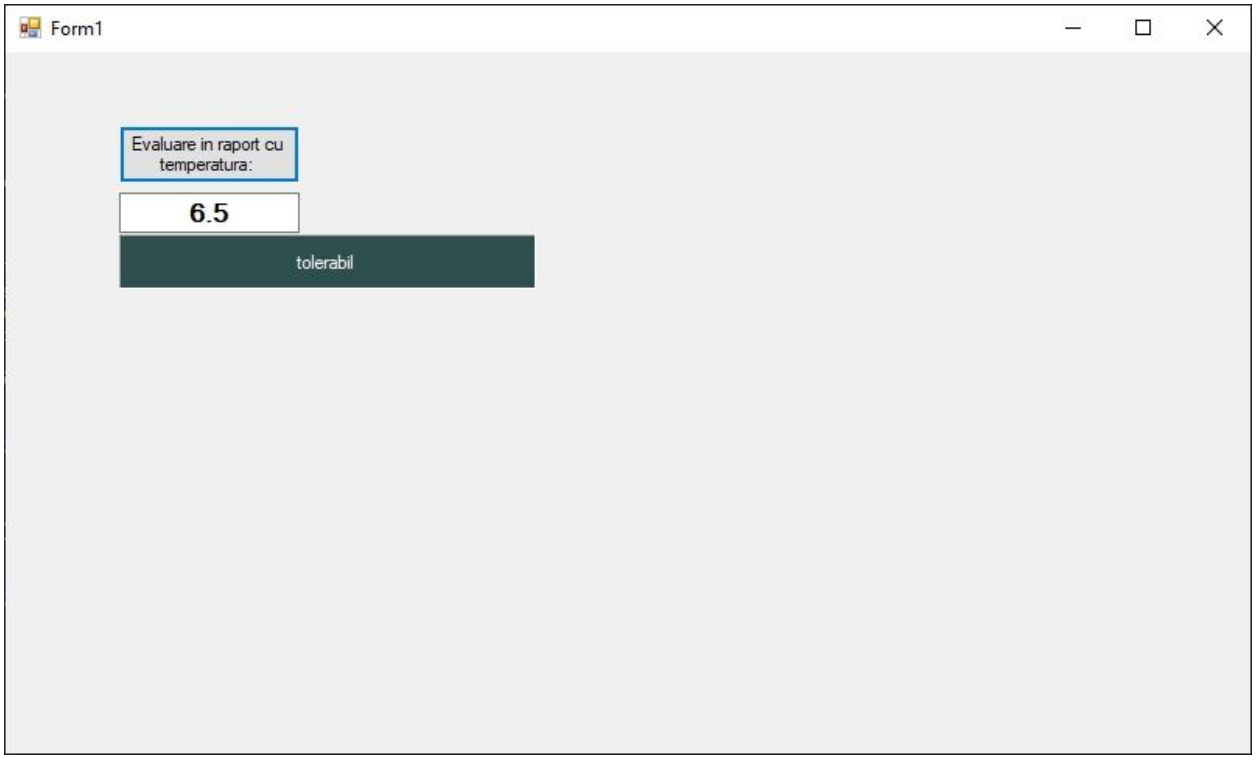
string snum = textBox1.Text.Trim();
double dnum = double.Parse(snum);
declarăm încă două variabile:

- una care conţine textul din "textBox1"
- şi una de tip *double* care provine din *conversia variabilei de tip string la tipul* ***numeric*** *double*

Blocul if-else care urmează acestor instrucţiuni este uşor de înţeles şi de construit pe ideea: dacă valoarea lui *dnum* obţinută adineauri se află între anumite intervale numerice, atunci în fiecare caz se preia un element din tabloul "de *string*-uri":

- dacă se află ***între 1, inclusiv şi 5, inclusiv***: atunci se preia primul element,
  aflat la poziţia/indexul 0
  şi este afişat în eticheta "label1": label1.Text = sa_parereî0ş;
- dacă este ***mai mare decât 5 şi este mai mic decât 10, inclusiv***, atunci se preia al treilea element,
  aflat la poziţia/indexul 2
  şi este afişat în eticheta "label1": label1.Text = sa_parereî2ş;

Să testăm programul nostru: apăsăm tasta F5:



Cum am evaluat numărul **6.5** şi acesta se potriveşte cazului:

else if (
(dnum > 5) & (dnum <= 10)

a fost preluat al treilea element (aflat în tablou la indexul 2) din tabloul de *string*-uri, anume "tolerabil".

De acum, bineînţeles că putem închide programul (click pe X-ul din dreapta-sus al *Form*-ei), încât să schimbăm Proprietăţi ale Controalelor, să le reaşezăm, să adăugăm altele pe interfaţă şi să repornim şi să închidem iar şi iar programul acesta minunat.

Crainic

Vom crea un program care, în apropierea unei zile importante şi chiar în acea zi, afişează un Dialog (o *Form*-ă) pe care de asemenea este afişat un mesaj de atenţionare, de reamintire a ocaziei.

Pentru aceasta, schema sau "white paper"-ul programului este gândită-proiectată astfel, pornind desigur de la întrebarea "Ce anume vrem să facă programul?" Anume:

- trebuie să verificăm data curentă. În acest caz, ne interesează o zi anume dintr-o lună anume.
- dacă ziua curentă ***a computerului pe care rulează programul*** este atât din luna de interes, cât şi în apropierea zilei de interes sau dacă este chiar aceasta, atunci
- se afişează un mesaj de atenţionare/de reamintire despre ocazia respectivă

Până acum, ştim deja cum pe un Dialog se plasează Controale (butoane, etichete, casete de editare aa textului şamd):

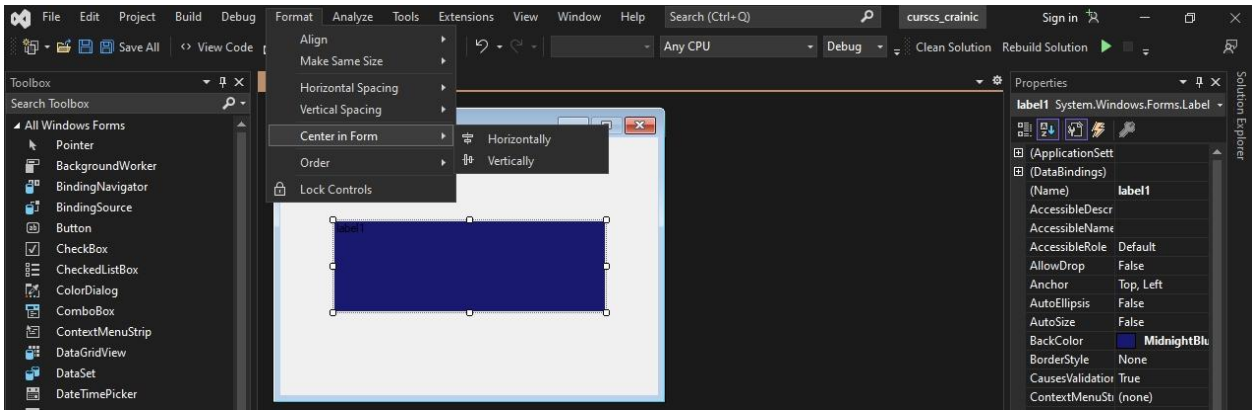
- fiecare Control se trage din Toolbox cu mouse-ul şi se plasează pe Dialog, sau
- în Toolbox, dublu-click pe Controlul care trebuie plasat pe Dialog

Astfel, pe Dialogul creat automat de catre Visual Studio plasăm aceste Controale:

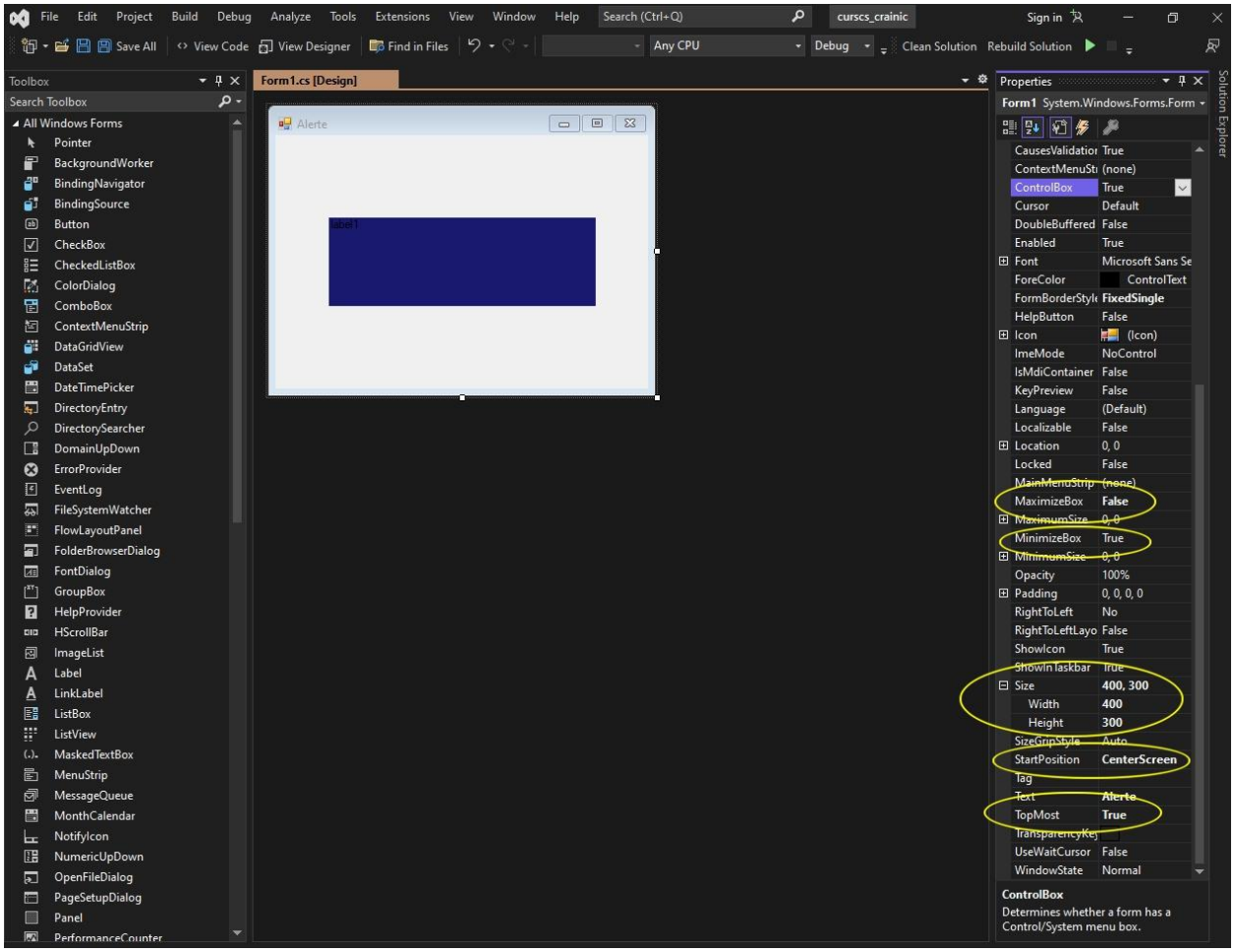
- un cronometru (Timer), "timer1"
- o etichetă unde va fi afişat un text (Label), "label1"

Aranjăm eticheta exact în mijlocul Dialogului:

- click pe label1
- din meniul Visual Studio selectăm "*Format*" şi opţiunile din imaginea următoare:



Stabilim câteva Proprietăţi ale Dialogului:
(denumirea lor este sugestivă, iar valorile posibile de asemenea)

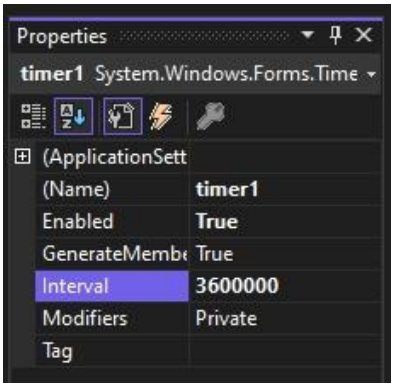


Se observă ce valori au fost alese:

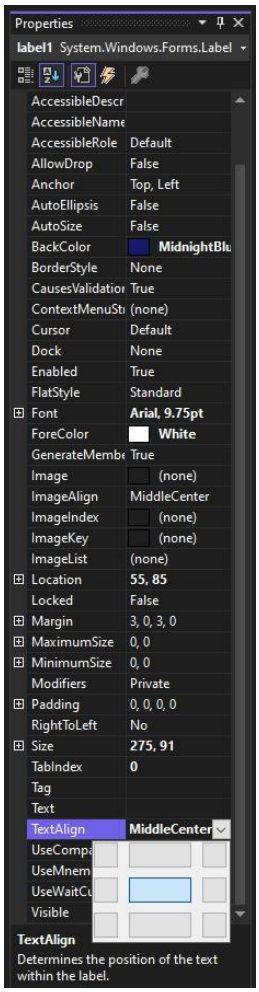
- *ControlBox* (True/False): specifică dacă în colţul din dreapta-sus vor fi afişate butoanele de Minimize-Maximize-Close ale acestui Dialog
- *MaximizeBox*: specifică dacă va fi afişat butonul de Maximizare a Dialogului.
  Această Proprietate este relevantă numai când *ControlBox* = True
  MinimizzeBox: se referă la butonul de Minimizare (pe "bara de jos" din *Windows*), cu aceeaşi menţiune ca pentru *MaximizeBox*
- *Size*: dimensiunea, în pixeli pe orizontală şi pixeli pe verticală, a acestui Dialog
- *StartPosition*: specifică poziţia Dialogului pe ecran (pe monitor, pe *desktop*) atunci când Dialogul va fi afişat (de fapt, oricând în timpul rulării programului, nu numai la pornirea acestuia)
- *Text*: titlul acestei *Form*-e, afişat desigur pe "bara de sus" a acestui Dialog. Anume, "Alerte"
- *TopMost* (True/False): specifică dacă Dialogul acesta va fi permanent deasupra altor Dialoguri vizbile în acelaşi timp şi eventual aparţinând altor programe decât acesta

Stabilim Proprietăţile de interes ale cronometrului:

- *Enabled*, activ, pornit, adică *Tick*-ăie – încă de la pornirea programului (la afişarea singurei *Form*-e a programului): True
- *Interval*-ul la care cronometrul va *Tick*-ăi: îl stabilim la 1 oră (suficient pentru a monitoriza ziua) 60 minute x 60 secunde = 3600 secunde = 3600 x 1000 milisecunde = 3 600 000 milisecunde



Stabilim câteva Proprietăţi de interes ale etichetei: *BackColor*, *Font*, *TextAlign* şi neapărat *Visible* True:



Acum scriem codul programului.

Vrem ca deja la pornirea programului să fie aflate ziua şi luna curente.

Aceasta se va întâmpla la Evenimentul *Load* al Dialogului *Form1*.
Acesta Eveniment este cel implicit al unui Dialog, ceea ce însemnă că metoda asociată poate fi creată automat de către Visual Studio, solicitând aceasta prin dublu-click pe *Form1*.
Rezultatul este, ca pentru orice alt Control:

- generarea auomată a metodei asociate Evenimentului implicit al acelui Conrol şi
- trimiterea noastră în modul de scriere a codului. Ceea ce şi facem, scriind această instrucţiune de executat la încărcarea (*Load*)/afişarea Dialogului *Form1*:

```
private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "[se asteapta apropierea unei zile insemnate]";
}
```

Am scris că deja la afişarea Dialogului *Form1*, pe eticheta "label1" să fie afişat un text informativ:

label1.Text = "îse asteapta apropierea unei zile insemnateş";

Acum scriem codul care face programul într-adevăr util, anume scriem instrucţiunile de executat la fiecare *Tick*-ăit (1 oră, 3600000 milisecunde) al cronometrului "*timer1*":

- înapoi în mediul de proiectare a interfeţei: tastele Shift+F7, păsate simultan. Înapoi la cod ne putem întoarce cu tasta F7.
- dublu-click pe "timer1" pentru a solicita de la Visual Studio să genereze metoda Evenimentului *Tick*, cel implicit al lui "*timer1*"
- în această metodă scriem litania din continuare, fiindcă, ei bine, n-avem încotro decât să verificăm ce anume zi şi lună sunt cele ale computerului pe care rulează programul, încât să fie afişat mesajul adecvat strict acelei date (*DateTime*):

codul din aceste tutoriale
poate fi copiat de aici (Copy)
şi plasat (Paste) în Visual Studio

```
private void timer1_Tick(object sender, EventArgs e)
{
	switch (DateTime.Now.Month)
	{
		//aflam ce luna este acum, la Tick-aitul cronometrului
		case 1:
			//ianuarie
			if (DateTime.Now.Day == 1)
			{
				//1 ianuarie
				label1.Text = "PRIMA ZI A ANULUI\r\n\r\nLA MULTI ANI!";
			}
			break;
		case 2:
			//februarie
			if (DateTime.Now.Day == 12)
			{
				//12 februarie
				label1.Text = "Poimaine este Valentine's Day";
			}
			else if (DateTime.Now.Day == 13)
			{
				//12 februarie
				label1.Text = "Maine este Valentine's Day";
			}
			break;
		case 3:
			//martie
			if (DateTime.Now.Day == 1)
			{
				//1 martie
				label1.Text = "PRIMA ZI DE PRIMAVARA";
			}
			else if (DateTime.Now.Day == 8)
			{
				//8 martie
				label1.Text = @"ASTAZI E 8 MARTIE,
						ZIUA FEMEII\r\n\r\nFELICITARI!";
				//textul (string-ul) de mai sus este precedat de caracterul @
				//inainte de ghilimelele de [nceput "
				//care permite scrierea (aici, in codul programului)
				//a unui text intins pe mai multe randuri
				//dupa virgula urmeaza un text care desi e pe alta linie in instructiune,
				//va fi afisat pe aceeasi linie/rand cu textul dinaintea virgulei
				//
				//indicatorul de LINIE/RAND NOU
				//este secventa \r\n (echivalentul apasarii tastei Enter= rand nou)
				//o linie goala este reprezentata de secventa \r\n\r\n
				//(adica, doua apasari pe tasta Enter)
```

```
                        //
                        //
                        //
                        //
                        //textul din comentarii precedate de // poate fi scris pe mai multe
                        randuri:
                        //fie precedata fiind fiecare linie de //
                        //fie incepand litania comentariului cu caracterele /* si inchizand-o
                        cu */
                }
                        break;
                case 8:
                        //august
                        if (DateTime.Now.Day == 6)
                        {
                                //6 august
                                label1.Text = "6 august 1945\r\nMass-murder at
Hiroshima";
                        }
                        else if (DateTime.Now.Day == 9)
                        {
                                //9 august
                                label1.Text = "9 august 1945\r\nMass-murder at
Nagasaki";
                        }
                        break;
                case 12:
                        if (DateTime.Now.Day == 31)
                        {
                                //31 decembrie
                                label1.Text = @"ULTIMA ZI A ANULUI\r\n\r\n
                                        LA NOAPTE, PARTY!";
                        }
                        break;
                //se pot insera/sterge blocuri case... break;
                //depinzand de care luna anume si de care zi(le) din acea luna
                //suntem interesati
                default:
                        label1.Text = "[se asteapta apropierea unei zile insemnate]"
+
                                "\r\n\r\n"+
                                "Astazi este:\r\n" +
                                DateTime.Now.ToString("dd          MMMM
yyyy");
                        break;
                }
        }
```

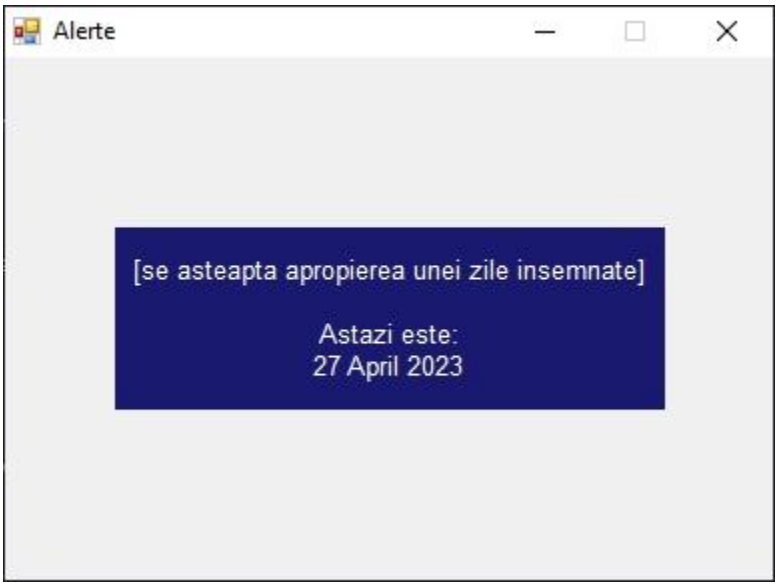
În codul de mai sus observăm instrucţiunea pentru cazul ***default*** –
adică, instrucţiunea (sau instrucţiunile) de executat la oricare *Tick*-ăit al lui "timer1"
când ziua şi luna ***nu sunt prinse de variantele case***:
este afişat un text informativ despre situaţia/data curentă **la momentul *DateTime* al afişării *Form1***.
Plasăm această instrucţiune şi în metoda Evenimentrului *Load* al *Form1*, pentru ca, atunci când programul porneşte odată cu afişarea acestui Dialog, pe "label1" să fie afişat un text – este posibil ca data curentă să nu fie încadrată la un ***case***:

```
private void Form1_Load(object sender, EventArgs e)
{
        label1.Text = "[se asteapta apropierea unei zile insemnate]" +
                      "\r\n\r\n" +
                      "Astazi este:\r\n" +
        DateTime.Now.ToString("dd MMMM yyyy");
}
```

Desigur, în Form_Load putem pune întreg codul din metoda timer1_Tick, de la switch încolo.
Efectul va fi acelaşi ca la fiecare *Tick*-***ăit al cronometrului "****timer1****": la pornirea programului se va verifica data şi dacă este cazul va fi afişat mesajul adecvat datei curente.***

Primul Tick-ăit al cronometrului este după o (1) oră
de la pornirea programului –
când porneşte şi cronometrul

Acum pornim programul – apăsăm tasta F5 - şi ca de obicei ne aşteptăm să funcţioneze.

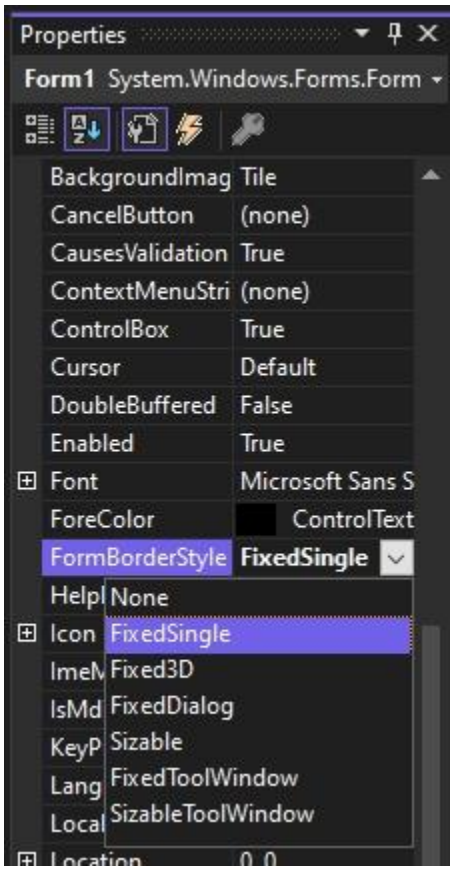


Putem lăsa programul să ruleze (să "meargă") fără a-l vedea pe *desktop* – apăsând pe butonul de Minimizare din dreapta-sus, anume cel din grupul de butoane indicat de Proprietatea *ControlBox* pe care am modificat-o mai devreme.
Astfel, Dialogul *Form1* va fi minimizat – plasat pe "bara de jos" a *desktop*-ului.

Încă o Proprietate de interes în acest caz a Dialogului – utilă inclusiv când vrem ca un Dialog să poată fi numai Minimizat sau numai Maximizat, este:

FormBorderStyle

Pentru a modifica Proprietatea aceasta:
- închidem programul (click pe X în ControlBox-ul din dreapta-sus al *Form1*
- înapoi în modul de proiectare a *Form1*
- click-stânga pe *Form1* (selectăm acest Control)
- apăsăm tasta F4 pentru a fi afişată secţiunea de Proprietăţi ale *Form1*
- schimbăm după plac Proprietatea *FormBorderStyle*, pornim iarăşi (F5) programul şi tot aşa până ne dăm seama de aspectul şi de comportarea acestui Dialog în acord cu fiecare valoare a acestei Proprietăţi:



În acest caz, după Minimizarea *Form1* (a unicului Dialog al programului) pe "bara de jos", pentru ca programul să nu deranjeze fiind afişat *permanent* (fiindcă pentru *Form1* am stabilit deja Proprietatea *TopMost*= True), observăm că programul nostru se comportă ca oricare altul – anume, titlul lui *Form1*:



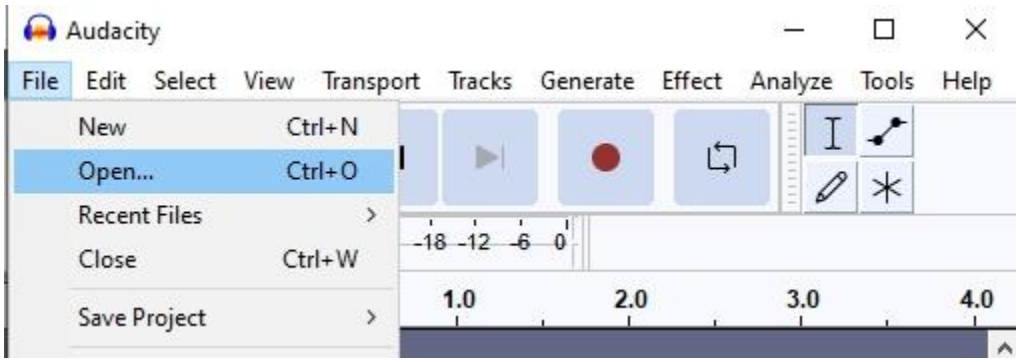
Meniuri

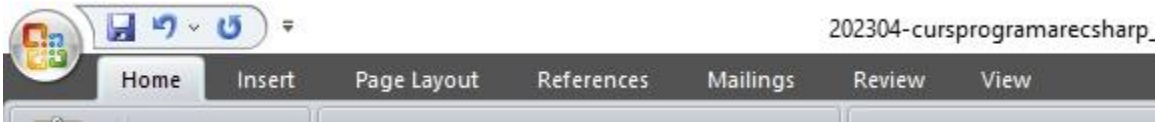
Vom face un program în care sunt folosite *meniuri* afişate pe un Dialog (pe o *Form*-ă).

Un *meniu* este un Control care, ca toate Controalele cu contur delimitând o regiune pe Dialog, poate fi plasat pe un Dialog încât utilizatorul să interacţioneze cu el fie prin click pe butoanele mouse-ului, fie prin tastatură.

Meniul (principal al) Dialogului poate consta dintr-unul sau mai multe regiuni ilustrate de secvenţe de text (cel puţin un cuvânt) de altfel familiare oricărui utilizator al programelor care pot rula într-un sistem de operare – de obicei (şi de fapt), pentru înlesnirea interacţiunii utilizatorului cu programul, acele *intrări de/în meniu* (*menu entries*):

- sunt afişate imediat sub bara de titlu a programului
- *sub/în* fiecare intrare "principală" (de asemenea) a acestui meniu principal pot exista comenzi distincte (grupate logic după tipul operaţiunilor sugerate de textul intrării-părinte de meniu)
- meniul principal *poate* avea aceste aspecte:





Se observă că meniurile din cele două imagini precedente sunt diferite, între cele două programe:

- denumirea intrărilor de meniu - de obicei, fiecare program are propriul meniu
- numărul acelora– cu aceeaşi menţiune
- culorile, *Font*-ul şi formele – la fel, cu aceeaşi menţiune

Aspectul, denumirea şi numărul intrărilor meniului principal şi de altfel ale oricărui alt tip de meniu afişat pe un Dialog nu contează decât pentru

- facilitarea interacţiunii utilizatorului cu acel Dialog/cu programul
- aspectul general al Dialogului

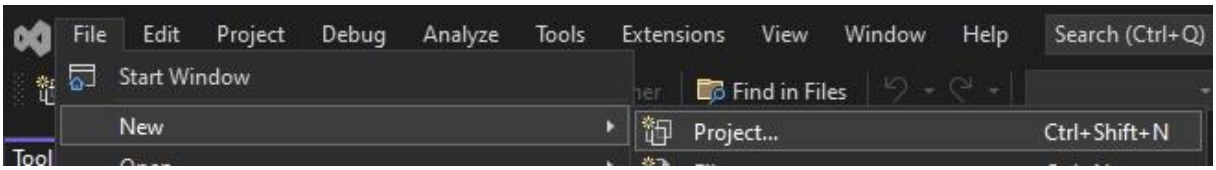
Funcţionalitatea oferită de structura unui meniu este cea care contează.
Astfel, un program în care vor fi manipulate (frecvent) fişiere trebuie să aibă o intrare de meniu numită "Fişiere", "Files" sau denumită similar – unde comenzile care lansează operaţiuni distincte cu fişiere pot fi înşirate ca în prima imagine din cele două precedente: File -> New, Open, Recent Files, Close şamd.

Deci se reţine – un programator serios nu pune o comandă referitoare la deschiderea unui fişier sub intrarea principală denumită "Font" sau "Numere de telefon", ci grupează comenzile referitoare la operaţiuni cu acelaşi tip de entitate sub/într-o intrare de meniu denumită astfel încât să sugereze ce anume se află acolo.

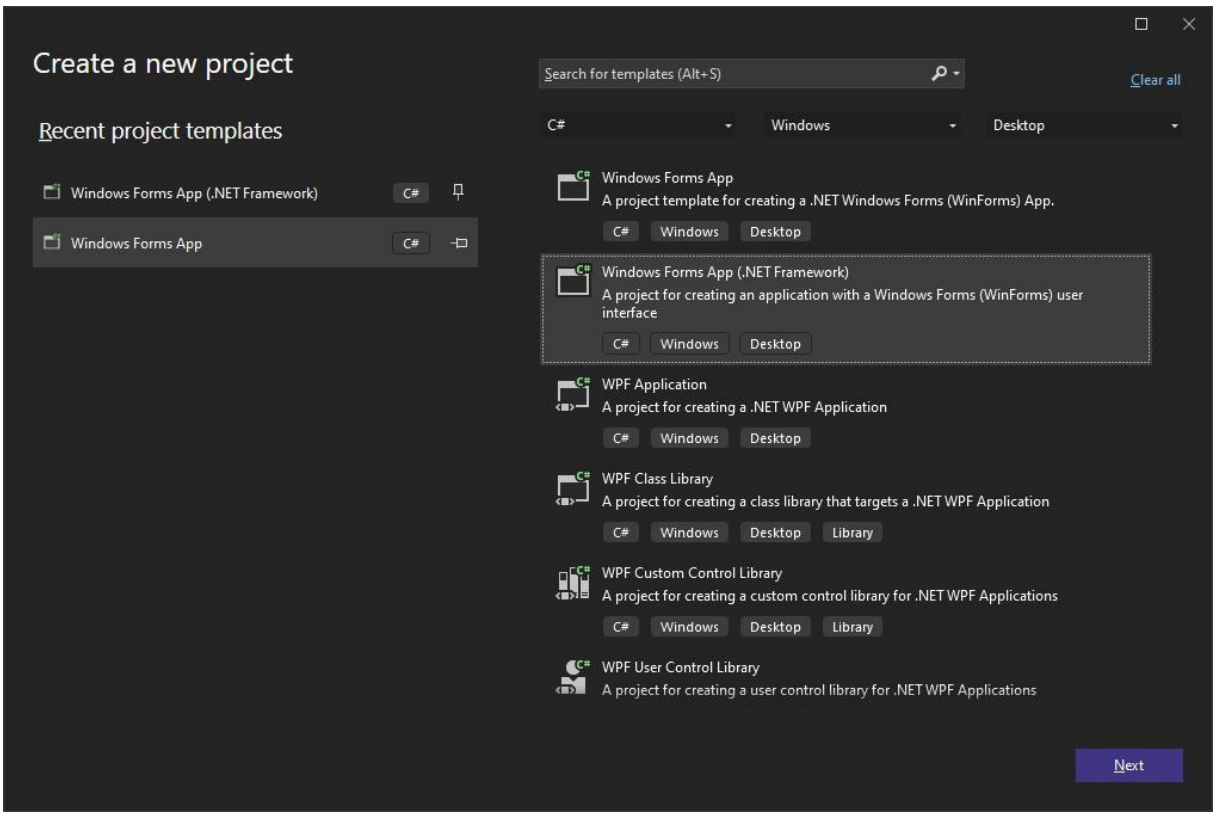
Acestea fiind clare, să începem a construi programul nostru cu meniuri.

Pornim Visual Studio, apoi:

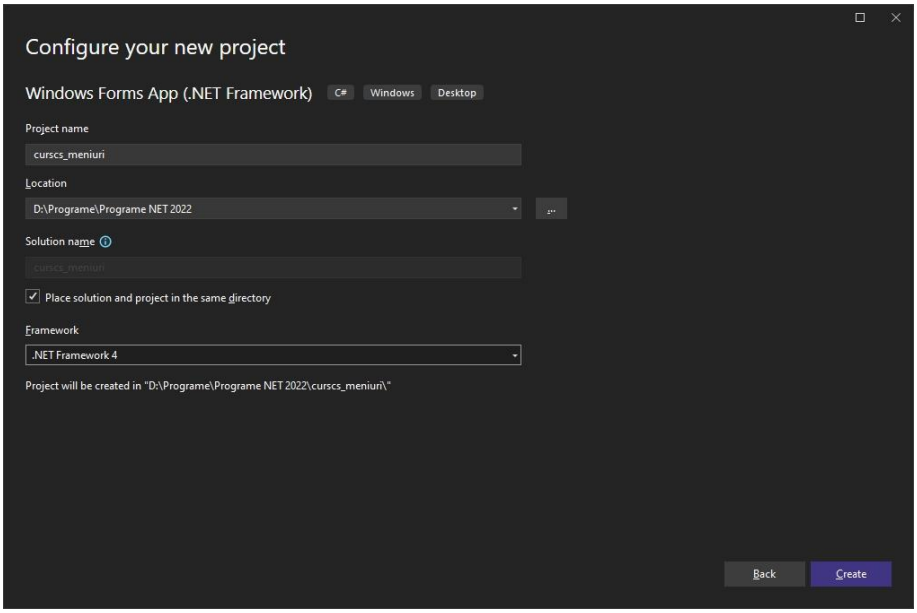
- alegem din meniul, desigur, principal intrarea *File,*
- de sub/din aceasta comanda *New...,*
- apoi din optiunile derulate din aceasta alegem *Project...*



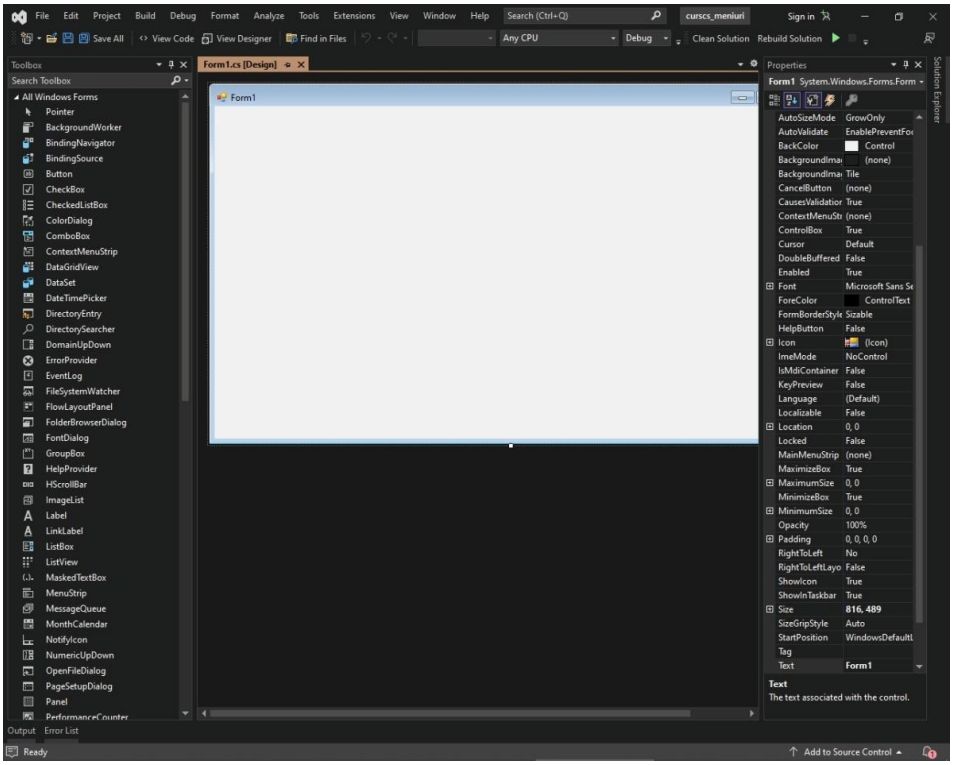
Mai departe, la fel ca în exemplele de programe dinaintea acestuia: vrem să creăm un program (o aplicaţie de tipul "Windows Forms App (.NET Framework), cu interfaţă (cu Dialoguri/*Form*-e):



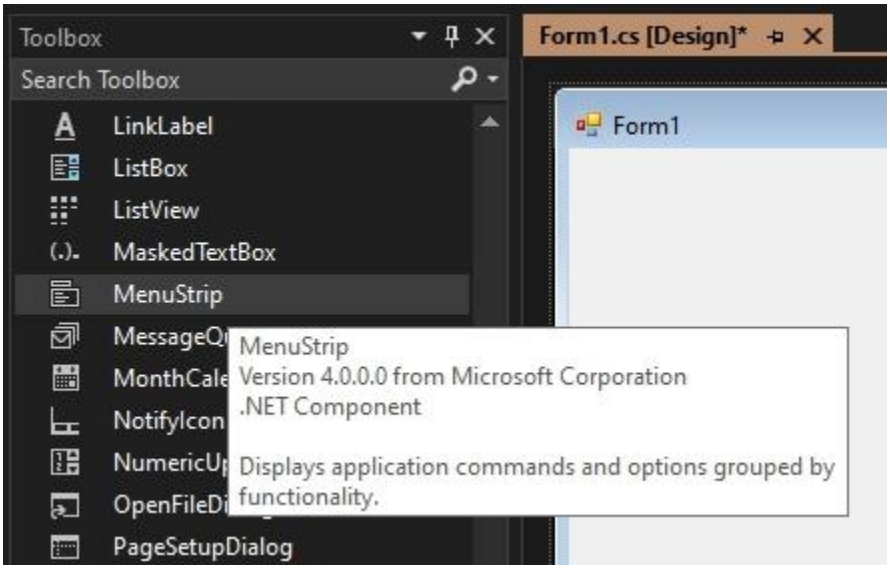
Specificăm denumirea şi locaţia programului şi versiunea platformei .NET pe care o vom aplica atât în cod, cât şi la proiectarea Dialogului cu Controale disponibile în acea versiune de .NET, de exemplu ca în figura următoare şi de altfel cum se ştie deja:



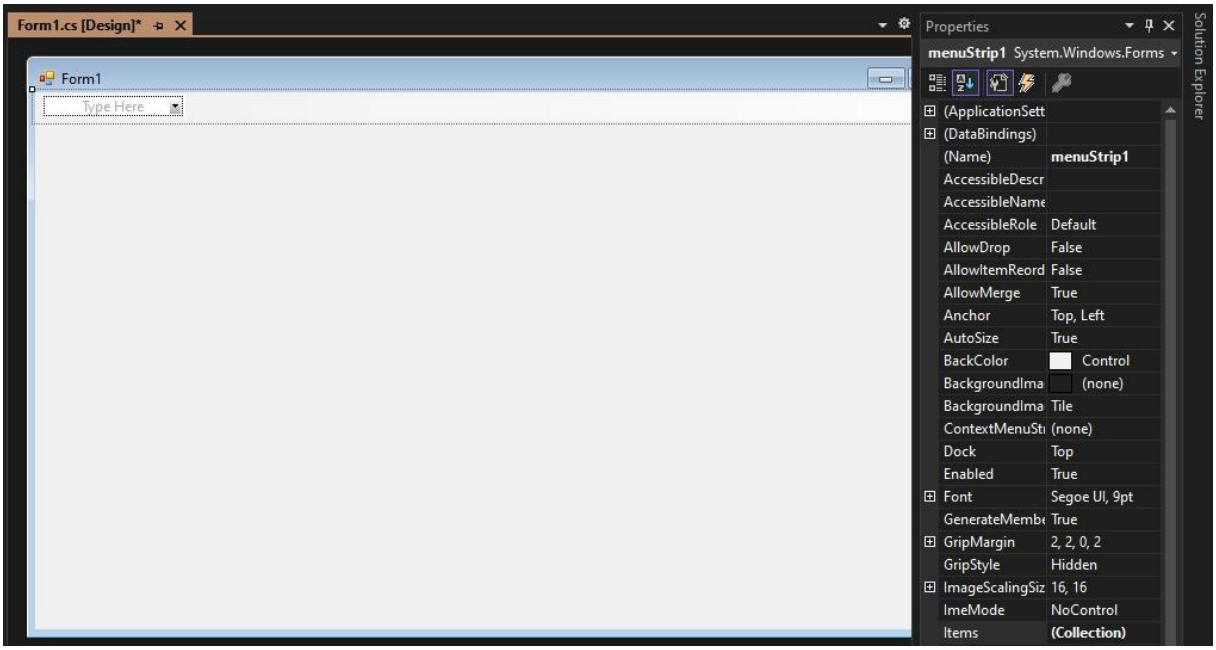
Apăsăm butonul "Create" şi obţinem, ca de obicei, un Dialog generat automat pe care putem plasa Controale şi căruia să-i asociem cod:



Pe Dialog, din Toolbox tragem (sau dublu-click pe) un Control denumit "MenuStrip":
Acesta este un *meniu principal* pentru un Dialog (desigur, aici Dialogul fiind *Form1*). Pot fi adăugate mai multe astfel de meniuri principale, dar de obicei nu e necesar decât unul, uneori chiar şi unul singur fiind inutil (depinzând de numărul comenzilor conţinute şi de priceperea/dispoziţia/simţul estetic ale programatorului):

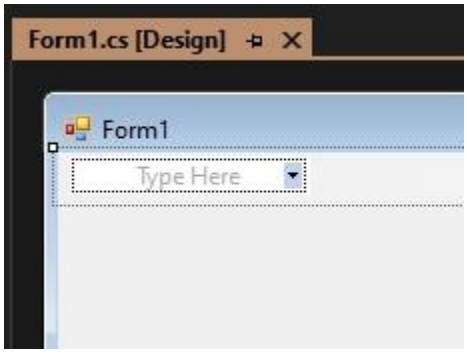


Vedem cum meniul "menuStrip1" a fost plasat imediat sub bara de titlu a Dialogului şi cum secţiunea de Proprietăţi ne oferă posibilitatea de a stabili diverse Proprietăţi a meniului, de exemplu culorile *BackColor*, *Font*-ul şi chiar *Dock* – care specifică unde anume va fi afişat meniul: sus, jos, în stânga, în dreapta sau în mijlocul lui *Form1*.



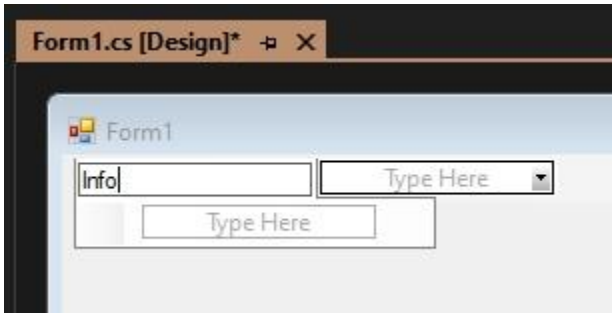
Începem să adăugăm intrări de meniu:

Pe acest meniu principal, "menuStrip1" este deja indicat locul (*placeholder*-ul) unde se poate scrie denumirea unei intrări de meniu:
Click-stânga în caseta de editare a textului din imaginea urmatoare:



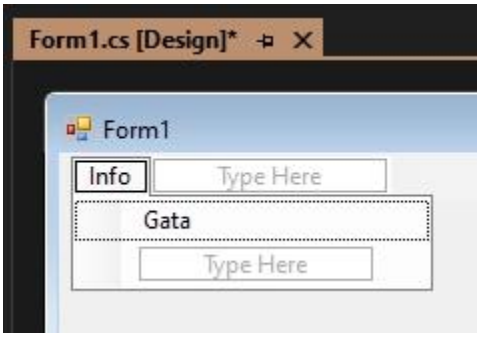
Acolo scriem "Info":
Deci astfel vom crea o intrare propriu-zisă de meniu, adică:

- aceea are o denumire
- utilizatorul va putea interacţiona cu acea intrare de meniu fie prin click cu mouse-ul, fie din tastatură (de obicei, ţinând apăsată tasta Alt şi apăsând tasta primei litere a intrării de meniu).



Mai vedem cum la dreapta şi dedesubtul textului acestei intrări principale de meniu tocmai scris(e) se află *placeholder*-e pentru comenzi din grupul asociat acestei intrări de meniu.

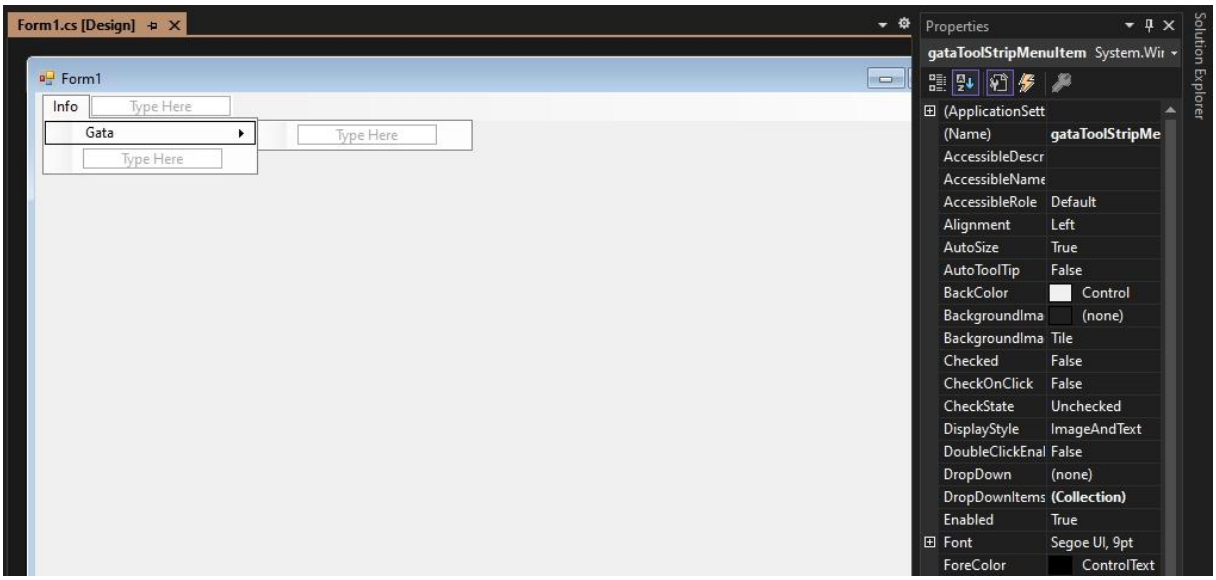
Astfel, în caseta de sub "Info" scriem textul (comanda): "Gata" (de exemplu); aceasta indică o comandă pentru închiderea programului – cod pe care îl vom scrie în curând:



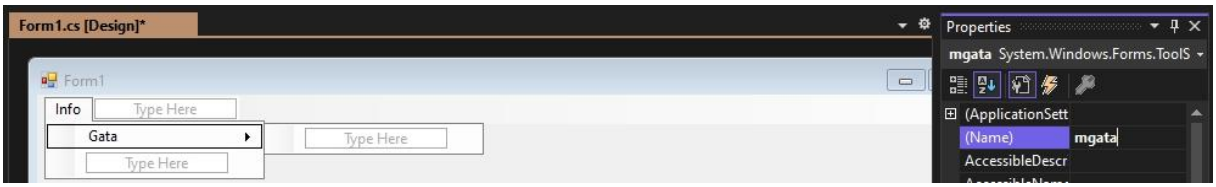
În acest exemplu, "Gata" nu prea are ce căuta sub intrarea de meniu "Info", ci mai degrabă sub "Program", sau "Ajutor" .
Dar ne mulţumim cu asta fiindcă:

Putem schimba oricând textul intrărilor de meniu
Funcţionalitatea lor contează, cea indicată în metoda asociată Evenimentuui Click ale fiecăreia

La fel ca pentru alte Controale, pentru comanda "Gata" pot fi stabilite Proprietăţi:



Proprietatea ("*Name*"), denumirea implicită a acestui Control este "gataToolStripMenuItem" – totuşi, prea lungă pentru a menţine codul cât de cât scurt, aşa că o schimbăm la "mgata" (sau similar, dar sugerând natura acestui Control: "meniu-gata", ca pentru orice variabilă denumită pentru a şti ce anume e cu ea):



Acum sciem codul asociat acestei comenzi, instrucţiunile, operaţiunile ce trebuie efectuate atunci când utilizatorul alege/acţionează această comandă/intrare secundară de meniu, "Gata":

Pentru aceasta, bănuind că Evenimentul implicit asociat unei comenzi/intrări (secundare) de meniu este Evenimentul *Click*, solicităm de la Visual Studio să genereze metoda acelui Eveniment, pentru ca mai departe să scriem acolo codul de executat după *Click* pe "Gata":

Deci, ca de obicei pentru generarea automată a metodei asociate Evenimentului *implicit* al unui Control: *dublu-click* ***pe textul*** *tocmai scris (pe "Gata")*.
Suntem trimişi imediat în modul de scriere a codului, unde scriem:

```
private void mgata_Click(object sender, EventArgs e)
{
        DialogResult dres = MessageBox.Show(
                "Confirmati inchiderea programului?",
                "Confirmare",
                MessageBoxButtons.YesNo,
                MessageBoxIcon.Question,
                MessageBoxDefaultButton.Button2);
        if (dres == DialogResult.Yes)
        {
                Application.Exit();
        }
}
```

Linie cu linie:

DialogResult dres = MessageBox.Show(
Am creat o variabilă de tip *DialogResult* a cărei valoare va fi aflată după afişarea şi închiderea unei casete de dialog de tipul "MessageBox" – aceasta fiind, probabil, cea mai familiară secvenţă din interacţiunea cu un program: ***MessageBox.Show****(un mesaj pentru utilizator);*
.Show este metoda care afişează o casetă *MessageBox*, unde utilizatorul va putea alege un răspuns.

"Confirmati inchiderea programului?"
Acesta este textul afişat în casetă. Poate fi, un text informativ, o întrebare, sau combinaţii de texte de diverse feluri pe care utilizatorul (ar) trebui(e) să le citească şi la care eventual să răspundă.
În acest caz, utilizatorul este întrebat ceva.

"Confirmare"
Acesta este titlul casetei de dialog MessageBox.

MessageBoxButtons.YesNo
Am specificat ce anume butoane se vor afla în caseta de dialog – anume, unele adecvate scopului: utilizatorul trebuie să răspundă la o întrebare, aşa că din opţiunile proprietăţii *MessageBoxButtons* se alege cea care afişează un buton pentru răspunsul afirmativ şi pe care scrie "Yes" şi un buton pentru răspunsul negativ.

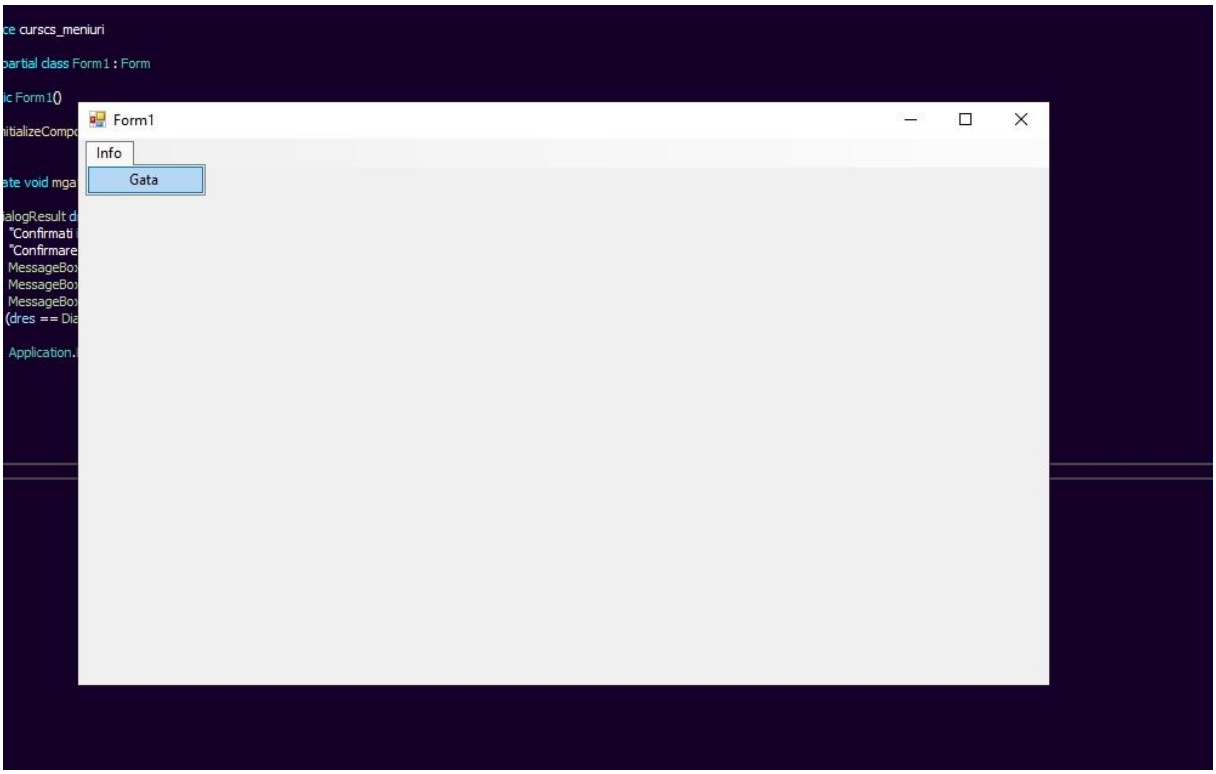
MessageBoxIcon.Question
Aceasta este pictograma de afişat pe caseta de dialog. La fel ca pentru butoanele din casetă, pot fi alese diverse valori ale acestei proprietăţi a casetei. Testaţi.

MessageBoxDefaultButton.Button2
Am specificat că în această caseta de dialog, la afişarea sa va fi selectat (ca focalizare/Focus) *implicit* butonul 2, anume "No".

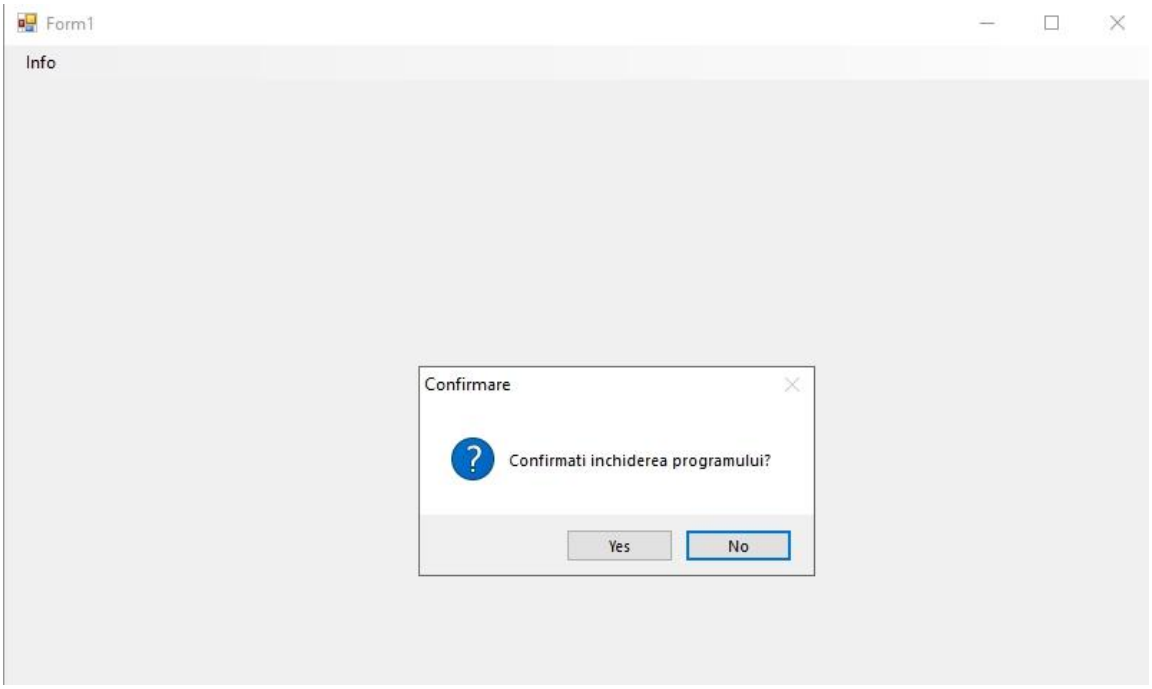
if (dres == DialogResult.Yes)
Dacă utilizatorul răspunde afirmativ la întrebare, atunci

Application.Exit();
Programul va fi închis.

Lansăm/pornim programul: tasta F5:



Observăm cum în caseta de dialog, la afişarea sa, este preselectat butonul "No" ("Button2" de mai sus):



Click pe "Yes" şi programul se va închide.

Accesarea unor părţi ale sistemului de operare Windows

Adineauri am prezentat cum se afişează o casetă de dialog *MessageBox* (în acest caz fiind vorba de un dialog în sensul primar al cuvântului:

O casetă de dialog MessageBox
solicită un răspuns de la utilizator

Interacţiunea utilizatorului cu programul
este întreruptă/blocată
până când utilizatorul închide caseta de dialog
apăsând unul din butoanele de pe aceasta –
adică, până când utilizatorul participă la dialogul început cu mesajul din MessageBox.

Până una-alta, să mai aflăm câte ceva despre instrucţiunea *MessageBox.Show*, pe lângă aceea că este utilizată foarte frecvent:

- foloseşte la solicitarea unui răspuns de la utilizator, de exemplu Da/Nu, OK sau Anulare
- pot fi afişate mai multe *MessageBox*-uri; totuşi, în acest caz se presupune că programatorul chiar a avut nevoie de răspunsurile utilizatorului care mai departe vor folosi mersului programului
- utilizatorul NU poate interacţiona prin mouse şi tastatură cu interfaţa programul până când închide ultimul *MessageBox* afişat în program
- cea mai simplă instrucţiune pentru afişarea unei *MessageBox* are sintaxa:
  MessageBox.Show(mesaj); //unde, desigur, "mesaj" este un *string* (un text)
  Această instrucţiune afişează *mesaj*-ul sub care se află un buton marcat "OK"; ca orice alt fel de *MessageBox*, aceasta obligă utillizatorul să o închidă, încât acesta să poată interacţiona în continuare cu programul
- nimic din ceea ce se află într-o *MessageBox* nu poate fi schimbat în timp ce aceasta este afişată
- de exemplu, un *Timer* nu poate trimite periodic (la Evenimentul său *Tick*) data-ora curente
- *MessageBox.Show poate avea orice sintaxă validă -*
- *de exemplu, poate afişa o întrebare şi pictograma cu semnul exclamării şi un singur buton (OK) dar utilizatorul trebuie să o închidă acţionând unul din butoane*

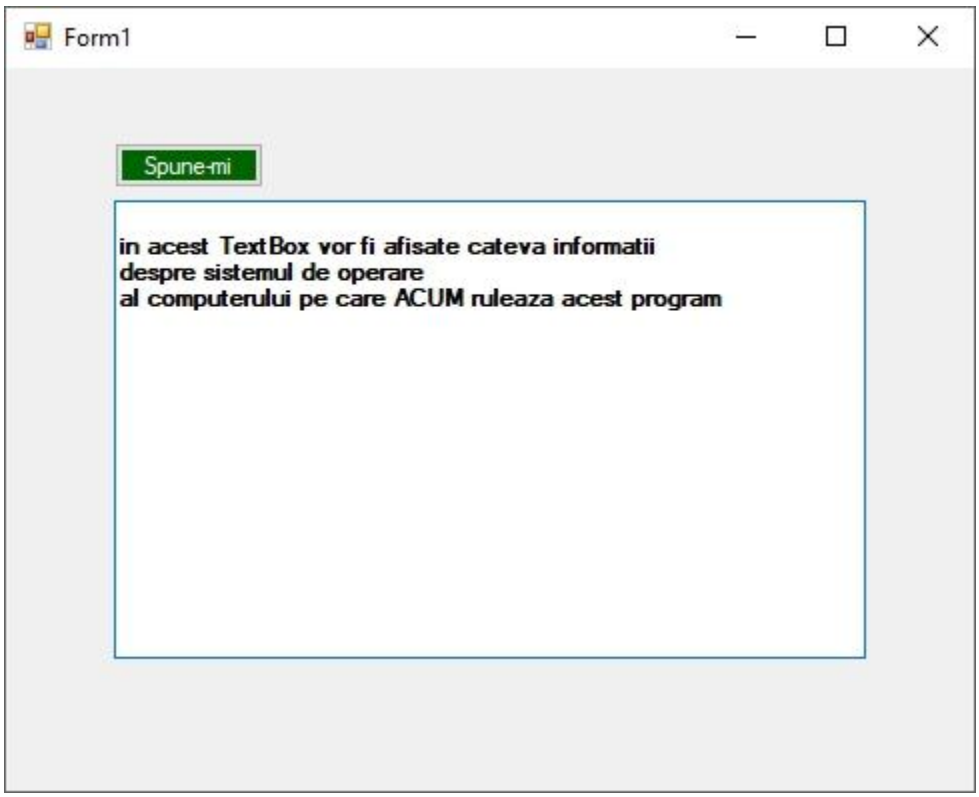
Acum, putem presupune foarte bine că un program care ruleaz ă în sistemul de operare (*Windows*, *unix* sau altul) are cunoştinţă de unele informaţii despre acel sistem şi poate interacţiona cu unele porţiuni ale acestuia:
De exemplu, un program poate fi instruit să folosească locaţia (directorul/folderul) de unde a pornit, adică locul unde se află fişierul executabil, "numeleprogramului.exe".
Reciproca este adevărată:
De exemplu, un program nu poate porni dacă sistemul de operare nu are suficientă memorie (RAM) pentru a suporta funcţionalitatea programului.

Să aflăm cum accesăm din programuil nostru:

- câteva informaţii despre sistemul de operare *Windows*
- locul de unde a pornit programul – această informaţie ne trebuie dacă vrem să pornim programul din afara Visual Studio (unde îl pornim cu tasta F5) – adică, dublu-click pe fişierul .exe al său listat de exemplu în "Windows Explorer" ("My Computer")

Începem un program nou: ca de obicei: pornim Visual Studio, *File -> New... Project...*

Pe *Form1* tragem un *Button* (denumit implicit "button1") şi un *TextBox* (denumit implicit "textBox1"). Stabilim câteva Proprietăţi ale acestora, încât interfaţa să nu fie inexpresivă; de exemplu, *BackColor*, *ForeColor* şi *Font*, apoi aliniem cumva Controalele şi centrăm *TextBoxul* pe Dialog (din meniul *Format*, opţiunea *Center in Form* şamd.)

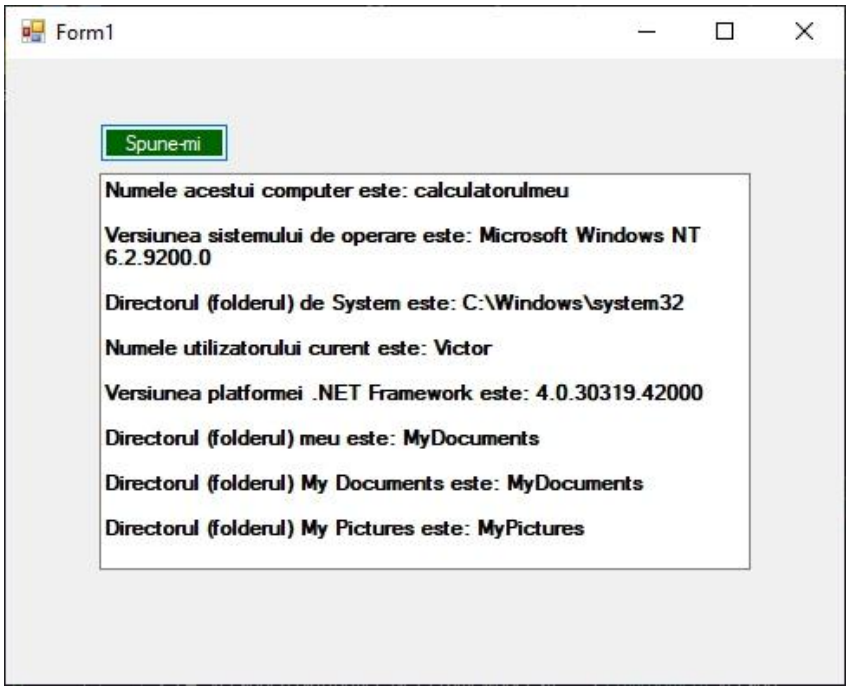


Dublu-click pe *button1*, ceea ce ne trimite în modul de scriere a codului, în metoda asociată Evenimentului *Click* al acestui *Button* – metodă unde scriem astfel:

```
private void button1_Click(object sender, EventArgs e)
{
        //initializam o variabila de tip string
        //la care vom adauga stringuri cu informatii
        //cu operatorul += (care la valoarea existenta /aduna/adauga/ valoarea
        //de dupa semnul egal
        //si pe care o vom afisa in textBox1
        //
        string s = "";
        //
        s += "Numele acestui computer este: " +
                Environment.MachineName;
        s += "\r\n\r\n";    //o linie noua la sfarsitul textului precedent
        //urmata de o linie noua => o linie goala
```

```
            s += "Versiunea sistemului de operare este: " +
                    Environment.OSVersion;
            s += "\r\n\r\n";    //inca un rand gol
            s += "Directorul (folderul) de System este: " +
                    Environment.SystemDirectory;
            s += "\r\n\r\n";
            s += "Numele utilizatorului curent este: " +
                    Environment.UserName;
            s += "\r\n\r\n";
            s += "Versiunea platformei .NET Framework este: " +
                    Environment.Version;
            s += "\r\n\r\n";
            s += "Directorul (folderul) meu este: " +
                    Environment.SpecialFolder.Personal;
            s += "\r\n\r\n";
            s += "Directorul (folderul) My Documents este: " +
                            Environment.SpecialFolder.MyDocuments;
            s += "\r\n\r\n";
            s += "Directorul (folderul) My Pictures este: "+
                            Environment.SpecialFolder.MyPictures;
            //
            //
            textBox1.Text = s;
}
```

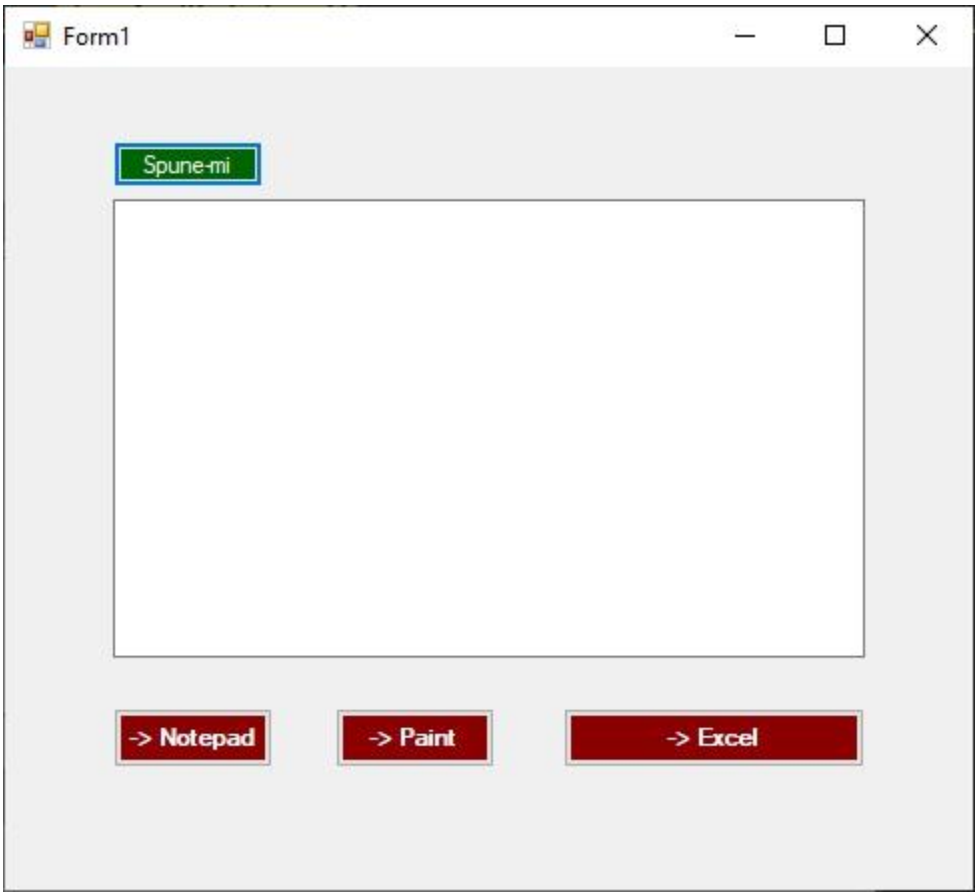
Pornim programul: apăsăm tasta F5: rezultatul:



Acum din programul nostru să pornim ***alte programe***:

Să redimensionăm Dialogul *Form1* încât să putem plasa câteva butoane: fiecare ne va ajuta să pornim un anumit program:
Vrem ca din program să pornim:

- Notepad
- Paint
- Excel



Se observă că butoanele tocmai adăugate pe *Form1* arată oarecum la fel:

- *BackColor* identice
- *ForeColor* identice

Aceasta fiindcă un Control poate fi duplicat pe interfață, încât să nu repetăm stabilirea Proprietăţilor care trebuie să fie identice cu cele ale Controlului original:

- click-stânga pe Control (îl selectăm)
- Copy (tastele Ctrl+C)
- Paste (tastele Ctrl+V) pe Dialogul unde vrem să plasăm acea copie a Controlului

Duplicarea unui Control
nu implică duplicarea Evenimentelor sale
Fiecare copie va avea propriile Evenimente la care să răspundă

Să scriem codul corespunzător fiecărui buton:
Dublu-click pe butonul marcat "Notepad":
Trimişi fiind în modul de scriere a codului, în metoda Evenimentului *Click* a acestui buton scriem:

```
try
{
        System.Diagnostics.Process.Start("notepad.exe");
}
catch (Exception ex)
{
        MessageBox.Show("Eroare:\r\n\r\n" + ex.ToString());
}
```

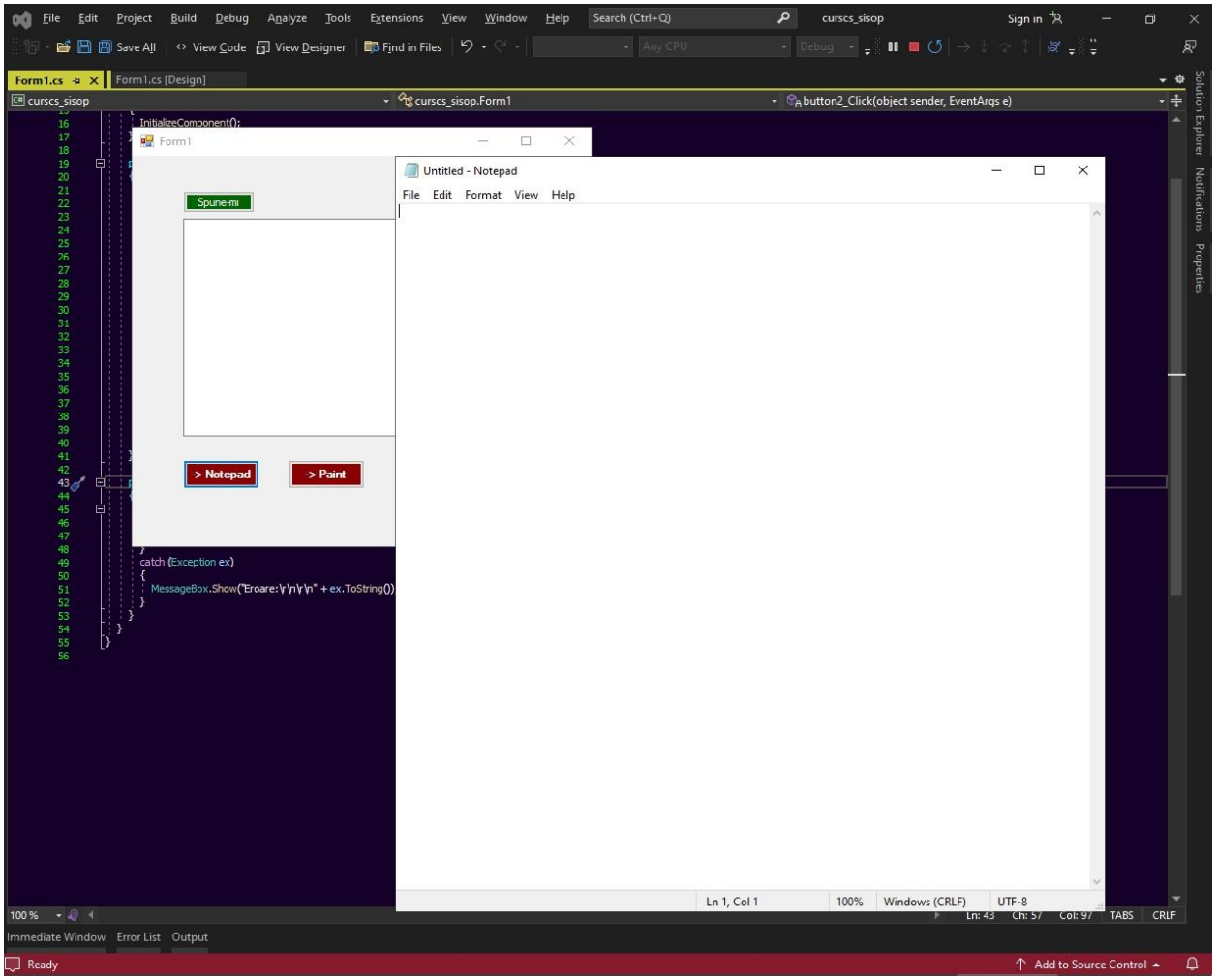
Linie cu linie:

System.Diagnostics.Process.Start("notepad.exe");
Din spaţiul de nume ***System.Diagnostics*** am ales obiectul "***Process***" a cărui metodă "***Start***" primeşte ca parametru necesar locaţia şi numele fişierului (executabil sau de alt fel, de exemplu unul cu extensia ".pdf" sau ".jpg") care trebuie pornit sau deschis cu programul asociat (de exemplu, un fişier-imagine cu extensia .jpg poate fi deschis (afişat pe ecran) cu programul asociat implicit "mspaint.exe").
Se observă cum *am specificat numai numele fişierului executabil*: "notepad.exe":
Aceasta, fiindcă fişierul solicitat este căutat în sistemul de operare mai întâi în aşa-numitele directoare de sistem ("Windows", "System32", "Program Files" ş. a.) ale Windows şi în directorul de unde porneşte programul care solicită acel fişier.
Deci, pentru Notepad care este instalat odată cu Windows, nu este nevoie de specificarea întregii căi până la el, de exemplu "C:\Windows\notepad.exe" – deşi se poate scrie şi aşa, pentru siguranţă.

catch (Exception ex)
În cazul când fişierul solicitat nu este găsit, *fie şi dacă am specificat calea către el*, utilizatorului îi va fi prezentat un mesaj conţinând detalii despre eroarea identificată prin variabila *ex*:

MessageBox.Show("Eroare:\r\n\r\n" + ex.ToString());
Mesajul pentru utilizator constă din textul "Eroare:" urmat de o linie goală (echivalentul a 2 *Enter*-uri: "ţrţnţrţn") şi de transformarea la *string* a erorii *ex*.

Pornim iarăşi programul (tasta F5) şi apăsăm butonul "Notepad":



Procedăm similar cu butonul "Paint":

În acest exemplu, programul "mspaint.exe" a fost instalat odată cu Windows în directorul:

"C:\Windows\System32"

Copiem această cale din Windows Explorer şi scriem în codul programului nostru:

```
try
{
        System.Diagnostics.Process.Start("C:\\Windows\\System32\\mspaint.exe");
}
catch (Exception ex)
{
        MessageBox.Show("Eroare:\r\n\r\n" + ex.ToString());
}
```

Observăm două slash-uri directe succesive care într-un *string* desigur cuprins între ghilimele denotă un sigur caracter slash direct – în acest caz, este indicatorul de nivel al dircetoarelor

Procedăm similar cu butonul "Excel":
Se presupune că programul "Excel" este instalat pe computerul unde rulează programul nostru.

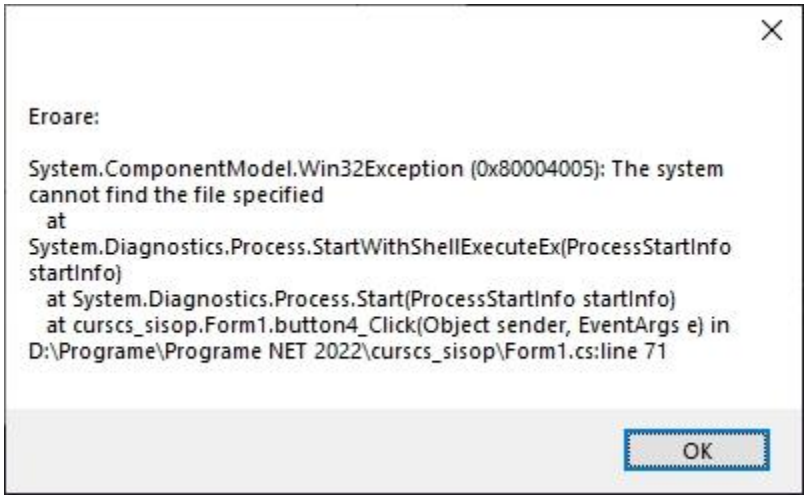
În acest exemplu, programul "excel.exe" se află în directorul:

"C:\Program Files (x86)\Microsoft Office\Office12"

Copiem această cale din Windows Explorer şi scriem în codul programului nostru, înlocuind fiecare slash cu două astfel de caractere:

```
try
{
        System.Diagnostics.Process.Start(
                "C:\\Program Files (x86)\\Microsoft Office\\Office12\\excel.exe");
}
catch (Exception ex)
{
        MessageBox.Show("Eroare:\r\n\r\n" + ex.ToString());
}
```

Dacă fişierul solicitat nu este găsit, mesajul de eroare arată astfel:



Testaţi cu diverse programe executabile (cu extensia .exe) şi cu fişiere-imagine (cu extensia .jpg sau .gif). Nu se strică nimic, atât timp cât nu ştergeţi si nu redenumiţi fişiere despre care nu ştiţi ce rost au.

Operaţiuni realizate la anumite intervale

Vom construi un program a cărui funcţionalitate depinde de un *Timer*:
- la fiecare *Tick*-ăit al *Timer*-ului se va executa o operaţiune

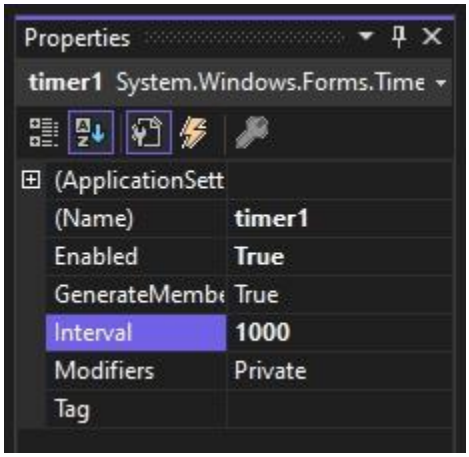
Anume:
- la pornirea programului va fi începută numărătoarea inversă de la, în acest exemplu, 60. Această valoare este numărul secundelor până când programul va îndeplini o anumită procedură.
- la fiecare *Tick*-ăit al *Timer*-ului va fi afişat numărul secundelor rămase până la iniţierea acelei proceduri
- la "expirarea timpului" va fi iniţiată acea procedură

Să începem:

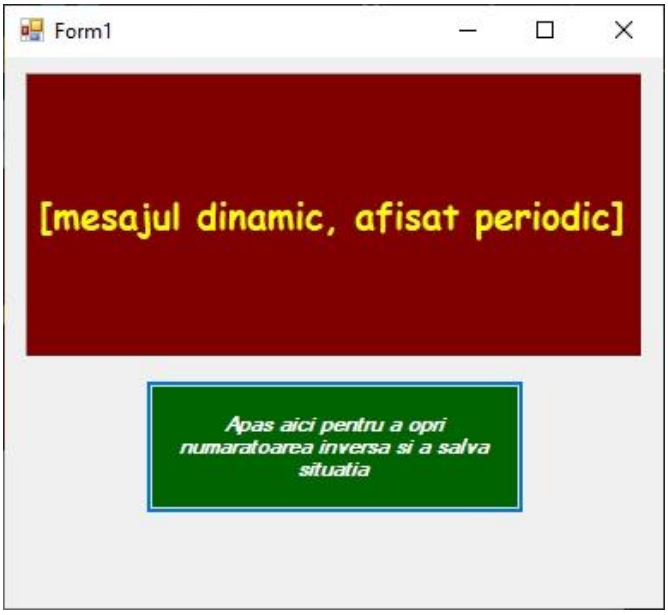
Ca de obicei, pornim Visual Studio, *File -> New... Project...*

Pe Form1 plasăm un *Timer*, un *Button* şi o *Label*.
Denumirile acestota le vom lăsa "timer1", "button1" şi "label1".

Întrucât ştim ce anume vom scrie în "label1", o redimensionăm (*AutoSize*: False şi apoi tragem de colţul său din dreapta-jos), îi stabilim Proprietăţile *BackColor* şi *ForeColor* la culori de asemenea adecvate scopului şi o plasăm pe *Form1* într-un loc unde să atragă atenţia utilizatorului.
Pentru "timer1":
- stabilim *Interval*-ul la care "timer1" va *Tick*-ăi la 1 secundă, deci timer1.*Interval* va fi 1000 (milisecunde)
- stabilim Proprietatea *Enabled* la "True": "timer1" va fi activ de îndată ce *Form1* va fi afişată:
  în acest caz, fiind unicul Dialog al programului nostru, afişarea sa denotă pornirea programului;de asemenea, închiderea sa este totuna cu închiderea programului)



Interfaţa programului, constând desigur numai din Dialogul *Form1*, va putea arăta astfel:

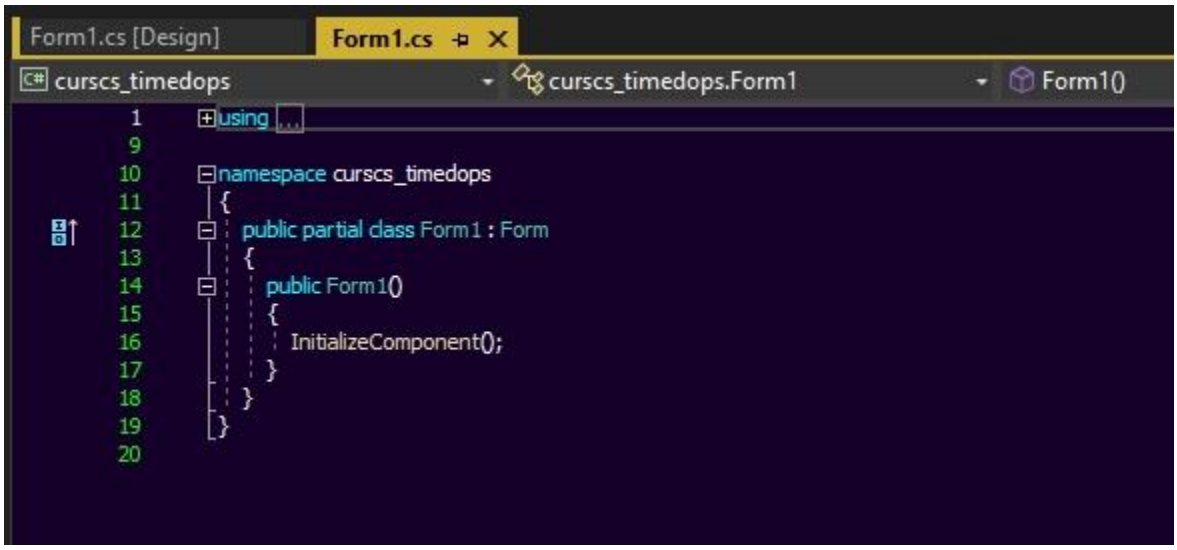


Să scriem codul programului:

Click pe *Form1* pentru a ne asigura că vom scrie codul asociat *acestui* Dialog – *Form1*.
Apăsăm tasta F7 ("View Code") – astfel vom ajunge în modul de scriere a codului.

Înapoi la modul de proiectare a lui *Form1* ne putem întoarce apăsând simultan tastele:
Shift şi F7 ("View Designer").

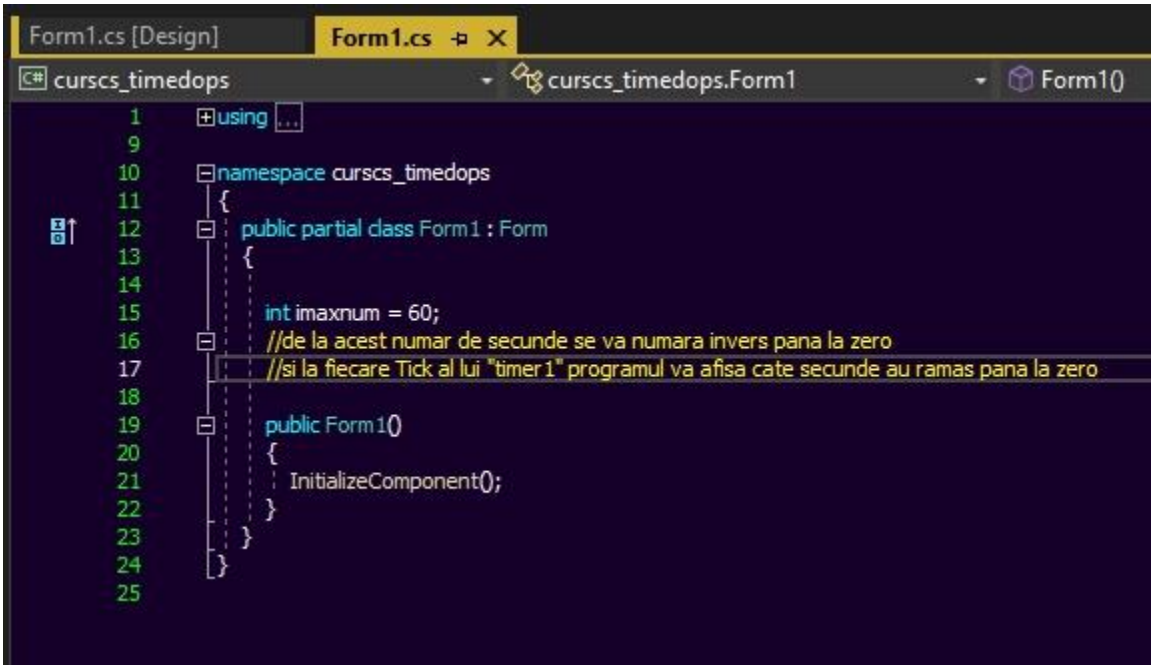
Vedem acest cod, generat automat de către Visual Studio şi care iniţializează *Form1* ca o *clasă*:



Sub declaraţia **public partial class Form1 : Form** declarăm variabila de tip număr întreg care va indica timpul (în secunde) disponibil pentru numărătoarea inversă:

int imaxnum = 60;
//de la acest numar de secunde se va numara invers pana la zero
//si la fiecare Tick al lui "timer1" programul va afisa cate secunde au ramas pana la zero

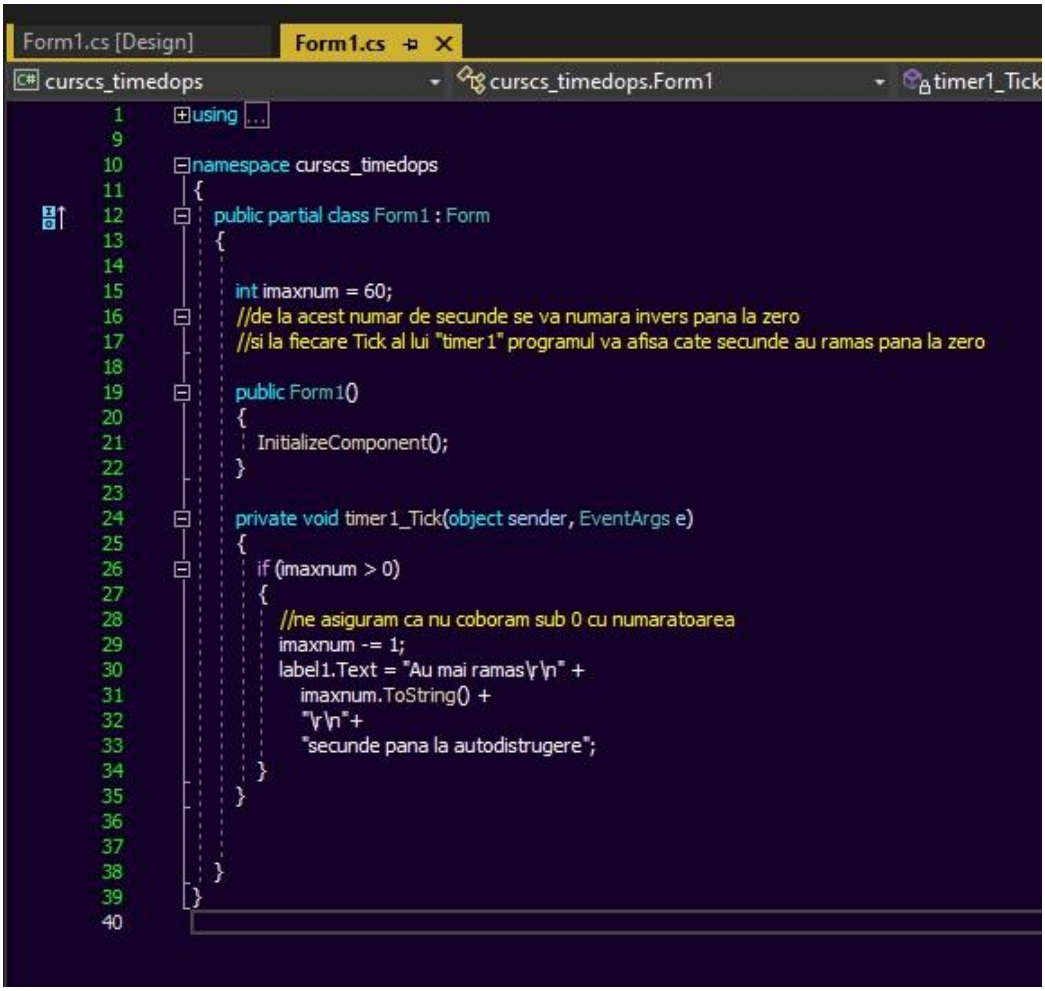
Am declarat această variabilă în acest loc – imediat după declararea tipului *Form1* – pentru ca variabila să fie *accesibilă pretutindeni mai jos de nivelul unde a fost declarată*, deci în orice bloc de instrucţiuni executate în/cu *Form1*.



Înapoi în modul de proiectare a Dialogului *Form1*: Shift+F7

Instruim cronometrul "timer1" ca la fiecare *Tick*-ăit să schimbe mesajul afişat pe eticheta "label1", înât utilizatorul să ştie câte secunde au rămas până la terminarea numărătorii inverse:

Dublu-click pe "timer1":
Ceea ce ne trimite înapoi în modul de scriere a codului, în metoda asociată Evenimentului *Tick* al lui "timer1", metodă în care scriem ca în imagine:




```
private void timer1_Tick(object sender, EventArgs e)
{
        if (imaxnum > 0)
        {
                //ne asiguram ca nu coboram sub 0 cu numaratoarea
                imaxnum -= 1;
                label1.Text = "Au mai ramas\r\n" +
                        imaxnum.ToString() +
                        "\r\n"+
                        "secunde pana la autodistrugere";
        }
}
```

Instrucţiunea
imaxnum -= 1;
scade 1 din valoarea curentă a variabilei *imaxnum* căreia, astfel, îi atribuie această valoare mai mică

Cu "button1" putem opri numărătoarea inversă.
Adică, apăsând (*Click* = click-stânga) pe el.

Pentru aceasta, înapoi în modul de proiectare a *Form1*: Shift+F7:

Dublu-click pe "button1" pentru a solicita Visual Studio să genereze metoda asociată Evenimentului *Click* la lui "button1";

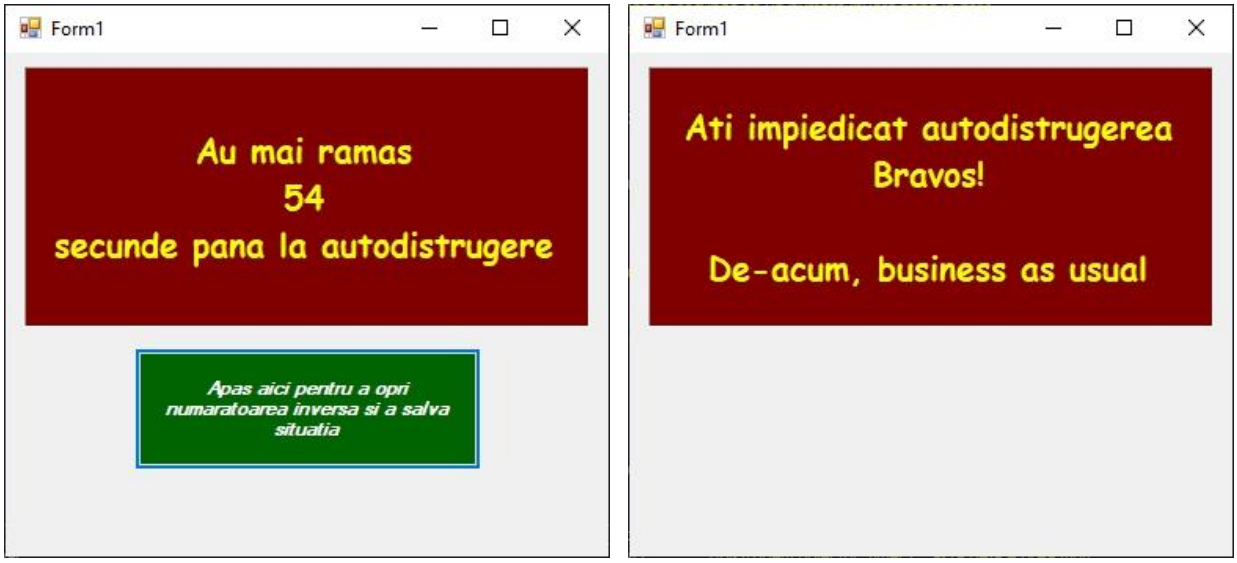
Ceea ce ne trimite înapoi în modul de scriere a codului, unde în acea metodă scriem ca în imaginea următoare care de altfel este întregul cod al programului nostru: circa zece instrucţini scrise manual:



```
private void button1_Click(object sender, EventArgs e)
{
        timer1.Stop();                  //oprim din Tick-ait cronometrul timer1"
        timer1.Enabled = false;         //ne asiguram ca "timer1" e dezactivat
        timer1.Dispose();               //distrugem obiectul "timer1"; asta deja e prea mult
        //
        //schimbam textul afisat pe label1:
        label1.Text =
                "Ati impiedicat autodistrugerea\r\nBravos!\r\n\r\nDe-acum, business as usual";
        //
        //nu mai are rost sa fie afisat butonul de oprire a numaratorii:
        button1.Visible = false;
}
```

Pentru o comunicare mai omenească a programului cu utilizatorul putem declara, pe acelaşi nivel cu *imaxnum*, o variabilă de tip numeric întreg în funcţie de a cărei valoare la oprirea lui "timer1" să îi spunem utilizatorului cu câte secunde înainte de expirarea timpului a oprit numărătoarea inversă.
Încercaţi asta.

Rezultatul efortului de până acum cu acest program:



Felicitări!
You saved the day!

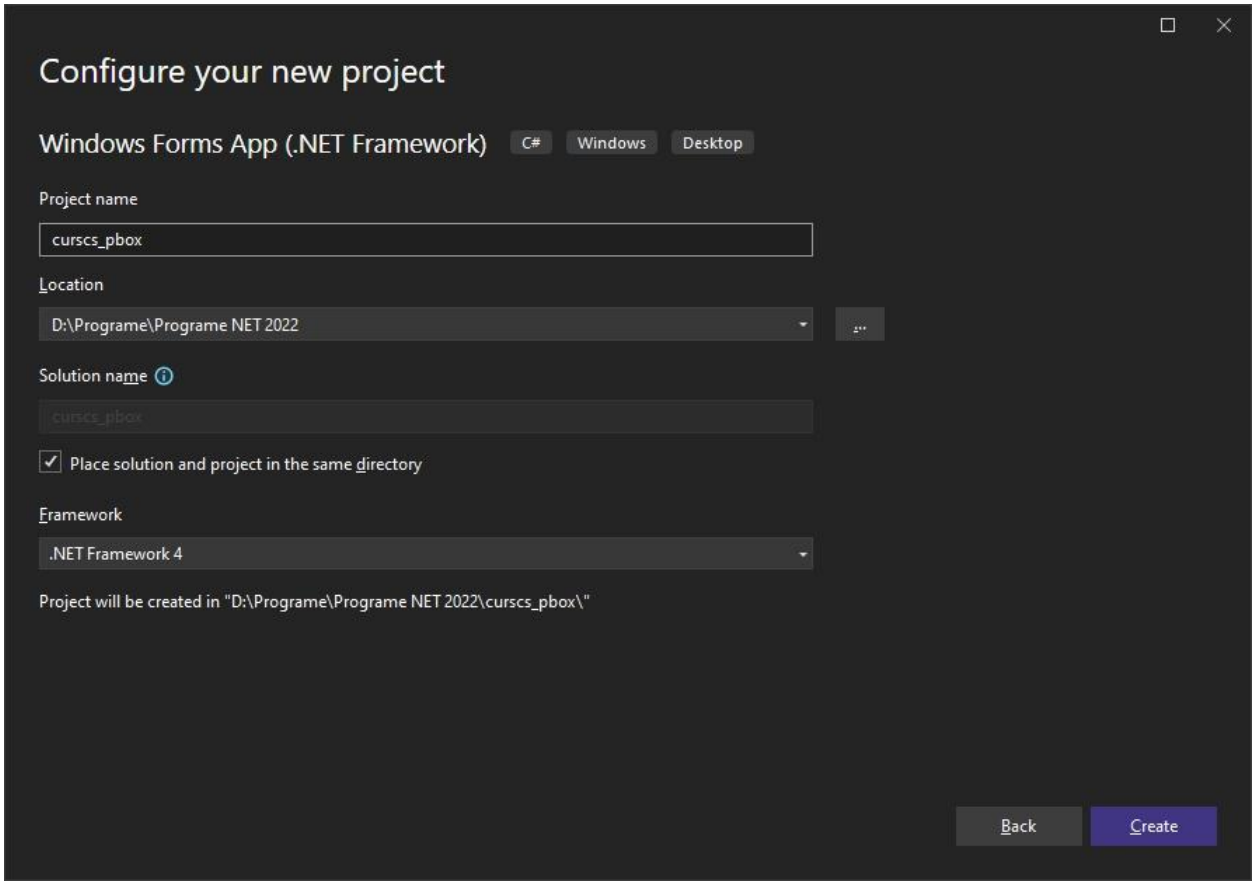
Imagini

Vom alcătui un program în care vor fi afişate imagini preluate atât de pe computerul unde rulează programul nosru, cât şi de pe Internet.

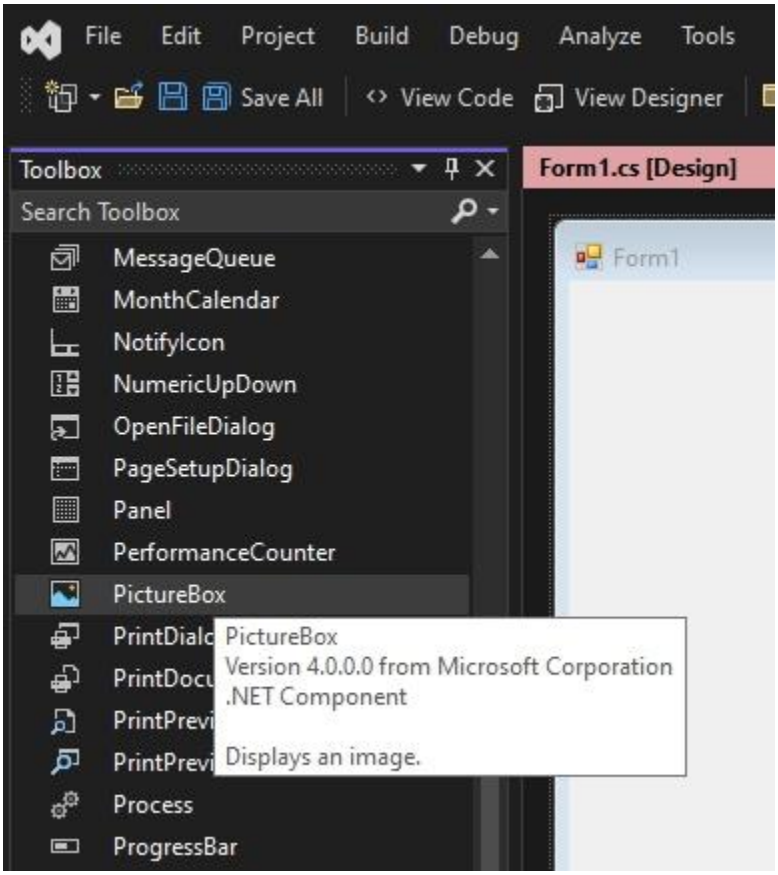
Control-ul utilizat pentru aceasta este *PictureBox* – unul din Controalele cel mai des folosite, la fel cu *Button* şi *Label*. Acesta oferă programatorului şi utilizatorului posibilitatea de a vedea imagini stocate în fişiere cu diverse extensii ca .jpg, .png, .gif.

Pornim Visual Studio, *File -> New... Project...*

În imaginea următoare se vede cum am denumit acest proiect/program "curscs_pbox"; poate fi denumit oricum (dar bineînţeles, cu respectarea convenţiilor aplicabile denumirilor fişierelor din *Windows*):

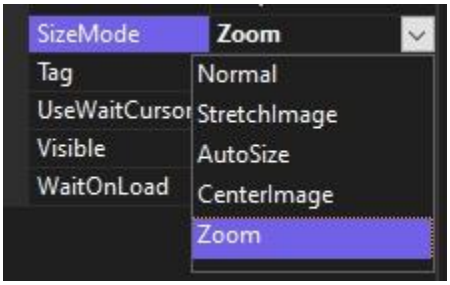


Pe Form1 plasăm un *PictureBox* (primind implicit numele "pictureBox1") şi un *Button* ("button1").



În imaginea următoare se vede cum am modificat Proprietăţi ale acestor două Controale, printre care:

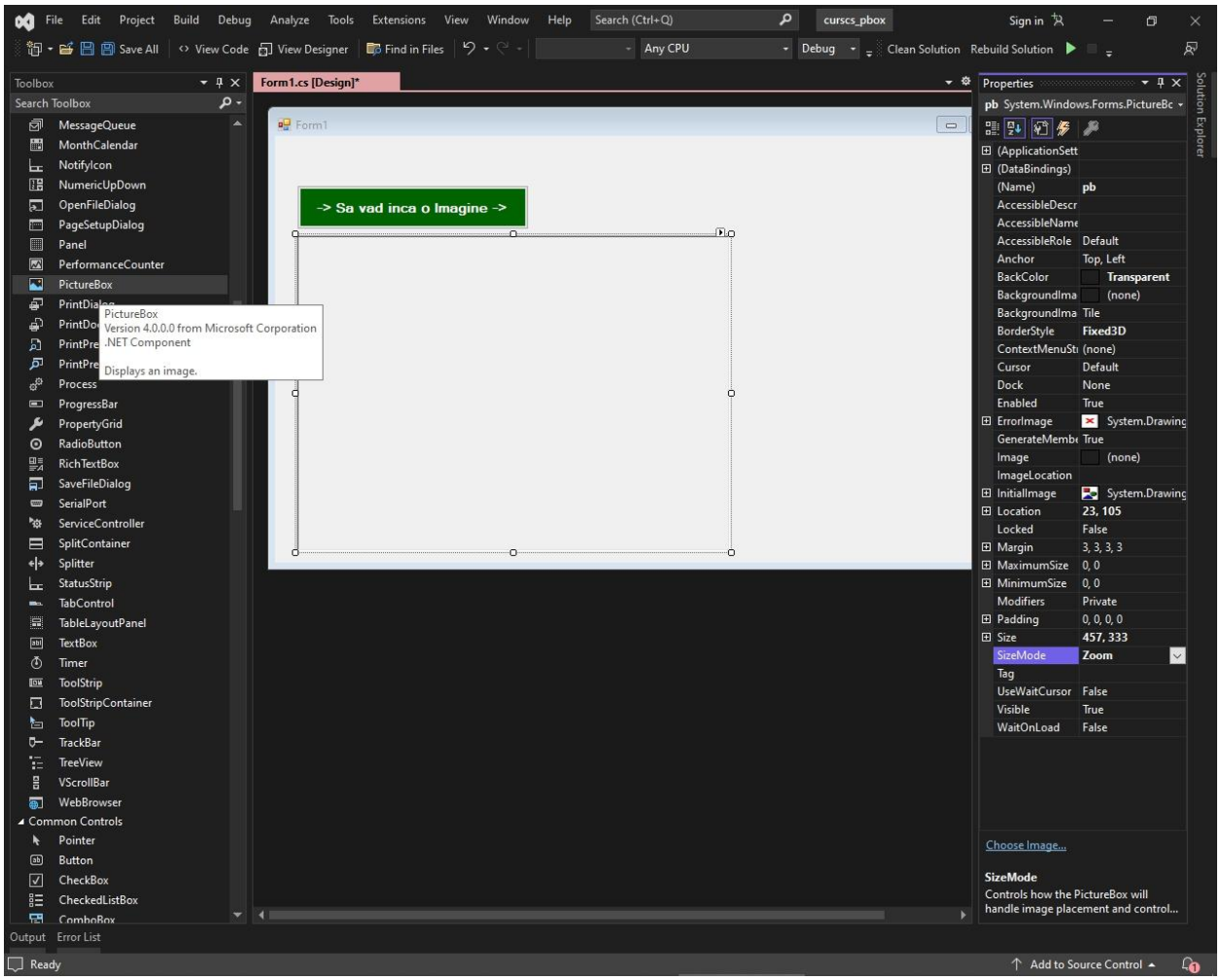
- *BackColor* şi *ForeColor* ale "button1" (ne amintim că putem da alte denumiri Controalelor)
- *BackColor* a "pb": Transparent
- *SizeMode* a "pb": Zoom. *SizeMode* este o proprietate complexă a acestui Control şi care poate avea una din aceste valori care, desigur, pot fi schimbate din cod:



În imaginea precedentă se observă Proprietatea "*Tag*".
Aceasta este o Proprietate comună multor Controale şi poate fi folosită pentru a stoca în valoarea sa diverse informaţii (de exemplu, un text care nu poate fi pus la îndemână în altă parte a programului).

Interfaţa poate arăta astfel:

- am redimensionat "pb" trăgând de colţul său din dreapta-jos
- am aliniat la stânga cele două Controale
- am scris pe "buton1" un text care să-i indice utilizatorului ce anume face acel buton



Click pe "pictureBox1" pentru a o selecta iarăşi, apoi stabilim Proprietatea sa...

- *Visible*: True. Proprietate comună tuturor Controalelor cu contur, vizibile pe un Dialog: Controlul fie este vizibil, fie nu.
  Ca majoritatea Proprietăţilor unui Control, aceasta poate fi stabilită/modificată din cod
  Bineînţeles, dacă vrem ca "pb" să NU fie vizibilă la pornirea programului (la afişarea Dialogului *Form1* pe care se află), o stabilim la False.
- SizeMode: am stabilit-o/setat-o la "Zoom", ceea ce înseamnă că imaginea originală va fi redimenisonată încât să încapă în Controlul de tip PictureBox

Acum să scriem cod pentru afişarea unor imagini în "pb":

Avem câteva imagini (fişiere cu extensiile .jpg şi .png" într-un director pe computerul unde rulează acest program şi vrem să le afişăm pe rând în "pb":

Pentru asta, trebuie să accesăm din cod acele fişiere – bineînţeles, după calea spre ele (locaţia lor) şi denumirea fiecărui astfel de fişier.

Cum am aflat în exemplificarea construirii unui editor de text asemănător cu *Notepad*, spaţiul de nume **System.IO** conţine membri (metode, *Enum*erări ş. a.) utile exact lucrului cu fişiere – sugestie provenind chiar din numele acestuia: IO = **I**nput-**O**utput.

Vom folosi un membru al acestui spaţiu de nume pentru a găsi **toate fişierele având anumite extensii şi care se află într-un director/folder anume**.

Acesta este membrul:

System.IO.*Directory*

La rându-i, *Directory* are metode pentru manipularea unui director (folder) ca atare, inclusiv a conţinutului său (poate conţine atât fişiere, cât şi directoare/foldere).

Aici, folosim *Directory* pentru a obţine calea spre şi numele fişierelor din directorul "D:\z_test\test_img":

Dublu-click pe Form1, ceea ce ne duce în modul de scriere a codului unde scriem ca în imaginea următoare unde se vede că:

- am declarat o variabilă de tip tablou (*Array*) de *string*-uri în care vor fi plasate numele "complete" ale fişierelor din acel director
- în metoda Evenimentului *Load* al lui *Form1* căutăm (deci, chiar la afişarea Dialogului; aceasta nu este o mişcare înţeleaptă – accesarea sistemul de fişiere chiar la pornirea programului, dar dacă nu avem de ales facem şi asta, având însă grijă să cuprindem codul trespectiv într-un bloc *try-catch* **ca de fiecare dată, de altfel, când lucrăm cu fişiere**:

```
private void Form1_Load(object sender, EventArgs e)
{
        try
        {
                sfilenames = System.IO.Directory.GetFiles(
                        "D:\\z_test\\test_img",
                        "*.jpg");
                MessageBox.Show("In directorul specificat se afla:\r\n"+
                        sfilenames.Length.ToString() + " fisiere cu extensia .jpg");
        }
        catch (Exception ex)
        {
        }
}
```

Ştim deja cum să citim (mesajul/textul afişat cu) instrucţiunea **MessageBox.Show.**

Versiunea folosită aici a metodei *GetFiles* a lui *Directory* primeşte doi parametri (două argumente):

- primul este denumirea directorului/folderului unde vor fi căutate ("get files") fişierele, aici aceasta fiind: **"D:\\z_test\\test_img"**
- al doilea este "filtrul" aplicat acestei metode – anume, extensia fişierelor care vor fi întoarse ca rezultat într-o variabilă de ti tablou de *string*-uri – pe aceasta am declarat-o deja: **string[] sfilenames =...** şi o folosim acum): **sfilenames = ...**

Vrem ca la fiecare apăsare pe "button1", să fie afişată o imagine găsită mai devreme, la încărcarea/afişarea *Form1*.

Vrem să afişăm *toate* imaginile găsite, una după alta, apăsarea butonului solicitând imaginea următoare celei afişate (sau pe prima dacă 1) nu este afişată niciuna, sau 2) dacă este afişată ultima)

Pentru asta, mergem înapoi în modul de proiectare a lui *Form1*: Shift+F7

Dublu-click pe "button1", ceea ce iarăşi ne trimite la cod, în metoda Evenimentului *Click* al lui "button"1", metodă unde scriem:

Pentru a ţine evidenţa imaginii curente afişate în "pb" trebuie să ştim la ce poziţie/index în tabloul *sfilenames* se află denumirea fişierului respectiv:

Pentru asta, pe acelaşi nivel cu declararea lui *sfilenames*, anume **string[] sfilenames =...** scriem:

int icp=0; //o denumire cat de cat sugestiva: variabilă de tip numeric intreg, "current", "picture"

```
string[] sfilenames;
int icp = 0;   //o denumire cat de cat sugestiva: variabil` de tip numeric intreg, "current", "picture"
```

În metoda button1_Click scriem instrucţiunile de mai jos şi din figura următoare, instrucţiuni care deja nu trebuie a pune dificultăţi de intrepretare:

```
private void button1_Click(object sender, EventArgs e)
{
        if (sfilenames.Length > 0)
        {
                if (icp < sfilenames.Length - 1)
                {
                        icp++;
                        pb.Image = Image.FromFile(sfilenames[icp]);
                }
                else
                {
                        icp = 0;
                        pb.Image = Image.FromFile(sfilenames[icp]);
                }
        }
}
```

Instrucţiunea

if (sfilenames.Length > 0)

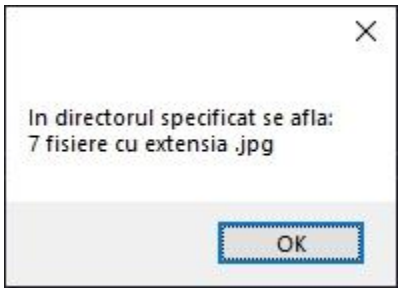
este inutilă aici, dar ilustrează cum ne putem asigura că a fost găsit fie şi un sigur fişier cu extensia specificată în metoda Directory.*GetFiles*.

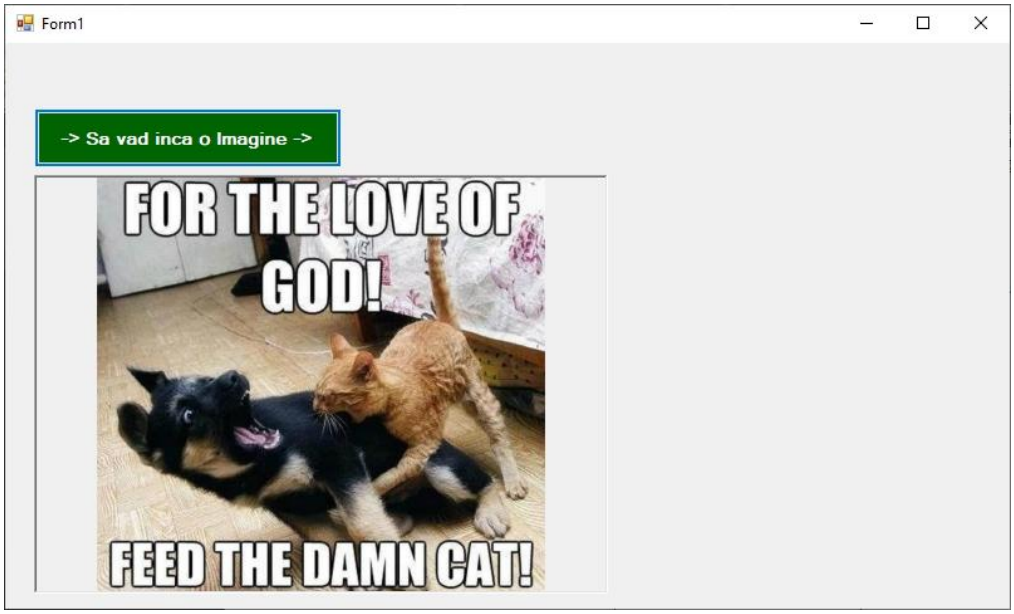
else

înseamnă:
Dacă a fost afişată ultima imagine – indicată de ***ultimul element***, aflat la ***indexul snumfiles.Length-1***
al tabloului *snumfiles*,
atunci reiniţializăm variabila *icp* cu care sunt parcurse elementele tabloului
şi ciclul se reia cu prima imagine – indicată de ***primul element***, aflat la ***indexul 0***
al tabloului *snumfiles*

Pornim programul: F5:

Rezultatul:
asemănător celor două imagini din continuare:





Se observă cum Proprietatea *SizeMode* a lui "pb" redimensionează fiecare imagine încât să fie afişată integral în "pb".

Pentru a afişa în programul nostru o imagine aflată pe Internet
apelăm
metoda Load a Controlului "pb". Aceasta este o Proprietate a Controlului PictureBox.

Pe *Form1* plasăm încă un *Button* care va fi implicit denumit "button2".

Dublu-click pe "button2", iar în metoda Evenimentului său *Click* scriem:

```
        private void button2_Click(object sender, EventArgs e)
        {
                pb.Load("http://live.cdn.renderosity.com/gallery/items/3089941/images/2073883/674
c3cbb27186d277aff0065644e634f_original.jpg");
        }
```

Se vede cum metodei *Load* a lui "pb" i-a fost trimis parametrul/argumentul constând în
adresa pe Internet a imaginii de afişat în "pb".

Încercaţi cu diverse adrese, imagini din fişiere locale şi diverse valori ale Proprietăţii *SizeMode* a lui "pb".

Colecţii

Vom crea un program în care vom prelucra *colecţii* de informaţie, fiecare colecţie referindu-se la un anumit subiect.

Astfel, vom folosi tipurile de *Collections*:

- tablou "simplu" de valori: *Array*
- NameValueCollection: acest tip de colecţii permite stocarea unri cantităţi relativ mari de informaţie-text asociate unui subiect/text/termen de dicţionar/cuvânt, ***subiectul fiind UNIC în colecţie şi astfel formând o pereche UNICĂ alături de informaţia asociată***.
  De exemplu, subiectului/termenului "azi" îi pot fi asociate: "cer senin", "vant slab", "23": "azi" se numeşte *cheia* ("Key") unică prin care această pereche de informaţii va fi identificabilă şi accesibilă ca parte a colecţiei, iar cele trei texte/*string*-uri separate prin virgulă se numesc *valoarea* ("Value) perechii.

În programul nostru:

- vom crea o astfel de NameValueCollection pentru regiuni din România, colecţie formată din perechi (numeleregiunii, informaţiiasociate)
- vom alcătui un text descriptiv şi relativ complex cu particulele de informaţie stocate în Valoarea fiecărei perechi din colecţie

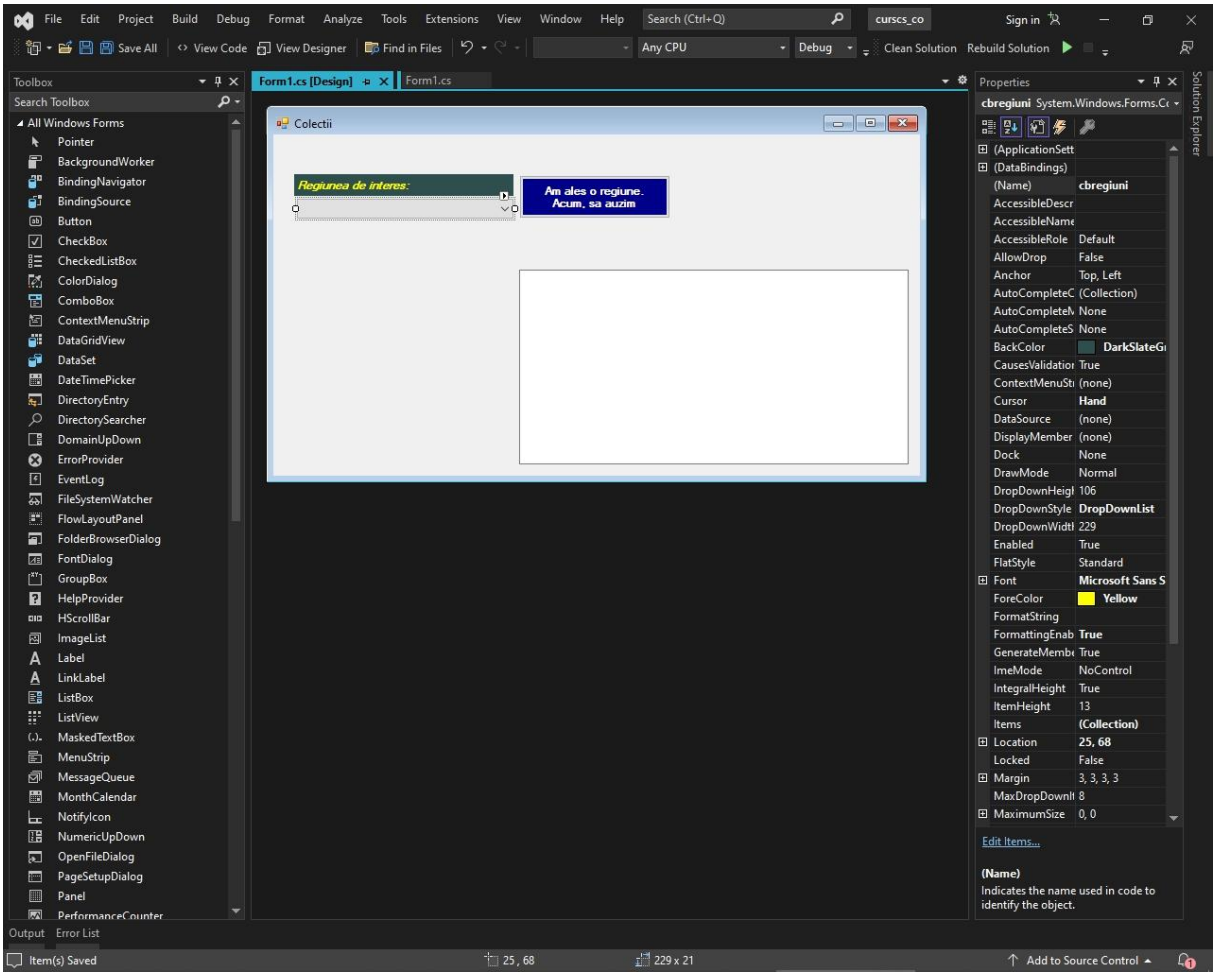
Începem:
Pornim Visual Studio, *File -> New... Project...*

Pe Form1 tragem câte un Control din acestea:

- un *Button* pe care îl denumim "binfo" (Proprietatea sa *Name*), îl colorăm şi îl redimensionăm încât textul conţinut să fie vizibil, contrastant cu fundalul butonului şi inteligibil
- o casetă *TextBox* de editare a textului pe care o denumim "tbinfo", o colorăm după bunul plac şi neapărat îi stabilim Proprietatea *Multiline* la True, fiindcă textul informativ despre o regiune va fi afişat pe mai multe rânduri; mai mult, neştiind acum cât de lung va fi textul afişat acolo, îi oferim utilizatorului posibilitatea de a derula în sus şi în jos textul: stabilim Proprietatea *ScrollBars* la "Both" (vor fi vizibile ambele bare, verticală şi orizontală, pentru derularea textului), sau măcar la "Vertical"
- o casetă derulantă *ComboBox* pe care o denumim "cbregiuni".
  Acest Control este util atunci când utilizatorul trebuie să selecteze o opţiune (un text) din mai multe asemenea opţiuni. Depinzând de unele Proprietăţi ale casetei, lista textelor/opţiunilor va fi vizibilă complet sau parţial (de exemplu, numai 4 opţiuni pot fi vizibile odată, deşi în casetă sunt 15 opţiuni – astfel că dacă trebuie selectată o opţiune invizibilă lista trebuie derulată în sus sau în jos, vertical deci până ce se poate selecta acea opţiune)
- o etichetă (*Label*) pe care scriem ceva referitor la ce anume foloseşte caseta derulantă

Redimensionăm, aliniem, colorăm (în general, aceasta este ordinea) Controalele încât să dăm un aspect decent interfeţei, de exemplu (nu e un exemplu ideal, dar măcar culorile nu deranjează):

Interfaţa, Dialogul *Form1* poate arăta astfel:



Să scriem cod:

Vrem ca la afişarea *Form1*/la pornirea programului, în "cbregiuni" să fie plasate numele tuturor regiunilor.
Acel moment este asimilabil (nu integral, dar pentru ce ne interesează este suficient) Evenimentului *Load* al Dialogului *Form1*.

Deci, pentru ca popularea "cbregiuni" să se facă la Evenimentul *Load* al lui *Form1*, care după cum deja ştim este Evenimentul implicit asociat unei *Form*-e:
în modul de Proiectare/Design al lui *Form1* dublu-click oriunde pe *Form1*, dar NU pe un Control:
ceea ce ne trimite în modul de scriere a codului, unde scriem:

```
string[ ] sa_regiuni = { "Banat", "Crisana", "Dobrogea", "Moldova",
                "muntenia", "oltenia", "Transilvania_nord", "Transilvania_sud" };

private void Form1_Load(object sender, EventArgs e)
{
        for (byte b = 0; b < sa_regiuni.Length; b++)
        {
                cbregiuni.Items.Add(sa_regiuni[b].ToUpper());
        }
}
```

Se observă şi se ia aminte la acestea:

- locul-nivelul unde am declarat variabila tablou cu numele regiunilor, încât să fie cunoscută de la aceat nivel inclusiv în jos
- în instrucţiunea unde se defineşte tabloul de texte (*string*-uri) cu numele regiunilor de adăugat în caseta "cbregiuni" iniţiala numelor "muntenia" şi "oltenia" nu este majusculă
- pentru parcurgerea (ciclul *for*) tabloului a fost folosită o variabilă de tip *byte* (cu valori de la 0 la 255), fiindcă domeniul său de valoir este suficient – tabloul are numai 8 elemente.Pentru un ciclu *for* pot fi folosite toate tipurile **numeric întreg**, de exemplu *short*, *int*, *long*; aceasta se stabileşte *în acord cu numărul* presupus cunoscut *al iteraţiilor*; dacă numărul iteraţiilor nu este cunoscut, atunci o variabilă de tip *int* este suficientă.

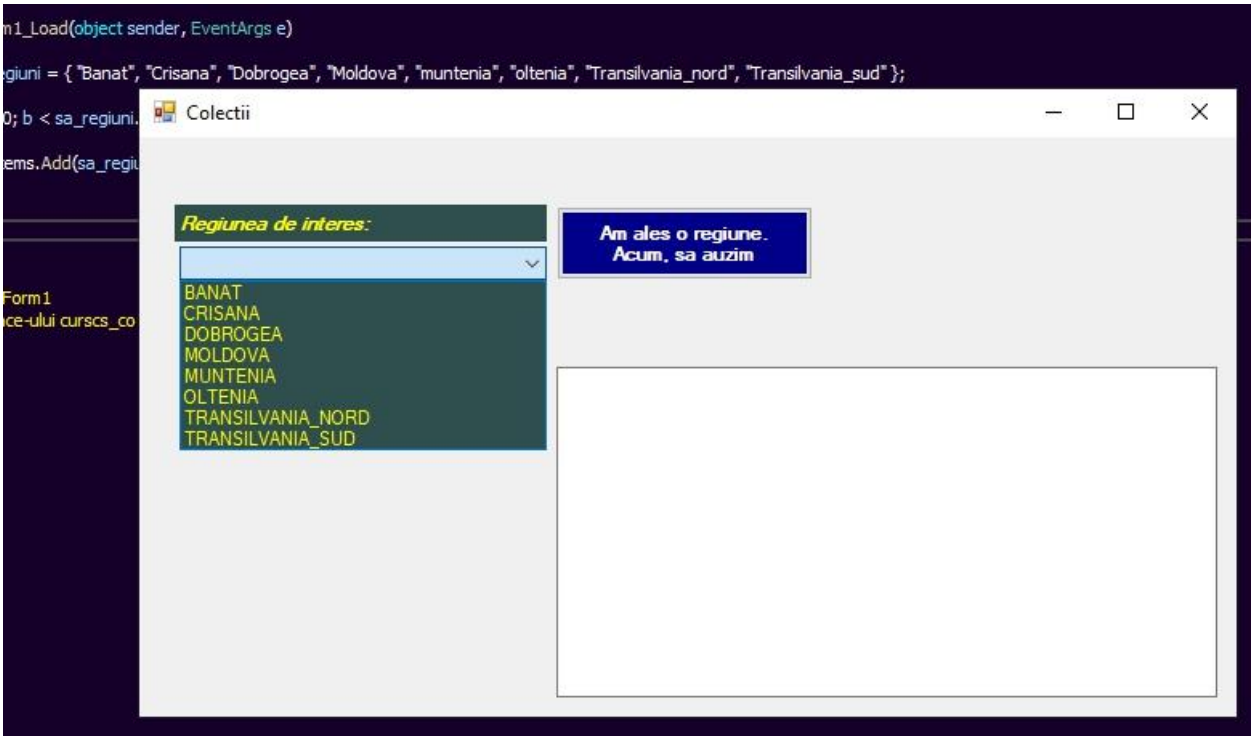
În acest exemplu, rezolvăm problema cu erorile de scriere a numelor trasformând cu totul numele regiunilor la cele scrise cu majuscule:

- metoda *ToUpper()* transformă o variabilă de tip *string* la una în care toate caracterele sunt MAJUSCULE
- metoda *ToLower()* face operaţiunea inversă lui *ToUpper()*
- metodele *ToUpperInvarian()* şi *ToLowerInvariant()* sunt similare şi pot fi folosite în locul acestora

deci textul aflat la indexul *b* în tabloul sa_regiuni va fi transformat în majuscule...
sa_regiuni[b].ToUpper()

... şi adăugat listei din caseta "cbregiuni":
cbregiuni.Items.Add(sa_regiuni[b].ToUpper());

Pornim programul – tasta F5 - şi verificăm rezultatul/funcţionalitatea a ceea ce am scris până acum:



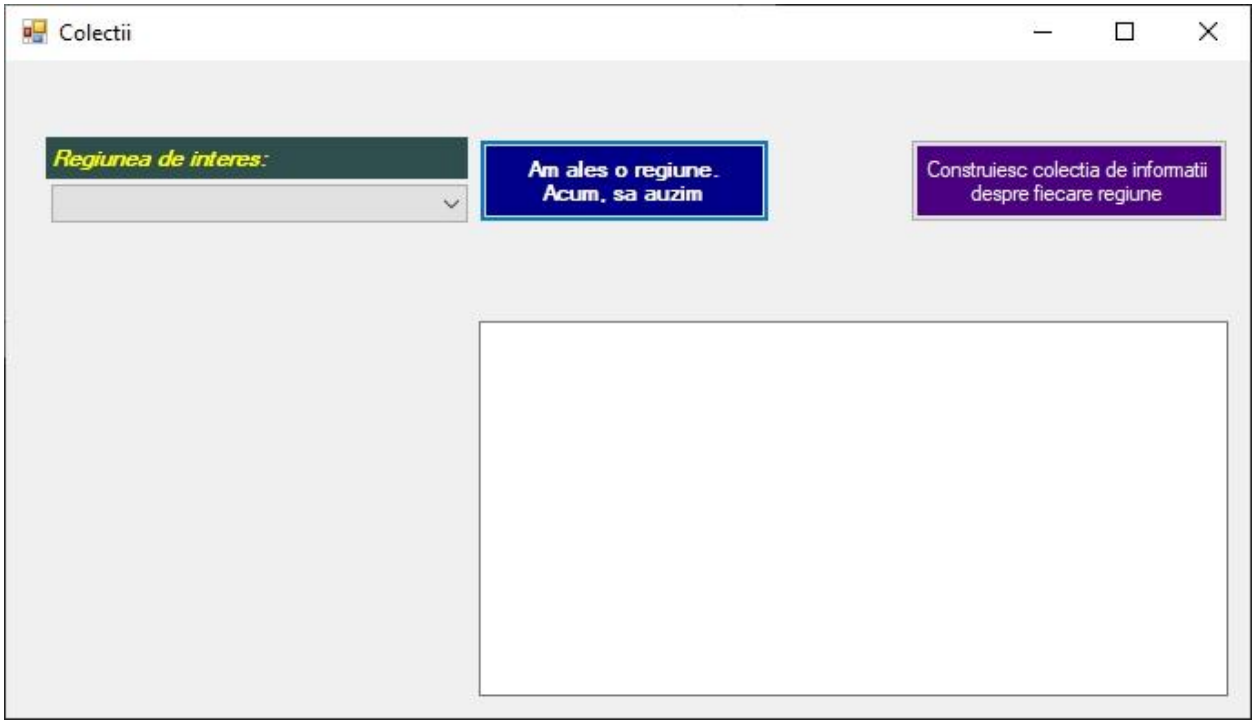
Pare OK.
De fapt, chiar este OK: programul a făcut ce am vrut.

Închidem programul /Dialogul *Form1* – ca de obicei, click pe X în dreapta-sus şi înapoi în Visual Studio.

Pe *Form1* tragem încă un *Button*:
Acesta va folosi la popularea colecţiei cu informaţii despre fiecare regiune.

Îl redenumim "bcreg", de exemplu; este recomandat pentru ca, a nu ne încurca noi înşine în Controale, să denumim fiecare Control cu un nume lung, dar oricum – un nume sugestiv pentru ceea ce face fiecare Control. În cazul acesta, am dat un nume scurt: "bcreg": ***b***uton-***c***olecţii-***reg***iuni

De-acum, *Form1* ar putea arăta aşa:



Înapoi în cod (tasta F7), unde sub declaraţia pentru tabloul *sa_regiuni* declarăm acea variabilă de tip *NameValueCollection* care va conţine informaţii despre fiecare regiune:

```
string[ ] sa_regiuni = { "Banat", "Crisana", "Dobrogea", "Moldova",
                         "muntenia", "oltenia", "Transilvania_nord", "Transilvania_sud" };
System.Collections.Specialized.NameValueCollection nvc_reginfos =
                         new System.Collections.Specialized.NameValueCollection();
```

Înapoi în modul de proiectare/Design al *Form1*,
dublu-click pe *Button*-ul "bcreg" tocmai adăugat, iar
în metoda Evenimentului *Click* al său scriem:

```
private void bcreg_Click(object sender, EventArgs e)
{
        foreach (string snumeregiune in sa_regiuni)
        {
                nvc_reginfos.Add(snumeregiune.ToUpper(), "");
        }
        //
        //urmeaza instructiuni pentru popularea colectiei
        //cu informatii despre fiecare regiune:
        nvc_reginfos["BANAT"] = "cer variabil, vant slab, 25";
        nvc_reginfos["CRISANA"] = "mai mult senin, vant slab, 24";
        nvc_reginfos["DOBROGEA"] = "senin, vant slab, 15";
        nvc_reginfos["MOLDOVA"] = "senin, vant slab, 12";
        nvc_reginfos["MUNTENIA"] = "mai mult senin, vant in general slab, 23";
        nvc_reginfos["OLTENIA"] = "cer variabil, vant in general slab, 23";
        nvc_reginfos["TRANSILVANIA_NORD"] =
                                        "cer variabil, vant slab pana la moderat, 1";
        nvc_reginfos["TRANSILVANIA_SUD"] =
                                        "cer variabil, vant in general moderat, 25";
}
```

Se observă cum la colecţie au fost adăugate de asemenea numele transformate la majuscule (*ToUpper()*).
Fiecare linie se citeşte astfel:

- elementul din colecţie identificat prin numele regiunii va avea valoarea cutare

Acum să încropim un text, un mesaj descriptiv oarecum detaliat folosind particulele de informaţie din fiecare pereche-element al variabilei "nvc_reginfos" de tipul *NameValueCollection*, pereche formată din (Cheie şi Valoare):

Înapoi în modul de proiectare/Design (Shift+F7):
Dublu-click pe butonul marcat "Am ales o regiune. Acum, sa auzim" sau similar, la apăsarea căruia va fi solicittat acel text descriptiv despre regiunea selectată în caseta derulantă "cbregiuni":

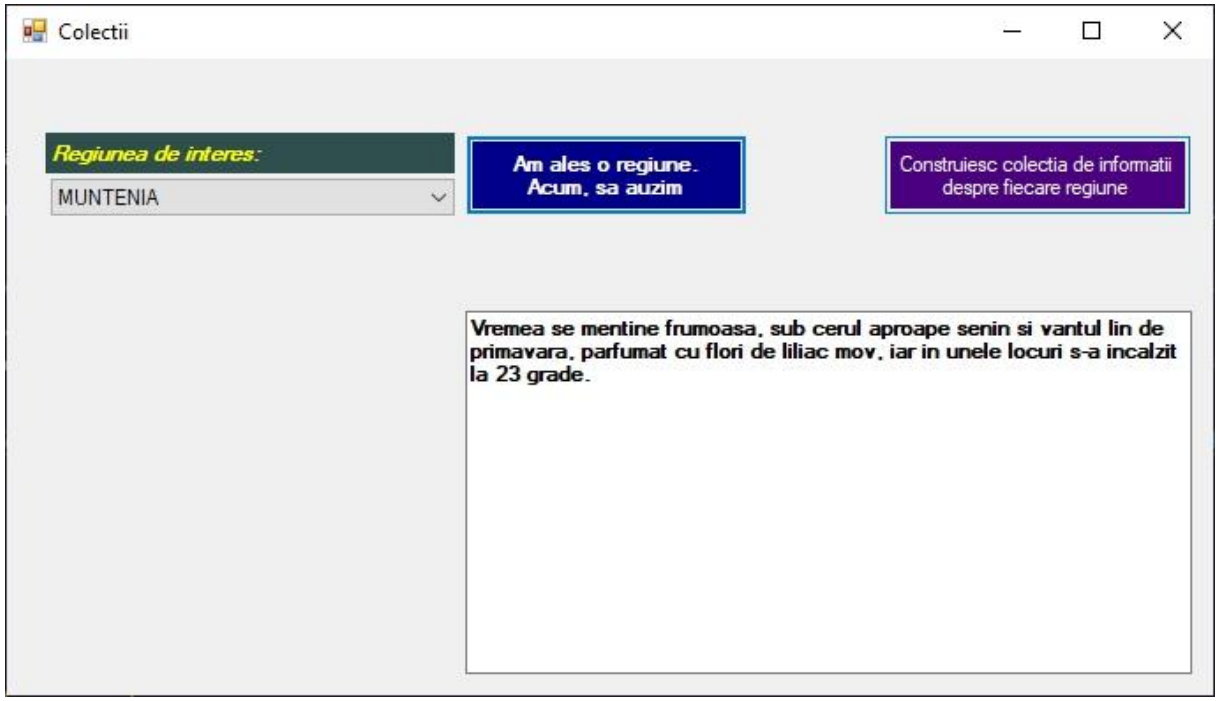
În metoda Evenimentului *Click* al acestui buton scriem:

Codul poate fi *Copy*-at aşa cum este şi plasat (*Paste*) în Visual Studio:
Se observă textele descriptive/*string*-urile întinse pe două sau pe trei rânduri,
dar codul poate fi luat aşa cum este:

```
private void binfo_Click(object sender, EventArgs e)
{
	if (cbregiuni.SelectedIndex > -1)
	{
		switch (cbregiuni.SelectedItem.ToString())
		{
			case "BANAT":
				tbinfo.Text = "Cerul are innorari trecatoare, aerul este linistit, iar temperatura aerului din unele locuri este de 25 grade.";
				break;
			case "CRISANA":
				tbinfo.Text = "Pe cer nu e niciun nor demn de atentie, aerul este calm, iar desi temperatura aerului a crescut rapid pana acum inca nu s-au depasit 24 grade.";
				break;
			case "DOBROGEA":
				tbinfo.Text = "Vremea este frumoasa, in conditiile cerului senin si vantului slab, dar temperatura aerului a ajuns abia la 15 grade.";
				break;
			case "MOLDOVA":
				break;
				tbinfo.Text = "Vremea a ramas frumoasa inca de dimineata devreme, vantul abia se simte, dar este racoare - temperaturi de pana la numai 12 grade.";
			case "MUNTENIA":
				tbinfo.Text = "Vremea se mentine frumoasa, sub cerul aproape senin si vantul lin de primavara, parfumat cu flori de liliac mov, iar in unele locuri s-a incalzit la 23 grade.";
				break;
			case "OLTENIA":
				tbinfo.Text = "Vremea este predominant frumoasa, pe cer sunt cativa nori, vantul nu se remarca prin nimic si este la fel de cald ca in Muntenia - cu temperaturi de pana la 23 grade.";
				break;
			case "TRANSILVANIA_NORD":
				tbinfo.Text = "Aspectul vremii este in general frumos, determinat esentialmente de cerul pe care din cand in cand trec nori de vreme buna si de vantul care cel mult zburleste coafura, iar aerul abia de a trecut de 20 grade in unele locuri.";
				break;
			case "TRANSILVANIA_SUD":
				tbinfo.Text = "Vremea este schimbatoare sau se prefigureaza astfel, sub cerul variabil si vantul care de curand s-a mai intetit oleaca, dar este cald - chiar 25 grade in unele insule de caldura urbana.";
				break;
			default:
				break;
		}
	}
}
```

În caseta derulantă "cbregiuni" am selectat elementul (textul) "MUNTENIA":
Apoi am apăsat butonul cu care solicităm textul mai lung, descriptiv referitor la elementul selectat:

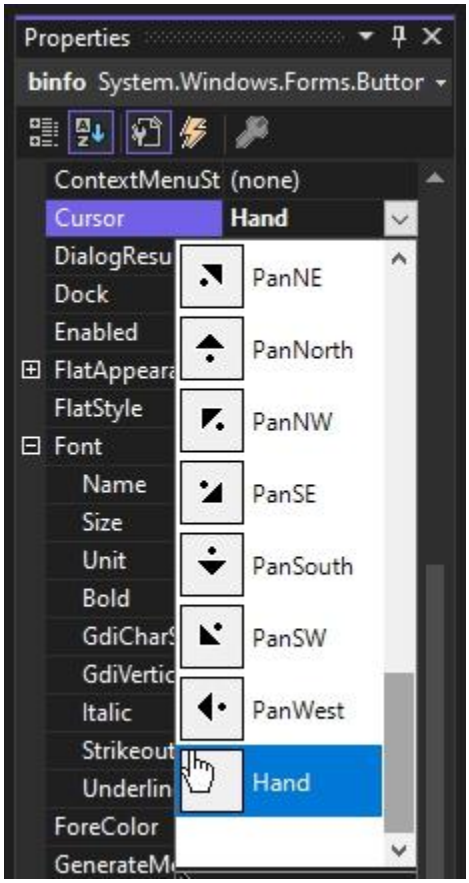
Rezultatul:



Pentru a da un aspect încă mai plăcut programului nostru, stabilim forma *pointer*-ului pentru Controale care de obicei trebuie să răspundă la Click:

- *Button*
- *ComboBox*

În acest fel:
În modul de proiectare/Design al *Form*-ei:
Se selectează Controlul – click-stânga pe el
Se apasă tasta F4 – va fi afişată (dacă nu este deja) secţiunea *Properties* - Proprietăţile Controlului
Valoarea Proprietăţii *Cursor* a Controlului se schimbă la "*Hand*" (sau similar, pictograma este sugestivă):



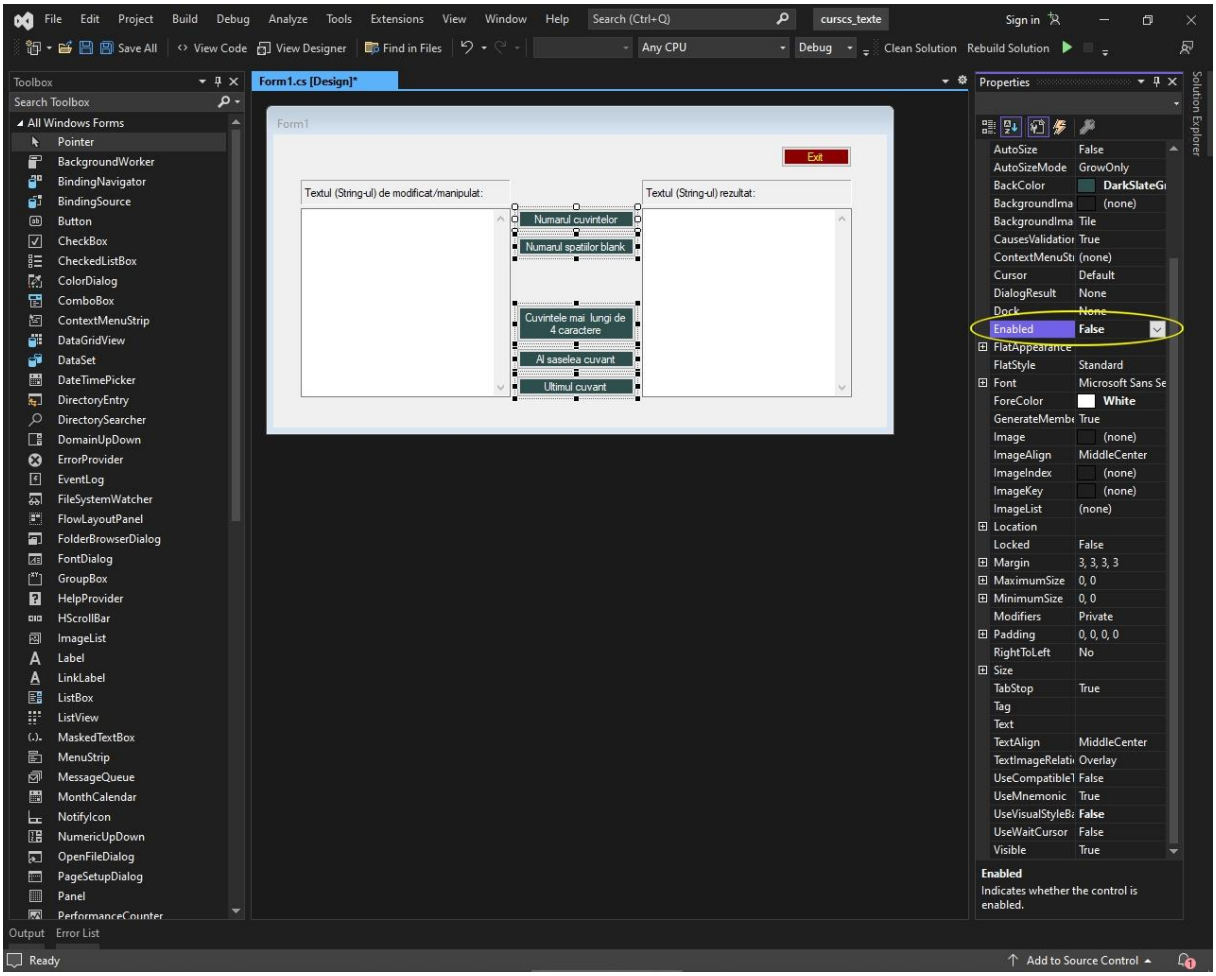
Încercaţi cu diverse colecţii de tipurile folosite aici, cu diverse Cursoare şi cu diverse aspecte ale interfeţei.

Nu aveţi ce strica.
Încă

Operaţiuni frecvente cu text (String)

Vom crea un program în care un text va fi prelucrat încât din el să fie extras un anumit cuvânt, numărul cuvintelor şi cel al spaţiilor libere/*blank*-urilor şi cuvintele de o anumită lungime.

Interfaţa *Form1* ar putea arăta astfel, unde toate Controalele necesare sunt prezente:



Se observă cum sunt selectate toate Controalele de tipul Button; selectarea se face astfel:

- click-stânga pe un Control
- ţinută apăsată tasta *Ctrl*
- click-stânga pe celelalte Controale pe care vrem sa le selectăm

Facem asta atunci când vrem să stabilim Proprietăţi comune acelor Controale – în acest caz, Propeietatea *Enabled* (activ, cu care utilizatorul poate interacţiona prin mouse/tastatură/un dispozitiv similar acestora).

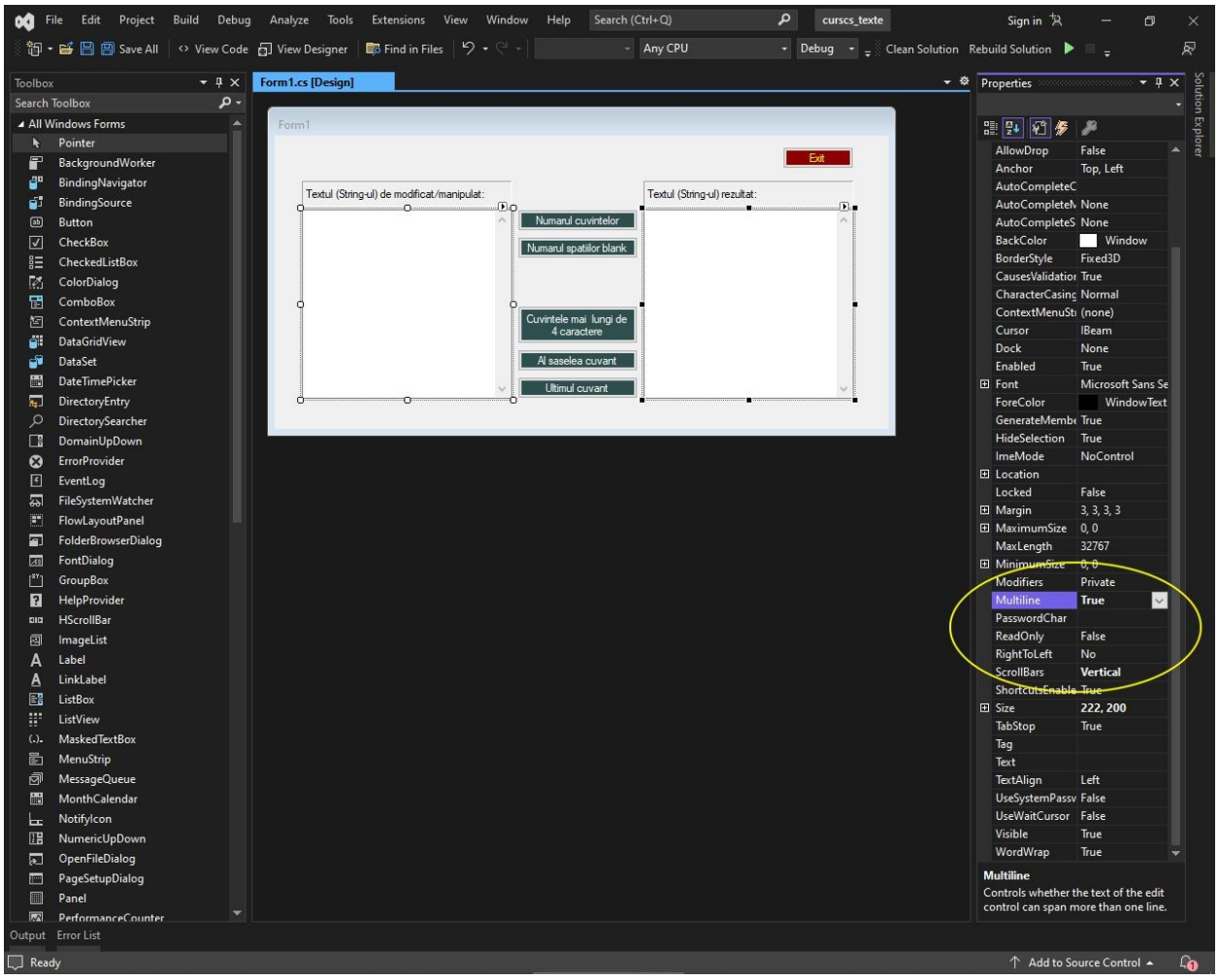
Am stabilit Proprietatea *Enabled* a tuturor butoanelor fiindcă, bineînţeles:

- aplicăm logica în proiectarea interfeţei programului şi Dialogurilor (*Form*-elor) individuale
- astfel încât, în acest exemplu, utilizatorul să nu poată solicita prelucrarea textului "iniţial" – aflat în caseta de editare din stânga - dacă nu există niciun astfel de text

Selectăm *ambele* casete de editare a textului în acelaşi fel cum am selectat toate butoanele, apoi:

Stabilim Proprietatea *MultiLine* a *ambelor* casete de editare a textului la: "True":
Ne asigurăm că *în fiecare casetă va fi afişat în mod corect un text întins pe mai multe linii/rânduri*.

Stabilim Proprietatea *ScrollBars* a *ambelor* casete de editare a textului la: "True":
Ne asigurăm că *în fiecare casetă va fi vizibilă, la nevoie, bara de derulare verticală a textului conţinut*.



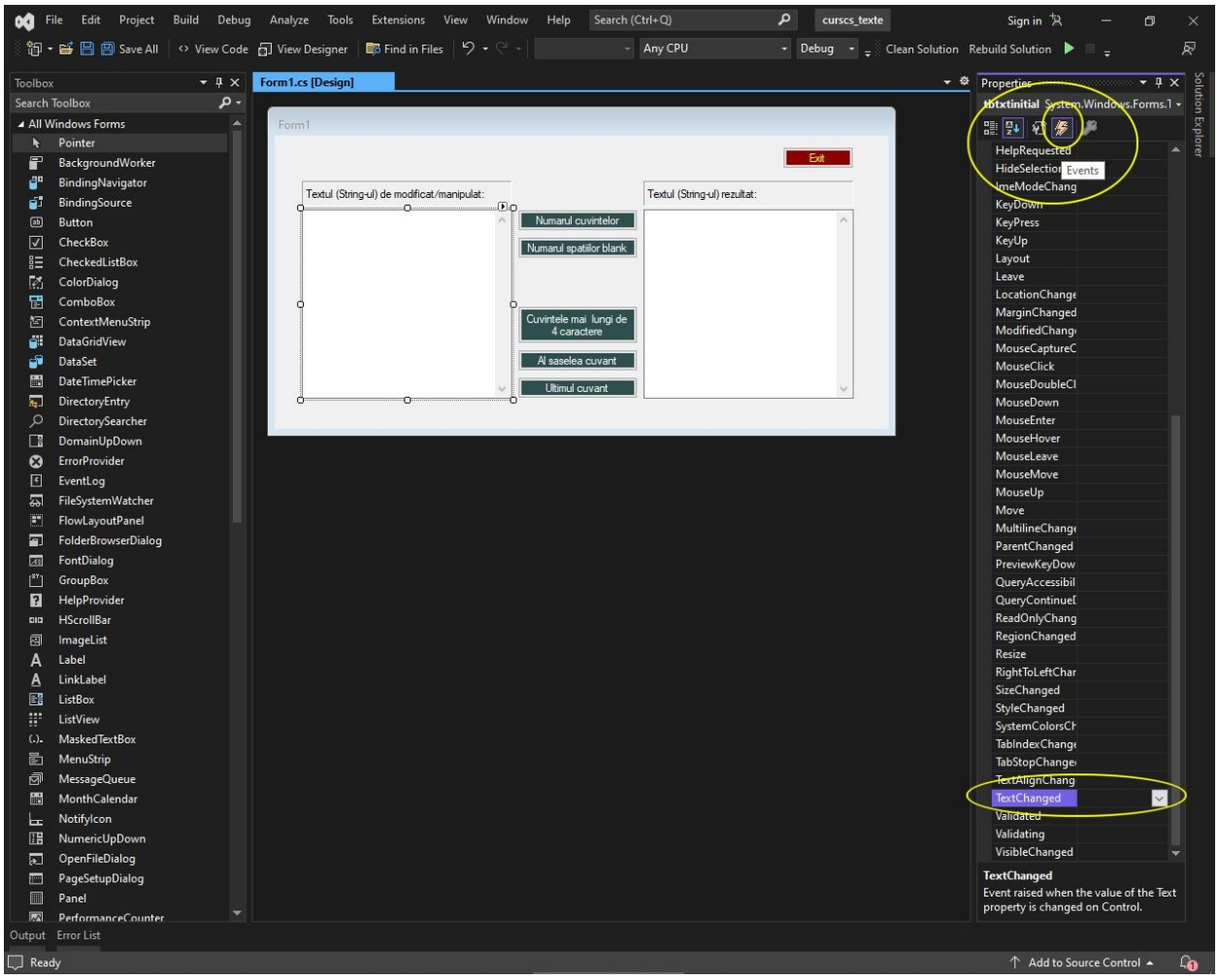
Vrem ca butoanele de prelucrare a textului "iniţial" să fie:

- activate atunci când acel text va fi alcătuit din cel puţin o literă/un caracter care nu este spaţiul liber/*blank*
- dezactivate când textul "iniţial" este fie nul (absent), fie alcătuit numai din spaţii libere/*blank*-uri

Ne dăm seama că este implicat un Eveniment al acelei casete de text – Eveniment referitor la ce anume text scrie utilizatorul.

Într-adevăr, un astfel de Eveniment al oricărui Control de tipul *TextBox* există şi se numeşte "*TextChanged*".

Acest Eveniment este, de altfel, cel implicit al Controlului *TextBox*:



Aşadar:
Pentru a spune butoanelor să devină active când textul din TextBox conţine cel puţin o literă/un caracter:
Dublu-click pe *TextBox*-ul cu textul de prelucrat/investigat:
Caseta de editare a textului pe care am denumit-o "tbtxtinitial:

În modul de scriere a codului, în medoda Evenimentului *TextChanged* a lui "tbtxtinitial" scriem:

```
private void tbtxtinitial_TextChanged(object sender, EventArgs e)
{
        //toate Butoanele vor fi active NUMAI DACA
        //TEXTUL INITIAL FARA A SOCOTI SPATIILE LIBERE/BLANK-URILE
        //NU ESTE NUL:
        //
        bcuvantul6.Enabled =
                bcuvantulfinal.Enabled =
                blistacuvintelor4plus.Enabled =
                bnumarulcuvintelor.Enabled =
                bnumarulspatiilor.Enabled =
                (tbtxtinitial.Text.Trim() != "");   //sau (tbtxtinitial.Text.Trim().Length > 0);
}
```

Ne întoarcem în modul de proiectare/Design al *Form1*, unde solicităm Visual Studio să genereze metoda asociată Evenimentului *Click* pe fiecare buton.

Butonul cu fundal roşu din dreapta-sus:
- a fost denumit "bexit"
- oferă utilizatorului posibilitatea de a confirma închiderea programului după ce răspunde afirmativ la întrebarea dintr-o *MessageBox*
- face inutilă prezenţa/vizibilitatea butonului "X" din dreapta-sus, de închidere a programului/a *Form1*, astfel că am stabilit Proprietatea *ControlBox* a *Form1* la: False

Fiecare buton a fost denumit în acord cu funcţia pe care o îndeplineşte; în metodele următoare, toare asociate Evenimentului *Click* al butonului respectiv, denumirea butonului se afla înainte de particula _Click:

Codul de mai jos este tot codul programului nostru, aflat în *Form1*.
Poate fi copiat (*Copy*) şi plasat (*Paste*) în modul de scriere a codului din Visual Studio
după ce se solicită generarea de către Visual Studio pentru fiecare buton de pe *Form1* a metodei asociate Evenimentului *Click* al acelui buton:

Comentariile explică instrucţiunile:
Citiţi-le cu atenţie:

Atenţie aparte trebuie acordată metodei *Split* care împarte un *String* în secvenţe separate de un caracter-separator: urmăriţi cum a fost folosită:

Split-area unui text după spaţiul liber/blank (caracterul ' ')
consideră secvenţa "\r\n" (CRLF=***C***arriage***R***eturn***L***ine***F***eed=*Enter*) drept ***parte a unui cuvânt***

```
using System;
using System.Linq;
using System.Windows.Forms;

namespace curscs_texte
{
	public partial class Form1 : Form
	{
		public Form1()
		{
			InitializeComponent();
		}

		private void Form1_Load(object sender, EventArgs e)
		{
			//acum Form1 este incarcata:
			//Controalele sunt pregatite si Dialogul este afisat
			//
			//dintru bun inceput stergem textul din caseta tbtxtfinal
			//pentru a ne asigura ca locul este curat si pregatit pentru alt text:
			//
			tbtxtfinal.Clear();
			//
			//acum stabilim textul din caseta tbtxtinitial la afisarea acestei Form-e:
			//
			tbtxtinitial.Text = @"Urmele craterului Chicxulub de la sud
					de Golful Mexic atesta impactul,
					de acum aproximativ 65 milioane de ani,
					cu Pamantul al unui asteroid avand diametrul
					de circa 10 kilometri.

					Evenimentul a cauzat, printre alte dezastre planetare,
					intreruperea lantului trofic util
					marilor reptile (dinozaurienilor, in general)
					si marcheaza inceputul erei mamiferelor.

					900, 800, 700, 600, 500, 400, 300, 200, 100,
					1000, 2000, 3 three one thousand.
						";
		}
```

```
private void bexit_Click(object sender, EventArgs e)
{
	//butonul de inchidere a programului
	//
	//vrem ca utilizatorul sa inchida astfel programul
	//in loc de a apasa pe X-ul din dreapta-sus al Form1/al programului
	//fiindca atunci nu i se va cere confirmare
	//
	if (MessageBox.Show(
				"Gata?",
				"Confirmare",
				MessageBoxButtons.YesNo,
				MessageBoxIcon.Question,
				MessageBoxDefaultButton.Button2) ==
				DialogResult.Yes)
	{
		Application.Exit();
	}
}


private void tbtxtinitial_TextChanged(object sender, EventArgs e)
{
	//toate Butoanele vor fi active NUMAI DACA
	//TEXTUL INITIAL
	//FARA A SOCOTI SPATIILE LIBERE/BLANK-URILE
	//NU ESTE NUL:
	//
	bcuvantul6.Enabled =
		bcuvantulfinal.Enabled =
		blistacuvintelor4plus.Enabled =
		bnumarulcuvintelor.Enabled =
		bnumarulspatiilor.Enabled =
		(tbtxtinitial.Text.Trim() != "");
		//sau (tbtxtinitial.Text.Trim().Length > 0);
}
```

```
        private void bnumarulcuvintelor_Click(object sender, EventArgs e)
        {
            //solicitam numarul cuvintelor din caseta de text tbtxtinitial
            //si odata obtinut il afisam in caseta de text tbtxtfinal
            //
            //stergem textul din caseta tbtxtfinal
            //pentru a ne asigura ca nu va ramane acolo:
            //
            tbtxtfinal.Clear();
            //
            //pentru a NU efectua degeaba operatiunile din continuare
            //verificam daca textul initial poate fi sectionat/impartit in cuvinte distincte
            //separate prin spatiul liber (blank):
            //
            if (tbtxtinitial.Text.IndexOf(" ") > -1)
            {
                int inumarulcuvintelor_dintextul_initial =
                tbtxtinitial.Text.Trim().Split(' ').Length;
                tbtxtfinal.Text = "Din textul initial --sectionat dupa spatiul liber/blank-- pot fi separate:\r\n" +
                inumarulcuvintelor_dintextul_initial.ToString() +
                " cuvinte";
            }
        }


        private void bnumarulspatiilor_Click(object sender, EventArgs e)
        {
            //solicitam numarul spatiilor libere/blank-urilor)
            //din caseta de text tbtxtinitial
            //si odata obtinut il afisam in caseta de text tbtxtfinal
            //
            //stergem textul din caseta tbtxtfinal
            //pentru a ne asigura ca nu va ramane acolo:
            //
            tbtxtfinal.Clear();
            //
            //pentru a NU efectua degeaba operatiunile din continuare
            //verificam daca textul initial contine cel putin un blank:
            //
            if (tbtxtinitial.Text.Contains(" ") == true)
            {
                int inumarulblankurilor_dintextul_initial =
                    tbtxtinitial.Text.Trim().Count(Char.IsSeparator);
                tbtxtfinal.Text = "In textul initial se afla:\r\n" +
                    inumarulblankurilor_dintextul_initial.ToString() +
                    " spatii libere/blank-uri";
            }
        }
```

```
        private void blistacuvintelor4plus_Click(object sender, EventArgs e)
        {
            //solicitam cuvintele mai lungi de 4 caractere din caseta de text tbtxtinitial
            //si odata obtinute le afisam in caseta de text tbtxtfinal
            //
            //stergem textul din caseta tbtxtfinal
            //pentru a ne asigura ca nu va ramane acolo:
            //
            tbtxtfinal.Clear();
            //
            //pentru a NU efectua degeaba operatiunile din continuare
            //verificam daca textul initial contine cel putin un cuvant:
            //
            if (tbtxtinitial.Text.Contains(" ") == true)
            {
                string[ ] sacuvinte_toate = tbtxtinitial.Text.Trim().Split(' ');
                tbtxtfinal.Text = "Cuvintele mai lungi de 6 caractere/litere din textul
initial --sectionat dupa spatiul liber/blank-- sunt:\r\n\r\n";
                for (int i = 0; i < sacuvinte_toate.Length; i++)
                {
                    if (sacuvinte_toate[i].Length > 4)
                    {
                        tbtxtfinal.Text +=
                                                sacuvinte_toate[i] +
                                                "\r\n__________\r\n";
                    }
                }
            }
        }
```

```
        private void bcuvantul6_Click(object sender, EventArgs e)
        {
                //solicitam al saselea - 6 - cuvant din caseta de text tbtxtinitial
                //si odata obtinut il afisam in caseta de text tbtxtfinal
                //
                //stergem textul din caseta tbtxtfinal
                //pentru a ne asigura ca nu va ramane acolo:
                tbtxtfinal.Clear();
                //
                //pentru a NU efectua degeaba operatiunile din continuare
                //verificam daca textul initial are cel putin 6 cuvinte:
                //il sectionam dupa spatiile libere/blank-uri
                //si aflam cat de lung este tabloul astfel rezultat:
                //
                if (tbtxtinitial.Text.Contains(" ") == true)
                {
                        string[ ] sacuvinte_toate = tbtxtinitial.Text.Trim().Split(' ');
                        {
                                //variabila sacuvinte_toate o putem DECLARA iarasi
                                //fiindca este IN EXTERIORUL blocului PE CEL MAI
INALT NIVEL =
                                //adica mai jos de locul unde am declarat-o pentru tot
(probabil tot) codul acestei Clase
                                if (sacuvinte_toate.Length >= 6)
                                {
                                        //ne referim la numarul
                                        //ELEMENTELOR din tablou,
                                        //numarand deci de la 1 incolo
                                        //NU LA INDECSII/POZITIILE lor
                                        //pentru care se numara de la 0 incolo
                                        string scuv6 = sacuvinte_toate[5];
                                        tbtxtfinal.Text =
                                        "Cuvantul #6 din textul initial este:\r\n" + scuv6;
                                }
                        }
                }
        }
```

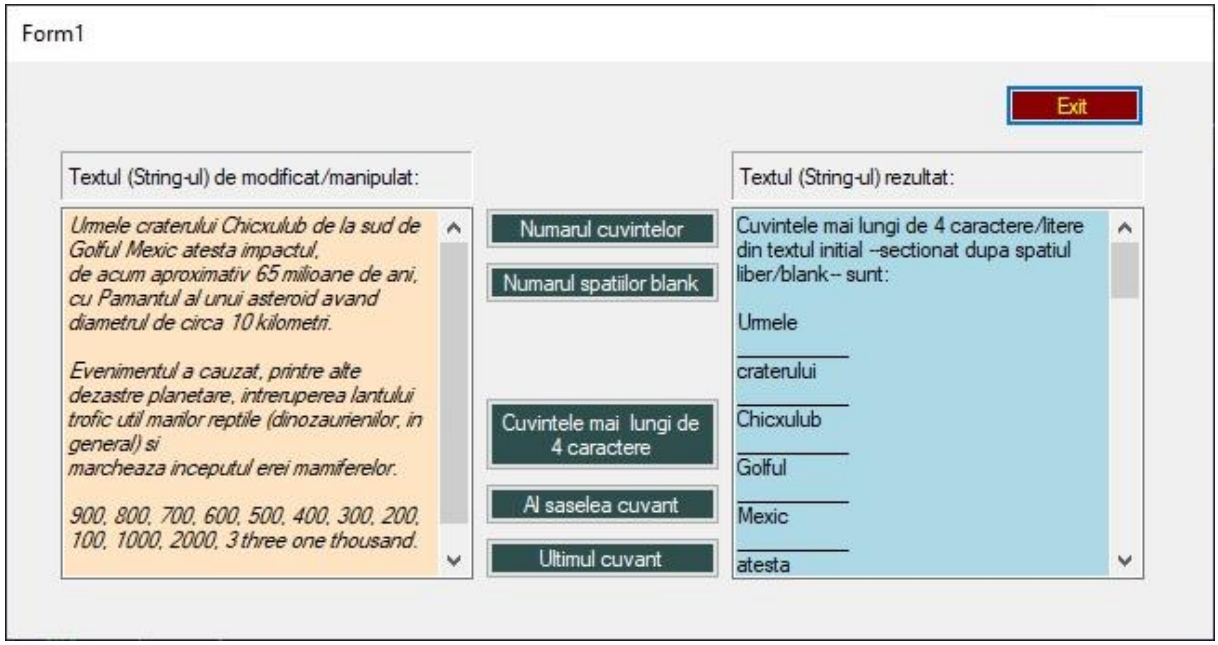
```
        private void bcuvantulfinal_Click(object sender, EventArgs e)
        {
            //solicitam ultimul cuvant din caseta de text tbtxtinitial
            //si odata obtinut il afisam in caseta de text tbtxtfinal
            //
            //stergem textul din caseta tbtxtfinal
            //pentru a ne asigura ca nu va ramane acolo:
            tbtxtfinal.Clear();
            //
            //pentru a NU efectua degeaba operatiunile din continuare
            //verificam daca textul initial are cel putin 1 cuvant:
            //il sectionam dupa spatiile libere/blank-uri si aflam cat de lung este tabloul
astfel rezultat:
            string stxtinitial = tbtxtinitial.Text.Trim();
            if (stxtinitial.Contains(" ") == true)
            {
                int ilastindexofblank = stxtinitial.LastIndexOf(' ');
                string sultimulcuvant = stxtinitial.Substring(ilastindexofblank + 1);
                tbtxtfinal.Text = "Ultimul cuvant din textul initial este:\r\n" +
sultimulcuvant;
            }
        }



    }   //sfarsitul clasei Form1
}   //sfarsitul namespace curscs_texte
```

Pornim programul – tasta F5:

Rezultatul ar trebui să fie asemănător cu imaginea următoare:



Paradoxul logic al Mincinosului (sau al lui Epimenides)

Vom crea un program care ilustrează aşa-numitul Paradox al Mincinosului sau al lui Epimenides (nu că Epimenides ar fi fost neapărat mincinos).

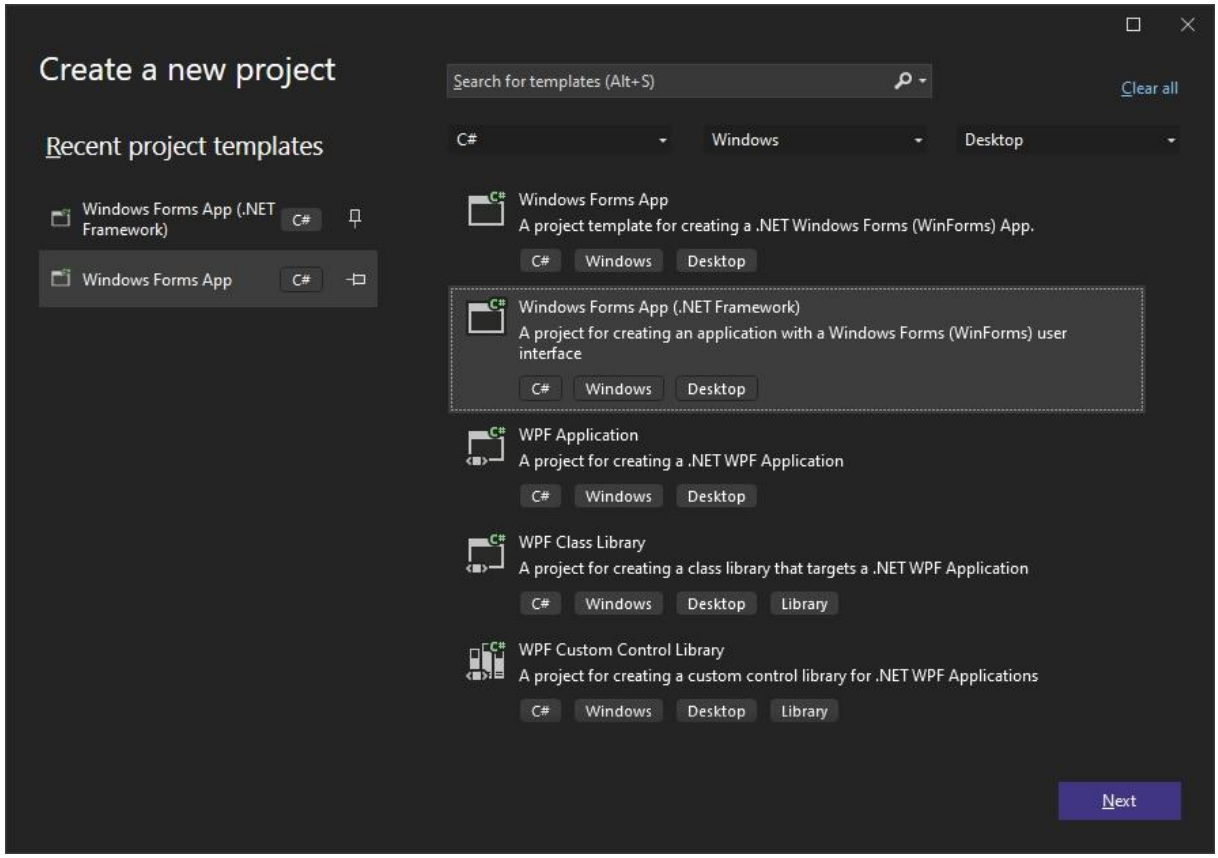
Anume, având la dispoziţie două afirmaţii/propoziţii succesive - *A1* precedând-o pe *A2* –

A1 este: A2 este adevărată
A3 este: A1 este falsă

Sau invers.

Rezultatul este un ciclu infinit în care valoarea de adevăr a unei afirmaţii se schimbă încontinuu la opusul celeilalte şi astfel logica este incertă şi de altfel distrusă.

Să începem.
Lansăm Visual Studio, *File -> New... Project...*



Până acum ştim cum să proiectăm o interfaţă adecvată scopului (rostului) programului nostru, aşa că pe Dialogul *Form1* plasăm câteva etichete *Label* şi un Button care de fiecare dată când va fi apăsat (Evenimentul *Click*, despre care de asemenea ştim deja că este cel implicit al unui Control de tipul Button) va schimba valoarea afirmaţiilor.

Pe etichete scriem strictul necesar.
Nu încărcăm interfaţa/un Dialog (o *Form*-ă) cu litanii, ci etichetele servesc la:

- indicarea unor operaţiuni care trebuie efectuate de către utilizator (deci, mesaje concise destinate acestuia)
- marcarea unor porţiuni/regiuni (mai) importante (decât altele) ale interfeţei/Dialogului

Interfaţa ar putea arăta astfel:



Denumirile Controalelor pot fi extrase din codul care urmează şi care poate fi *Copy*-at şi plasat *Paste*) în codul programului nostru:
Astfel:

- butonul albastru din dreapta se numeşte "bchange1"
- eticheta violet din dreapta-jos se numeşte " lab_a1a2_acum"

Înainte de a copia şi plasa în Visual Studio codul care urmează:

- în modul de proiectare a interfeţei: dublu-click pe "bchange1" pentru ca Visual Studio să genereze metoda asociată Evenimentului *Click* al acestui buton

După ce s-a făcut asta se poate copia şi transfera codul din continuare
ştergându-l pe cel existent în modul de scriere a codului:

```
using System;
using System.Windows.Forms;

namespace cs_liarparadox
{
	public partial class Form1 : Form
	{

		bool b_a1 = true;   //afirmatia A1
		bool b_a2 = true;   //afirmatia A2

		public Form1()
		{
			InitializeComponent();
		}

		private void Form1_Load(object sender, EventArgs e)
		{
			lab_a1a2_acum.Text =
				"Initial:\r\n\r\nA1 = adevarata =>\r\nA2 = adevarata";
		}

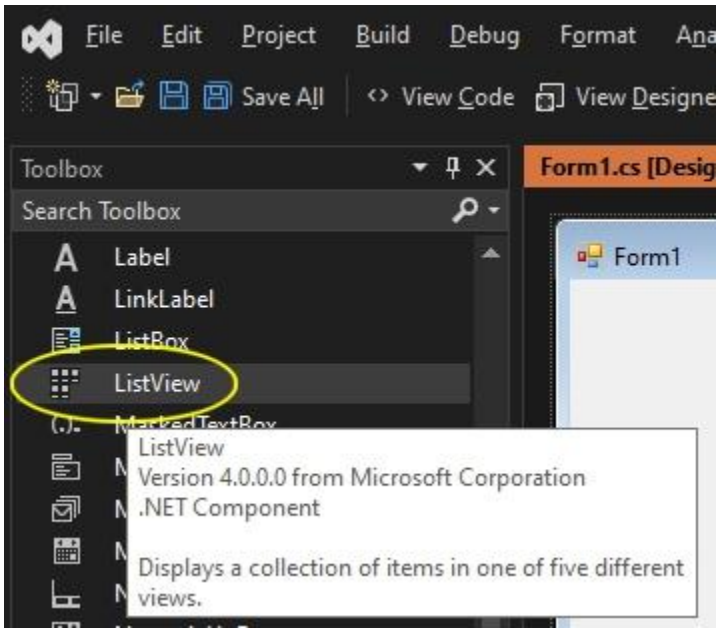
		private void bchange1_Click(object sender, EventArgs e)
		{
			//initial:
			//A1 : A2 = falsa
			//
			//logica ruinata cu fiecare click pe buton
			//
			if (b_a1 == true)
			{
				b_a1 = false;
				lab_a1a2_acum.Text =
					"Acum:\r\nA2 = adevarata =>\r\nA1 = falsa";
			}
			else if (b_a1 == false)
			{
				b_a2 = false;
				b_a1 = true;
				lab_a1a2_acum.Text =
					"Acum:\r\nA1 = adevarata =>\r\nA2 = falsa";
			}
		}

	}  //sfarsitul clasei Form1
}  //sfarsitul namespace cs_liarparadox
```

Porniţi programul – tasta F5. Apăsaţi butonul şi urmăriţi cum nu se mai ştie despre ce e vorba în cele două propoziţii. Distractiv, dar nu e sănătos să se acorde prea mult timp de gândire la ce se întâmplă.

Controlul *ListView*

Aici este prezentată funcţionalitatea de bază şi de fapt cea mai utilă a Controlului ***ListView*** în majoritatea programelor unde se foloseşte acesta.



ListView oferă programatorului şi utilizatorului posibilitatea ca, într-un tabel cu informaţii-text (deci, *String*-uri):

- să afişeze simultan numeroase valori numerice, mesaje şi într-adevăr diverse texte
- să ordone tabelul după valorile dintr-o anumită coloană a acestuia
- să aprecieze dintr-o privire caracteristici fie şi ale unor părţi ale ansamblului prezentat

Informaţia din acest Control poate fi afişată în mai multe feluri, de exemplu ca perechi de (text şi imagine),

Teoretic, un tabel de prezentat într-o *ListView* este proiectat încât să conţină – la fel ca un tabel obişnuit –
informaţii structurate în colecţii asemănătoare şi unice
asociate unui anumit concept sau unei particule anume de informaţie

Fiecare linie din tabel va fi prezentată ca o "înregistrare" distinctă,
dar posibil duplicată oriunde în tabel.

De exemplu, având la dispoziţie o listă alcătuită din elemente cu caracteristici identice într-o *ListView* pot fi prezentate simultan:

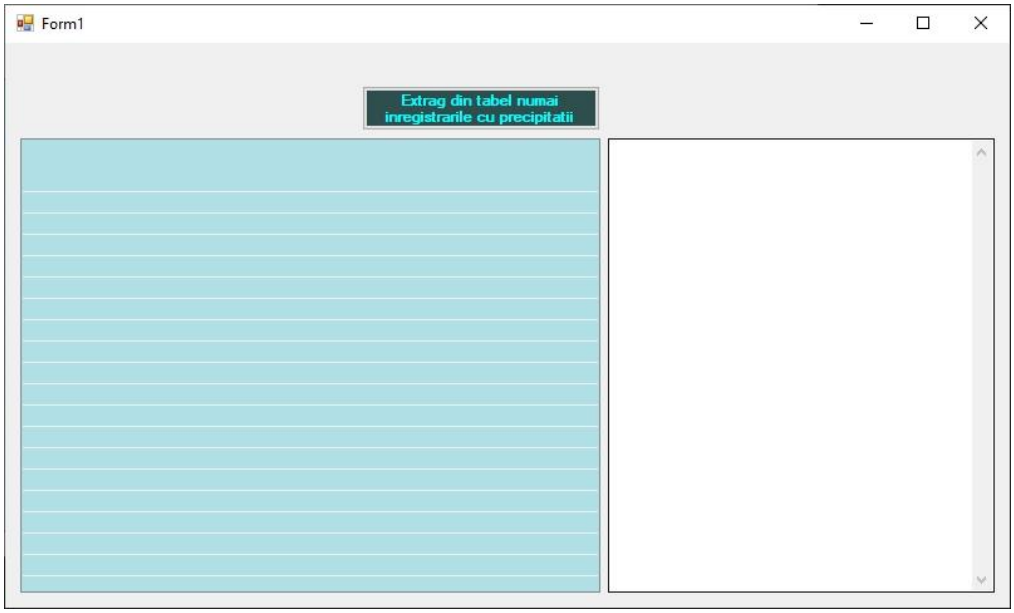
- pentru persoane: numele, prenumele, adresa, vârsta ş. a.
- pentru o localitate: altitudinea, latitudinea, longitudinea, numărul locuitorilor ş. a.
- pentru un număr: tipul său (întreg, raţional ş. a.), semnificaţia pentru ceva anume ş. a.
- pentru o emisiune radiofonică: postul de radio, ora de începere, numele moderatorului ş. a.

În acest exemplu de program unde se foloseşte o *ListView*:
- aceasta va fi populată cu:
  - numele unor regiuni
  - temperatura maximă (fără detalii, ca data observaţiei care poate fi pe o coloană distinctă)
  - temperatura minimă (idem)
  - tipul precipitaţiilor - dacă în regiune au fost precipitaţii şi cu textul nul ("") - dacă nu
  - cantitatea de precipitaţii - dacă este cazul şi cu textul nul ("") - dacă nu
- vor fi extrase şi prezentate, într-un *TextBox*, înregistrările care conţin informaţii despre precipitaţii

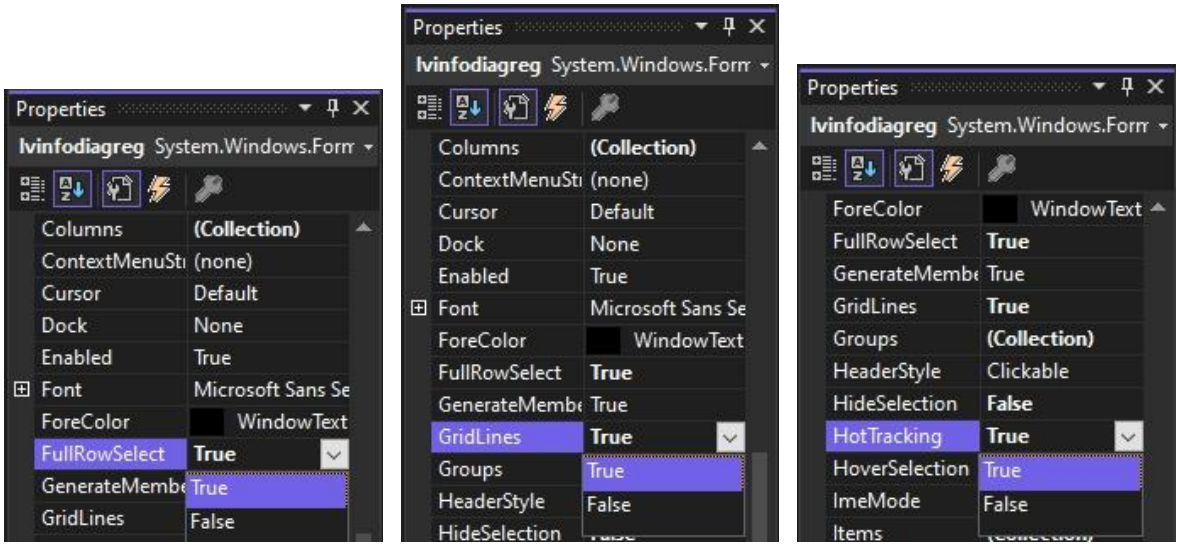
Să începem.
Pornim Visual Studio, *File -> New... Project...*
(deja ştim ce să selectăm din tipurile de aplicaţii/programe propuse; v. şi imaginea de la începutul exemplului care îl precedă pe acesta).

Pe *Form1* plasăm aceste Controale:
- o *ListView* pe care o denumim "lvinfodiagreg" (**L**ist**V**iew-**info**rmaţii-**diag**noză-**reg**iuni)
- un *Button* pe care pentru ca atunci când citim propriul cod să ştim la ce se referă *Buttonul* îl redenumim cu un nume sugestiv: "bpoplv" (**b**utton-**pop**ulare-**L**ist**V**iew); acesta va răspunde la Evenimentul său *Click* cu popularea "lvinfodiagreg"
- un *TextBox* pe care îl denumim "tbregcuprecip" (**T**ext**B**ox-**reg**iuni-**cu**-**precip**itaţii) unde vor fi afişate înregistrările din "lvinfodiagreg" şi unde se află informaţii despre precipitaţii

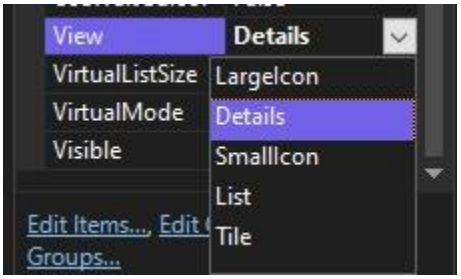
Interfaţa ar putea arăta astfel:
După ce 1) redimensionăm, 2) aliniem, 3) colorăm Controalele, eventual stabilim şi Proprietatea Cursor a fiecăruia până când considerăm că interfaţa este proiectată logic inclusiv pentru utilizator şi că are aspect cât de cât decent (o putem schimba oricând până trimitem programul la utilizator):



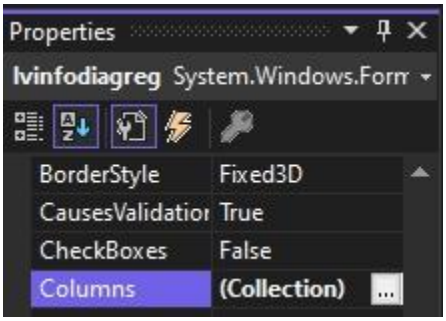
Stabilim Proprietăţi ale *ListView* încât să aibă un aspect prietenos:



Pentru cazul în care vrem ca informaţia să fie prezentată în format tabelar, Proprietatea "***View***" trebuie stabilită la "Details":
Această valoare semnifică faptul că vor exista coloane şi linii, pe fiecare linie urmând a fi prezentată o înregistrare despre un element:

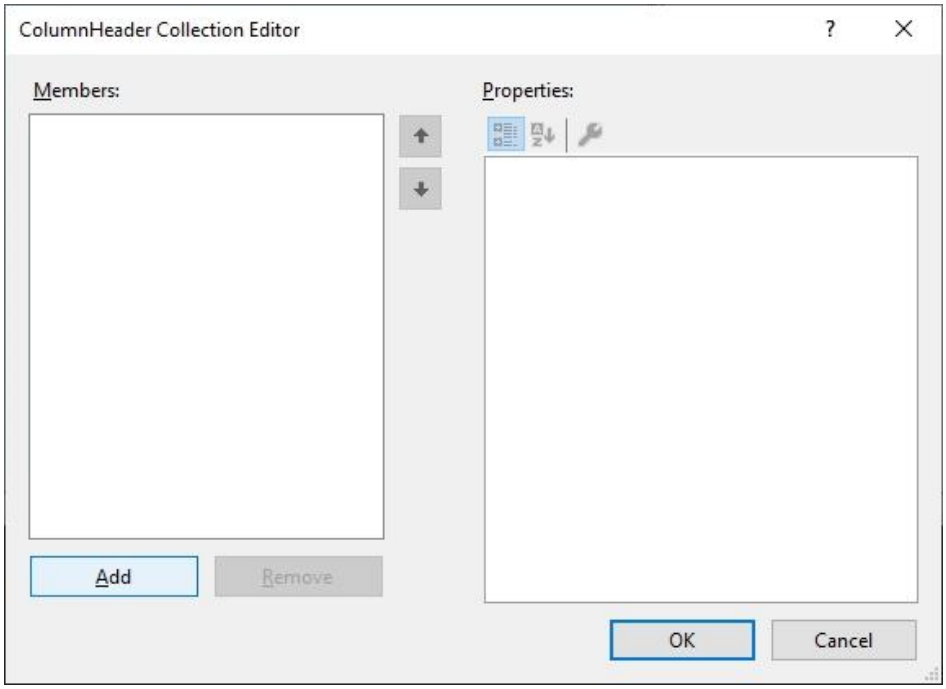


Întrucât ştim ce anume caracteristici ale unui element vor fi afişate în tabel:
stabilim Proprietatea "***Columns*** **- *în general asociată (sau care decurge din Proprietatea) "View"*** –

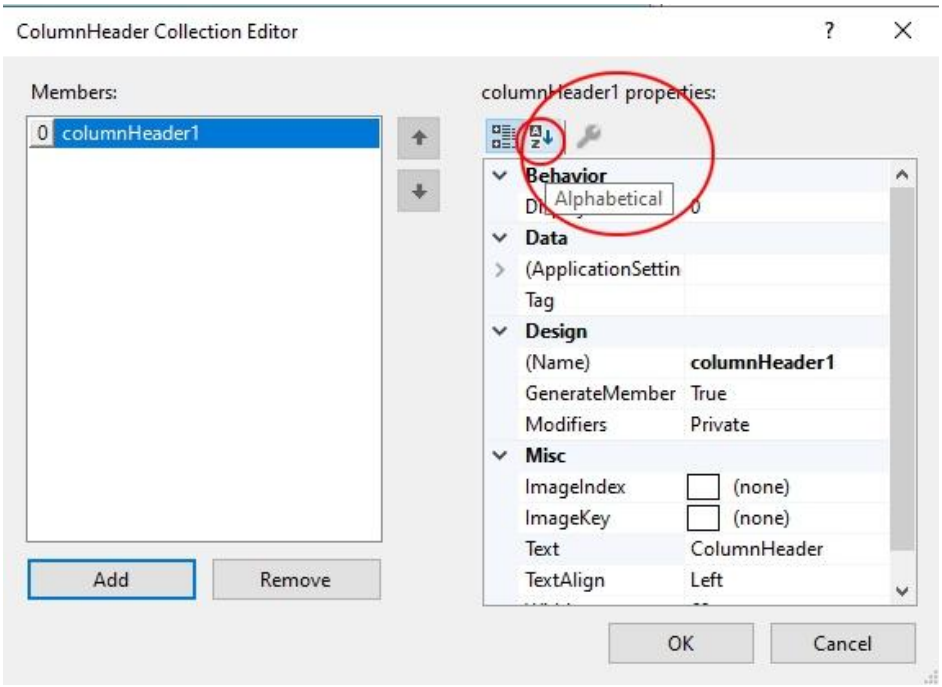


Click-stânga pe cele trei puncte:
În dialogul care apare specificăm numele coloanelor corespunzătoare caracteristicilor fiecărui element:

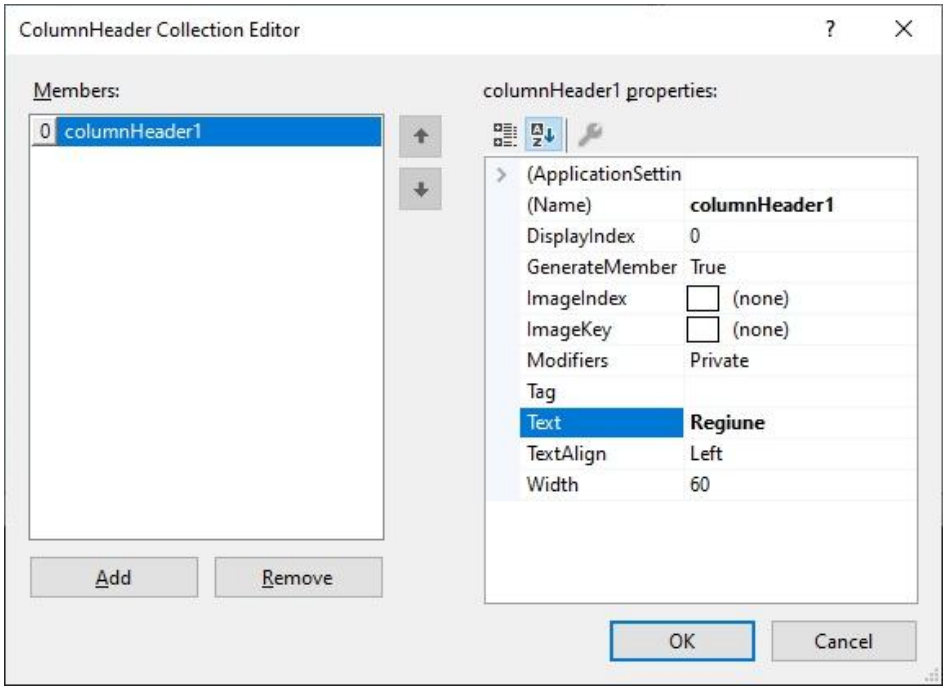
- se apasă butonul "Add"



- sortăm alfabetic coloanei pe care tocmai am propus-o, pentru a le accesa mai rapid – apăsând pe pictograma ("A-Z") încercuită în secundar din imaginea următoare:

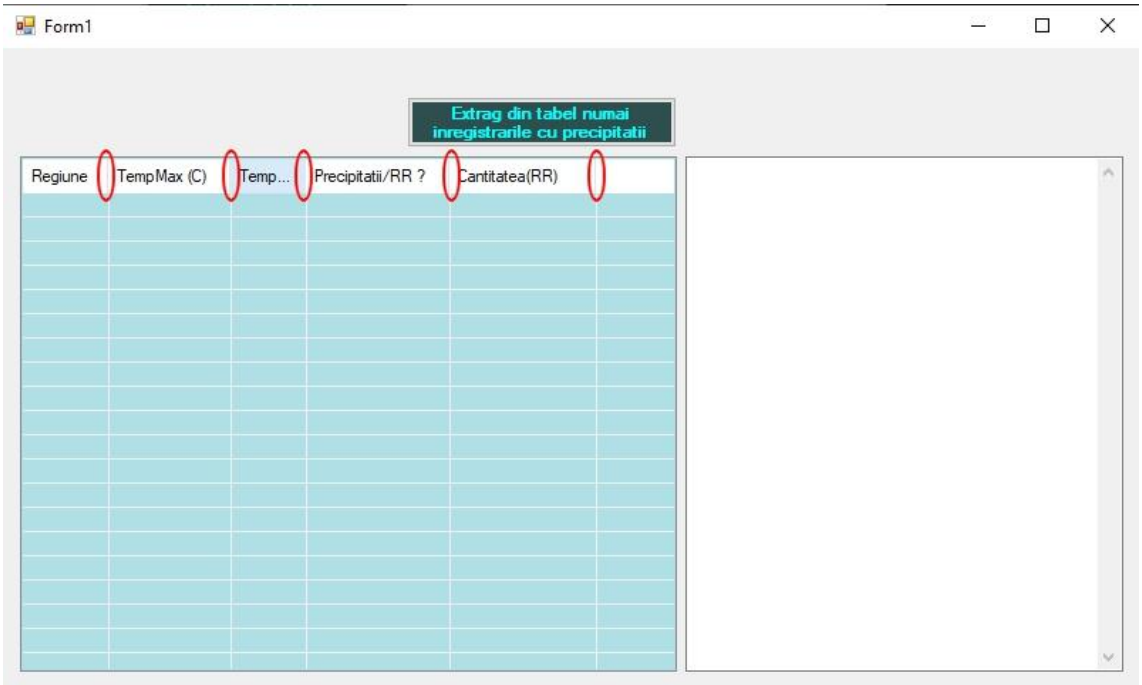


- denumim prima coloană "Regiune". **Acesta este textul afişat în dreptul** fiecărei înregistrări:



- putem modifica Proprietatea "Width" – lăţimea în pixeli *a oricărei* coloane încât să fie adecvată celui mai lung text pe care îl va conţine; de asemenea, aceasta se poate face în cod şi pentru toate coloanele odată. Dacă totuşi ţinem a stabili manual, aici, atunci putem aprecia lăţimea fiecărei coloane inclusiv în funcţie de dimensiunea (Proprietatea "Size" a întregii *ListView* -> "Width")
- după ce am pus un text pe această coloană apăsăm iarăşi butonul "Add" pentru a adăuga la tabel restul coloanelor (aici, pentru caracteristicile fiecărei Regiuni); stabilim un Text pentru fiecare coloană, la fel cum am făcut adineauri cu "Regiune"
- când terminăm adăugarea coloanelor şi redenumirea acestor particule-cap de tabel:
- apăsăm "OK" şi interfaţa se va schimba în consecinţă:

Se observă cum lăţimea fiecărei coloane poate fi modificată trăgând de marginea din dreapta a porţiunii de antet care conţine textul/titlul acelei coloane:

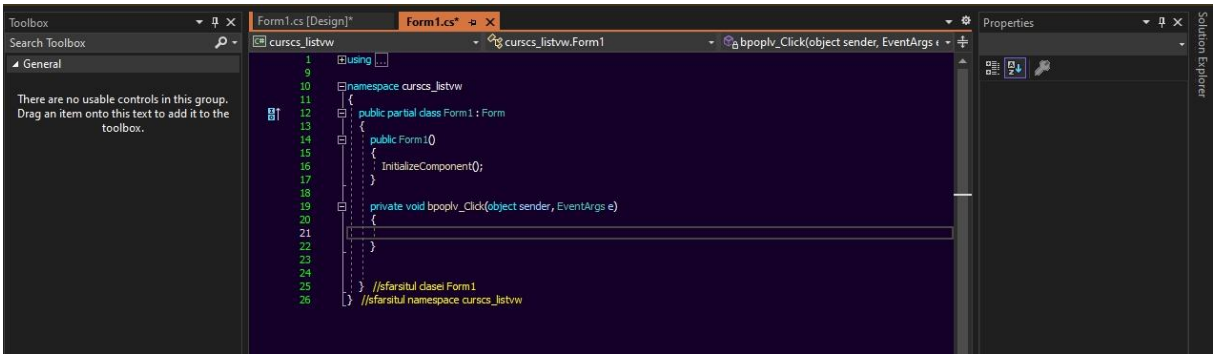


Deci tocmai am pregătit Controlul *ListView* pentru a susţine elementele şi caracteristicile lor pe care i le vom trimite.

Aşa că începem a scrie codul asociat Controalelor:

Să trimitem nişte înregistrări în *ListView*:

Dublu-click pe buton, ceea ce desigur ne trimite în modul de scriere a codului, în metoda asociată Evenimentului implicit *Click* al oricărui Button:



Un Control de tipul *ListView* conţine obiecte de tipul *ListViewItem* (element-de-ListView).

Astfel, pentru a adăuga înreistrări în tabelul de afişat în *ListView*:
- creăm astfel de obiecte *ListViewItem*
- le trimitem pe rând în *ListView*

Scriem astfel:

```
private void bpoplv_Click(object sender, EventArgs e)
{
        if (lvinfodiagreg.Items.Count > 0)
        {
                lvinfodiagreg.Items.Clear();
        }
        ListViewItem lvi = new ListViewItem();
        lvi.Text = "Banat";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Crisana";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Dobrogea";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Moldova";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Muntenia";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Oltenia";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Transilvania-Nord";
        lvinfodiagreg.Items.Add(lvi);
        lvi = new ListViewItem();
        lvi.Text = "Transilvania-Sud";
        lvinfodiagreg.Items.Add(lvi);
}
```

Citim instrucţiunile de mai sus:

if (lvinfodiagreg.Items.Count > 0)
Dacă **acum** în "lvinfodiagreg" există cel puţin un element, atunci:

lvinfodiagreg.Items.Clear();
Ştergem toate elementele afişate **acum** din "lvinfodiagreg"

ListViewItem lvi = new ListViewItem();
Creăm un element de afişat în "lvinfodiagreg"

lvi.Text = "Banat";
Am spectficat care va fi **textul indicator, afişat** al acestui element;
Acesta (proprietatea *.Text*) este textul din prima coloană a unei înregistrări

lvinfodiagreg.Items.Add(lvi);
Trimitem acest element în "lvinfodiagreg"

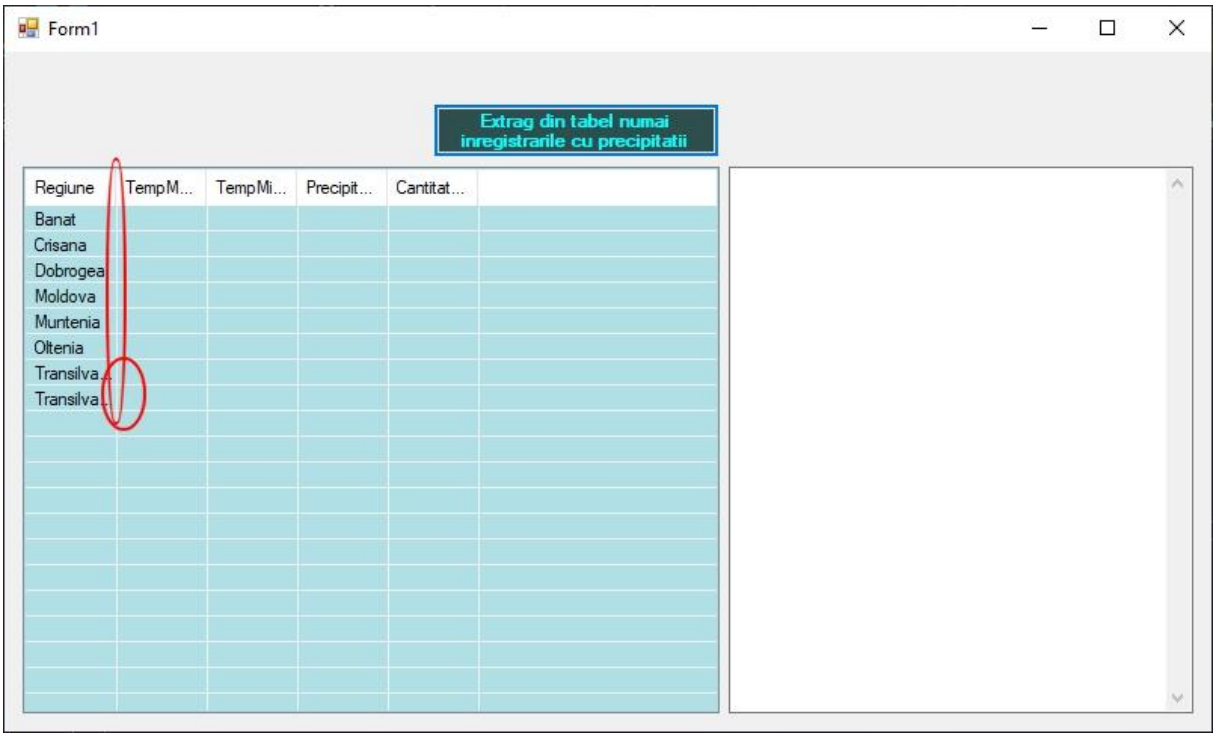
lvi = new ListViewItem();
lvi.Text = "Crisana";

[...] şamd.

Repetăm operaţiunile din cazul elementului precedent.
Se observă cum **nu declarăm iar şi iar obiectul "lvi" de tipul *ListViewItem*,**
fiindcă l-am declarat la început, când am iniţializat primul element de trimis în *ListView* –
adică l-am declarat pe acelaşi nivel cu obiectele de acelaşi tip.

Să verificăm codul de până acum: Pornim programul: Apăsăm tasta F5:

Apăsăm butonul şi Controlul nostru de tip *ListView* având Proprietatea "View" stabilită la "Details" încât să prezinte informaţia conţinută în format tabelar va arăta astfel:



Se vede cum numele unor regiuni nu sunt afişate complet: putem trage de marginea din dreapta a porţiunii de antet/cap de tabel corespunzătoare acelei coloane până ce numele regiunii este afişat integral.

Închidem programul ("X" din dreapta-sus al acestui Dialog/*Form1*) şi ne întoarcem la cod:
unde începem a scrie caractreisticile fiecărei Regiuni:
pe care le vom trimite în *ListView*:

Pentru aceasta vom completa instrucţiunile precedente cu unele care construiesc fiecare înregistrare:

Codul din continuare este cel al metodei bpoplv_Click:
Poate fi copiat (*Copy*) de aici şi plasat (*Paste*) **peste cel existent al aceleiaşi metode**:

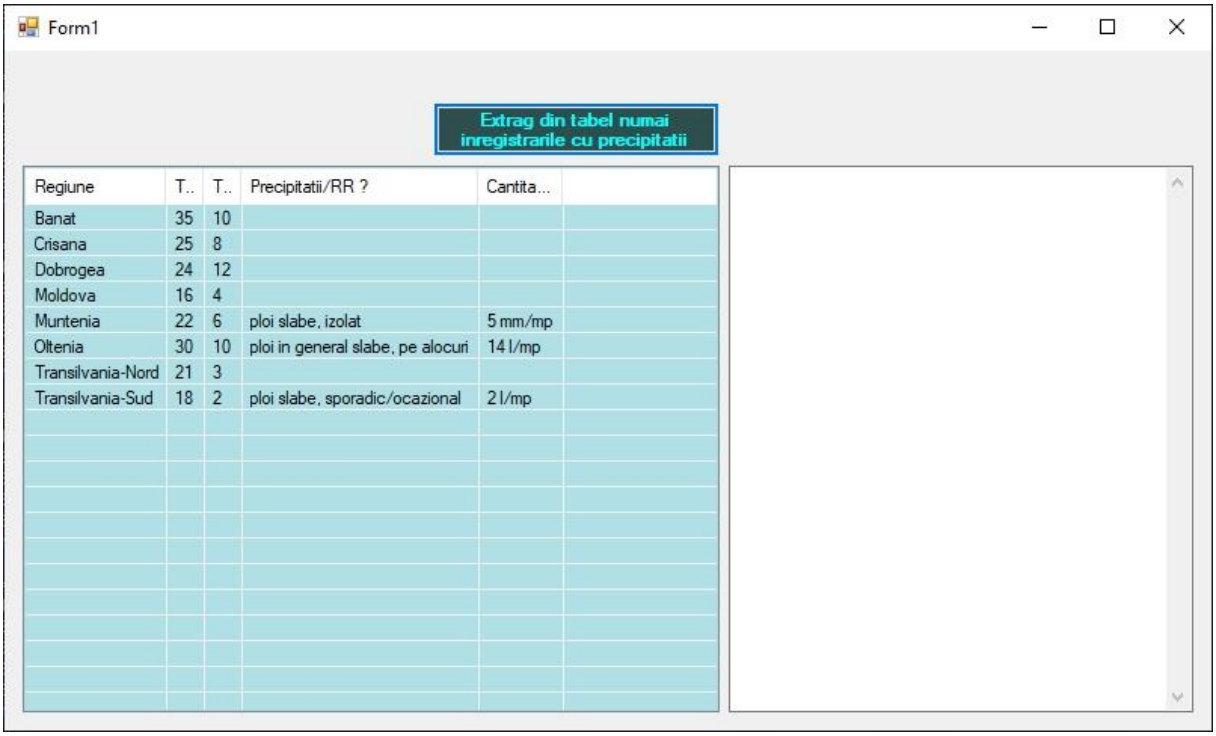
```
private void bpoplv_Click(object sender, EventArgs e)
{
        if (lvinfodiagreg.Items.Count > 0)
        {
                lvinfodiagreg.Items.Clear();
        }
        ListViewItem lvi = new ListViewItem();
        lvi.Text = "Banat";
        lvi.SubItems.Add("35");
        lvi.SubItems.Add("10");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Crisana";
        lvi.SubItems.Add("25");
        lvi.SubItems.Add("8");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Dobrogea";
        lvi.SubItems.Add("24");
        lvi.SubItems.Add("12");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Moldova";
        lvi.SubItems.Add("16");
        lvi.SubItems.Add("4");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Muntenia";
        lvi.SubItems.Add("22");
        lvi.SubItems.Add("6");
        lvi.SubItems.Add("ploi slabe, izolat");
        lvi.SubItems.Add("5 mm/mp");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Oltenia";
        lvi.SubItems.Add("30");
        lvi.SubItems.Add("10");
```

```
            lvi.SubItems.Add("ploi in general slabe, pe alocuri");
            lvi.SubItems.Add("14 l/mp");
            lvinfodiagreg.Items.Add(lvi);
            //
            lvi = new ListViewItem();
            lvi.Text = "Transilvania-Nord";
            lvi.SubItems.Add("21");
            lvi.SubItems.Add("3");
            lvi.SubItems.Add("");
            lvi.SubItems.Add("");
            lvinfodiagreg.Items.Add(lvi);
            //
            lvi = new ListViewItem();
            lvi.Text = "Transilvania-Sud";
            lvi.SubItems.Add("18");
            lvi.SubItems.Add("2");
            lvi.SubItems.Add("ploi slabe, sporadic/ocazional");
            lvi.SubItems.Add("2 l/mp");
            lvinfodiagreg.Items.Add(lvi);
            //
            //redimensionam fiecare coloana a "lvinfodiagreg" incat cel mai lung text
            //de pe oricare inregistrare (din oricare element) a tabelului prezentat
            //sa fie afisat integral:
            for (byte b = 0; b < lvinfodiagreg.Columns.Count; b++)
            {
                  lvinfodiagreg.Columns[b].Width = -1;
            }
}
```

Ultimele două instrucţiuni determină, după cum am şi scris în comentariul care le precedă, redimensionarea orizontală a tuturor coloanelor:
în funcţie de cel mai lung text dintr-un element al /înregistrare a tabelului,
nu de textul/titlul coloanelor

Proprietatea "Width" a acelei coloane din colecţia *Columns* a *ListView* o stabilim la -1 (o unitate mai mică decât zero).

Pornim iarăşi programul:



Se vede cum stabilirea la "-1" pentru fiecare coloană a proprietăţii.Width
nu a redimensionat-o în acord cu titlul coloanei,
ci cu cel mai lung text din acea coloană

Închidem programul.
Acum scriem cod pentru separarea înregistrărilor corespunzătoare acelor regiuni unde au fost precipitaţii:

Sub acolada de după instrucţiunea unde am redimensionat la "-1" coloanele adăugăm acest cod:

```
        //
        //
        //acum separam/extragem din tabel acele inregistrari cu informatii despre precipitatii
nenule:
        //intai ne asiguram ca in TextBox-ul unde vom trimite acele inregistrari
        //nu exista niciun text: stergem textul din TextBox:
        tbregcuprecip.Clear();
        //
        //acum cautam acele inregistrari si le trimitem in TextBox:
        for (int i = 0; i < lvinfodiagreg.Items.Count; i++)
        {
                ListViewItem lviregcuprecip = lvinfodiagreg.Items[i];
                if (lviregcuprecip.SubItems[3].Text.Trim() != "")
                {
                        tbregcuprecip.Text +=
                                "Regiune:" + lviregcuprecip.Text + "\r\n" +
                                "T_Maxima:" + lviregcuprecip.SubItems[1].Text + "\r\n" +
                                "T_minima:" + lviregcuprecip.SubItems[2].Text + "\r\n" +
                                "Precipitatii:" + lviregcuprecip.SubItems[3].Text + "\r\n" +
                                "Cantitatea maxima de apa din RR:" +
                                        lviregcuprecip.SubItems[4].Text + "\r\n" + "\r\n";
                                }
                        }
```

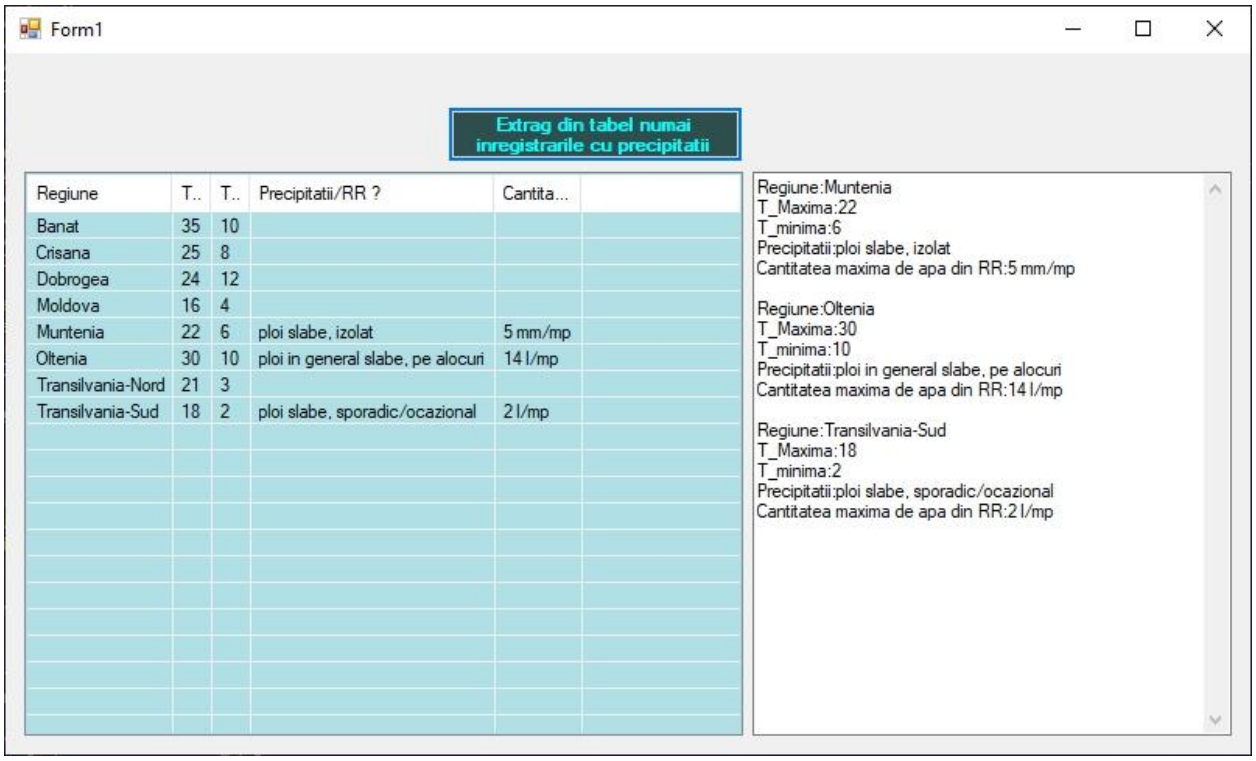
După ultima acoladă din acest cod se află cea care închide metoda **bpoplv_Click**.

Codul complet al programului se află după imaginea următoare.

Nu copiați și plasați în program codul de aici înainte
de a solicita cu dublu-click pe Button-ul "bpoplv"
crearea metodei asociate Evenimentului *Click* al acestuia

Pornim iarăşi programul: tasta F5:

Apăsăm butonul:
Rezultatul:



Codul complet al metodei bpoplv_Click începe pe pagina următoare:
Bineînţeles, liniile goale pot fi şterse:

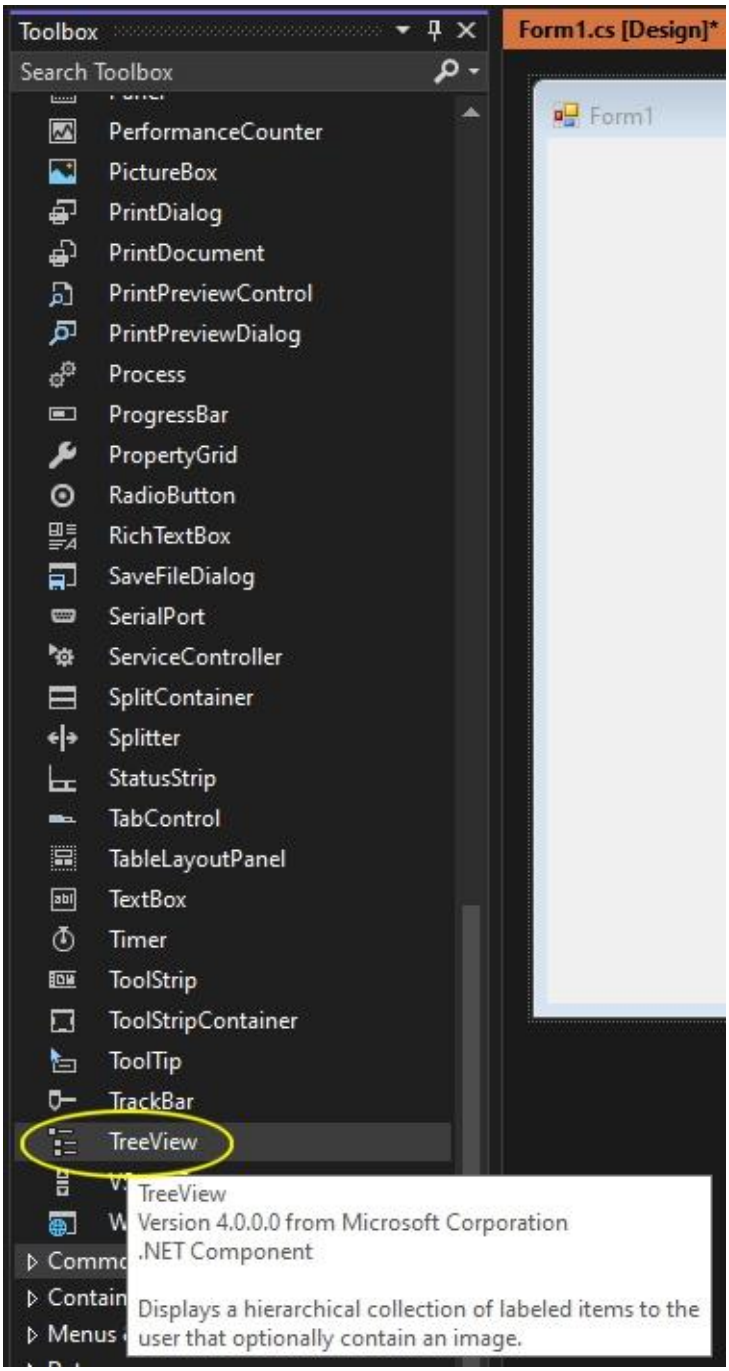
```
private void bpoplv_Click(object sender, EventArgs e)
{
        if (lvinfodiagreg.Items.Count > 0)
        {
        lvinfodiagreg.Items.Clear();
        }
        ListViewItem lvi = new ListViewItem();
        lvi.Text = "Banat";
        lvi.SubItems.Add("35");
        lvi.SubItems.Add("10");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Crisana";
        lvi.SubItems.Add("25");
        lvi.SubItems.Add("8");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Dobrogea";
        lvi.SubItems.Add("24");
        lvi.SubItems.Add("12");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Moldova";
        lvi.SubItems.Add("16");
        lvi.SubItems.Add("4");
        lvi.SubItems.Add("");
        lvi.SubItems.Add("");
        lvinfodiagreg.Items.Add(lvi);
        //
        lvi = new ListViewItem();
        lvi.Text = "Muntenia";
        lvi.SubItems.Add("22");
        lvi.SubItems.Add("6");
        lvi.SubItems.Add("ploi slabe, izolat");
        lvi.SubItems.Add("5 mm/mp");
        lvinfodiagreg.Items.Add(lvi);
        //
```

```
lvi = new ListViewItem();
lvi.Text = "Oltenia";
lvi.SubItems.Add("30");
lvi.SubItems.Add("10");
lvi.SubItems.Add("ploi in general slabe, pe alocuri");
lvi.SubItems.Add("14 l/mp");
lvinfodiagreg.Items.Add(lvi);
//
lvi = new ListViewItem();
lvi.Text = "Transilvania-Nord";
lvi.SubItems.Add("21");
lvi.SubItems.Add("3");
lvi.SubItems.Add("");
lvi.SubItems.Add("");
lvinfodiagreg.Items.Add(lvi);
//
lvi = new ListViewItem();
lvi.Text = "Transilvania-Sud";
lvi.SubItems.Add("18");
lvi.SubItems.Add("2");
lvi.SubItems.Add("ploi slabe, sporadic/ocazional");
lvi.SubItems.Add("2 l/mp");
lvinfodiagreg.Items.Add(lvi);
//
//redimensionam fiecare coloana a "lvinfodiagreg" incat cel mai lung text
//de pe oricare inregistrare (din oricare element) a tabelului prezentat
//sa fie afisat integral:
for (byte b = 0; b < lvinfodiagreg.Columns.Count; b++)
{
        lvinfodiagreg.Columns[b].Width = -1;
}
//
//
//acum separam/extragem din tabel acele inregistrari
//cu informatii despre precipitatii nenule:
//
//intai ne asiguram ca in TextBox-ul unde vom trimite acele inregistrari
//nu exista niciun text: stergem textul din TextBox:
tbregcuprecip.Clear();
//
```

```
        //acum cautam acele inregistrari si le trimitem in TextBox:
        for (int i = 0; i < lvinfodiagreg.Items.Count; i++)
        {
                ListViewItem lviregcuprecip = lvinfodiagreg.Items[i];
                if (lviregcuprecip.SubItems[3].Text.Trim() != "")
                {
                        //deci, daca in a patra coloana (pozitia/indexul 3)
                        //exista un text nenul, de exemplu "ploi izolate", atunci
                        //preluam acea inregistrare/descriere:
                        //
                        //la descrierile existente in TextBox
                        //adaugam cu operatorul += noua descriere
                        //
                        tbregcuprecip.Text +=
                                "Regiune:" + lviregcuprecip.Text + "\r\n" +
                                "T_Maxima:" + lviregcuprecip.SubItems[1].Text + "\r\n" +
                                "T_minima:" + lviregcuprecip.SubItems[2].Text + "\r\n" +
                                "Precipitatii:" + lviregcuprecip.SubItems[3].Text + "\r\n" +
                                "Cantitatea maxima de apa din RR:" +
                                        lviregcuprecip.SubItems[4].Text + "\r\n" + "\r\n";
                        //
                        //dupa fiecare astfel de descriere
                        // punem un rand liber: "\r\n" + "\r\n"
                        //(secventa de caractere echivalenta cu doua Enter-uri succesive
                        //la sfarsitul unui rand)
                }
        }
}
```

Controlul TreeView

Aici este prezentată funcţionalitatea de bază şi de fapt cea mai utilă a Controlului *TreeView* în majoritatea programelor unde se foloseşte acesta.



***TreeView* este util când se intenţionează afişarea**
structurată după anumite criterii
a unor informaţii de *diverse feluri*

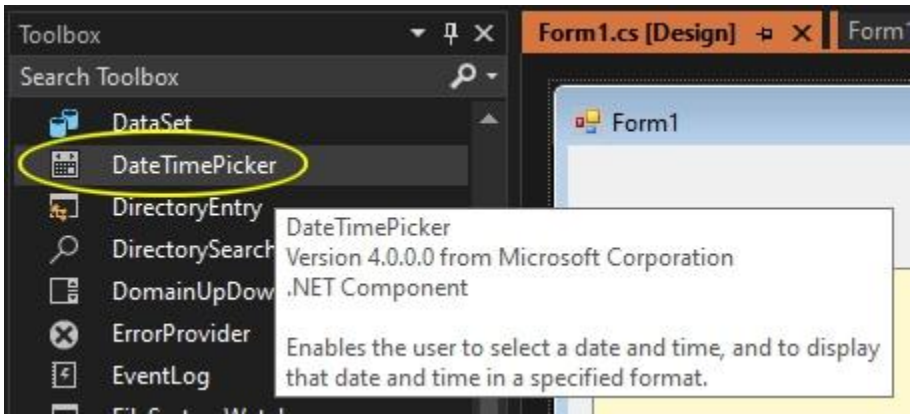
În aceasta, *TreeView* diferă de *ListView* prin aceea că
numeroase *informaţii de diverse feluri* pot fi grupate ramificat –
întocmai ca un arbore cu rădăcină/trunchi, ramuri şi noduri –
după o *caracteristică a lor comună*.

De exemplu:
- rădăcina sau trunchiul se poate numi "Băuturi"
- o ramură se poate numi "Sucuri de fructe" şi poate avea mai multe Noduri derivate din/situate pe acea ramură numite, de exemplu, "Suc de cireşe", "Suc de portocale"
- o ramură se poate numi "Spirtoase" şi poate avea mai multe Noduri derivate din/situate pe acea ramură numite, de exemplu, "Scotch", "Champagne"
- şamd

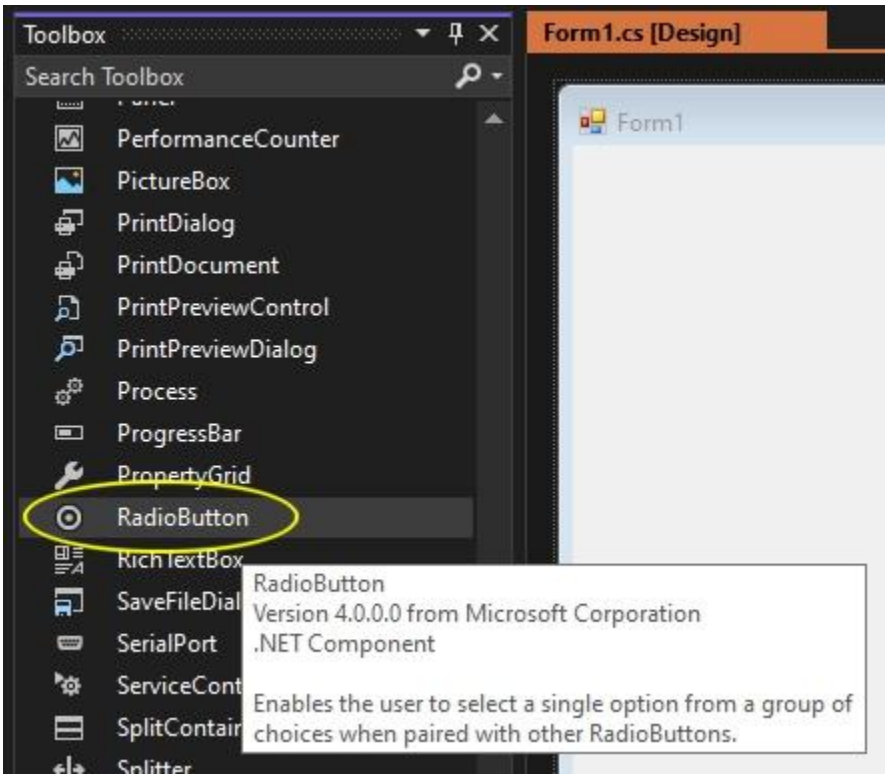
Vom construi un program în care la un TreeView vom adăuga:
- tipurile/felurile de tură la serviciu
- datele calendaristice ale unor ture de fiecare fel

Întrucât vom lucra cu date calendaristice, evident că vom avea nevoie de ceva care să ne ajute în selectarea acelor date şi a cărui funcţionalitate relativ complexă (stabilirea unor date, schimbarea lor, alegerea lor ş.a.) să fie deja pregătită, dacă e posibil chiar de mediul de programare Visual Studio.

Un astfel de Control există - anume, *DateTimePicker*:

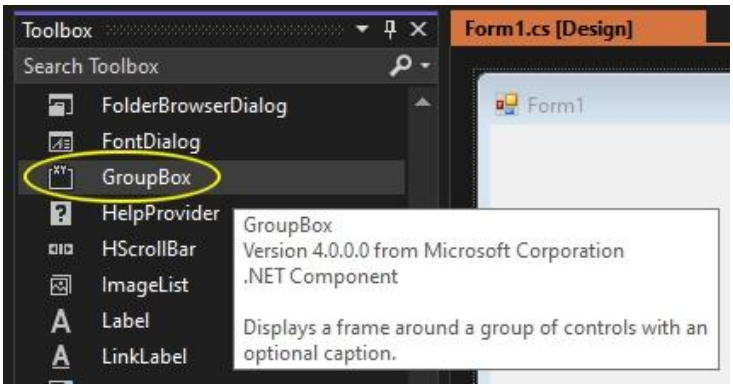


Mai departe, fiindcă vom folosi mai multe tipuri de tură la serviciu, putem presupune că avem la dispoziţie un control care ne asistă în selectarea unui tip anume de obiect, a unei opţiuni din mai multe (deci nu numai două, ca de exemplu *True* şi False).
Un astfel de Control există - anume, *RadioButton*:



Mai departe, întrucât e posibil ca în programul nostru, pe un Dialog (o *Form*-ă) putem folosi
mai multe grupuri de opţiuni multiple –
fiecare grup referindu-se la ceva anume, distinct de celelalte,
putem prespune că există un Control în care într-adevăr pot fi gurpate cumva astfel de Controale oferind opţiuni – deci, cele de tipul *RadioButton*.

Un astfel de Control există, de asemenea – anume, *GroupBox*:



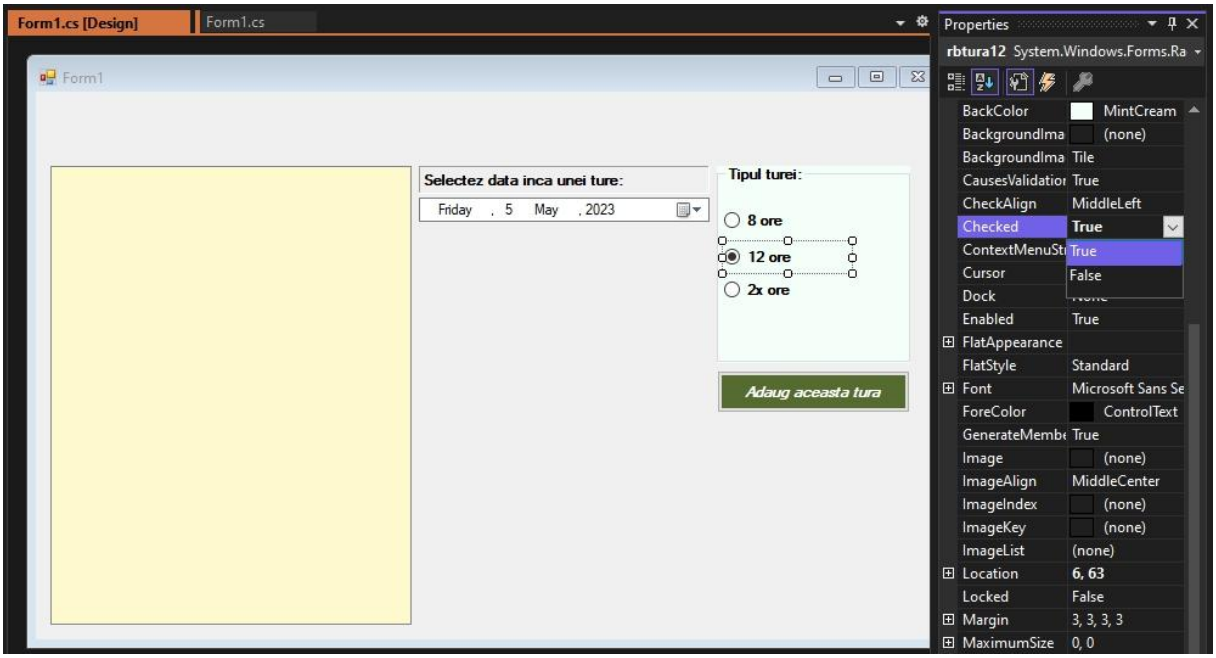
De-acum, când avem în imaginaţie modelul interfeţei, o şi proiectăm:

Din Toolbox tragem pe Form1:
- câte unul din Controalele tocmai prezentate, ***în afară de*** *RadioButton*
- un Button la a cărui apăsare/acţionare (Evenimentul *Click*) la *TreeView* va fi adăugat un Nod (un element)

Apoi, din Toolbox tragem ***în Controlul "groupBox1"***:
- trei-3 Controale *RadioButton*; deci, toate 3 trebuie să fie în interiorul lui "groupBox1"

Desigur, pot fi adăugate etichete (*Label*) care, imediat în apropierea unor Controale, să indice rolul fiecăruia.

După ce 1) redimensionăm, 2) aliniem, eventual 3) colorăm fiecare Control şi stabilim Proprietatea *Text* a fiecăruia care are această Proprietate (inclusiv "groupBox1" al cărui *Text* l-am stabilit la "Tipul turei":
Interfaţa ar putea arăta astfel:



Denumim Controalele astfel:
- *TreeView*-ul: "tvdateleturelor"
- *DateTimePicker*-ul: "dtpdateleturelor"
- *RadioButton*-ul marcat cu textul "8 ore": "rbtura8"
- *RadioButton*-ul marcat cu textul "12 ore": "rbtura12"
- *RadioButton*-ul marcat cu textul "2x ore": "rbtura2x"
- *Button*-ul: "badaugaretura"

În imaginea precedentă se observă cum pentru "rbtura12" am stabilit Proprietatea "Checked" la "True":
Aceasta înseamnă că, la pornirea programului/la afişarea *Form1*, această opţiune va fi bifată.

Ca toate Proprietăţile accesibile în secţiunea Properties din Visual Studio,
această proprietate poate fi stabilită în cod, anume:

rbtura12.Checked = true; //sau false

Să începem a scrie codul programului:

Orice fel de ture am adăuga la *TreeView*-ul "tvdateleturelor", acesta trebuie să aibă dintru bun început trei ramuri corespunzătoare celor trei tipuri de tură.
Astfel că, la afişarea *Form1*, în "tvdateleturelor" adăugăm trei ramuri al căror text va fi (de exemplu);:

- "Tura de 8 ore"
- "Tura de 12 ore"
- "Tura de 2x ore"

Pentru aceasta:

În modul de proiectare/Design al *Form1*:
Dublu-click pe *Form1*:
Aceasta ne trimite în modul de scriere a codului, în metoda generată automat de către Visual Studio asociată Evenimentului *Load* al Form1 – acesta fiind Evenimentul implicit pentru o *Form*-ă.

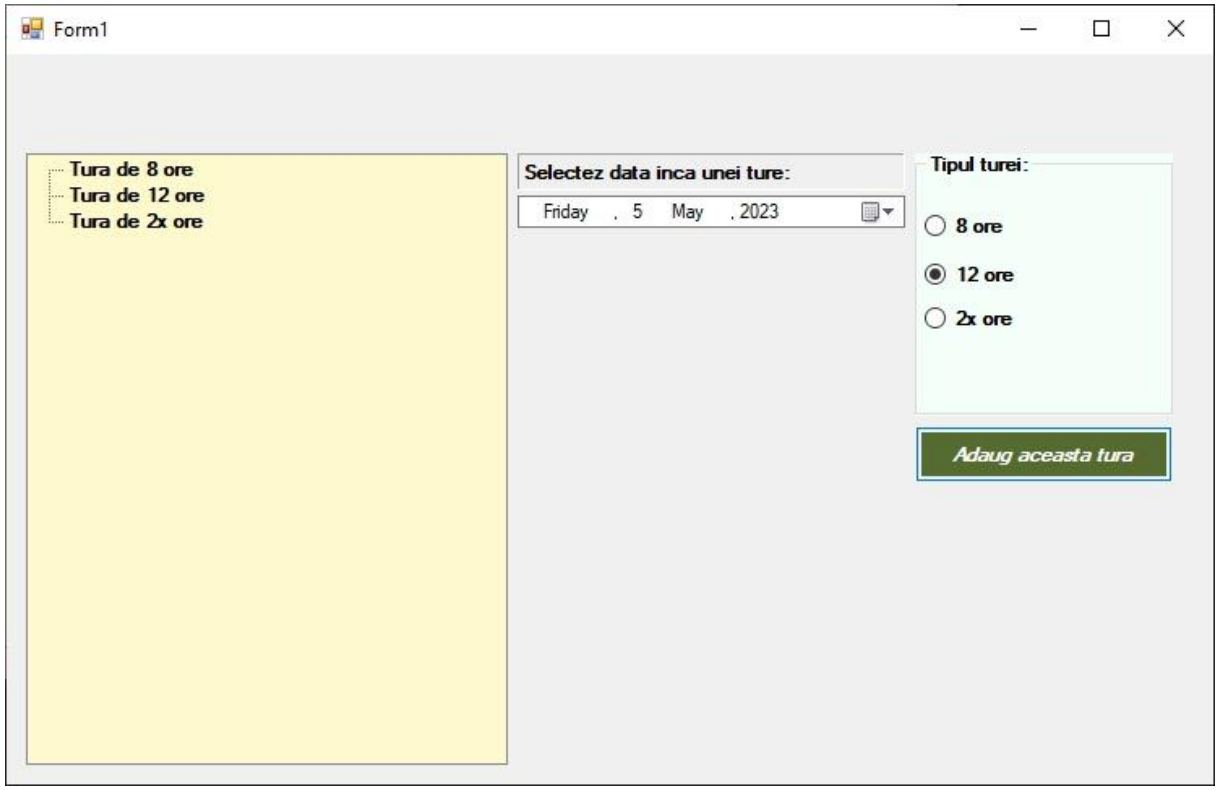
Scriem instrucţiunile pentru adăugarea la "tvdateleturelor" a elementelor menţionate în această metodă:

```
private void Form1_Load(object sender, EventArgs e)
{
        tvdateleturelor.Nodes.Add("Tura de 8 ore");
        tvdateleturelor.Nodes.Add("Tura de 12 ore");
        tvdateleturelor.Nodes.Add("Tura de 2x ore");
}
```

Instrucţiunea aceasta:
tvdateleturelor.Nodes.Add("Tura de 8 ore");
se citeşte astfel:
la colecţia de Noduri (.*Nodes*) a *TreeView*-ului "**tvdateleturelor**"
se adaugă - metoda .*Add* a colecţiei *Nodes*
un element având textul "Tura de 8 ore"

şamd.

Pornim programul – tasta F5:



Deşi cele trei tipuri de ture apar ca elemente/ramuri distincte în *TreeView*-ul nostru, pentru a adăuga elemente/Noduri la fiecare din ele trebuie să le putem identifica mai elegant şi mult mai uşor decât prin parcurgerea întregului arbore şi a verifica textul fiecarei ramuri.

Facem aceasta modificând instrucţiunile de mai sus:
Specificăm *numele* sau *cheia* fiecărei ramuri:

Fiecare element al unui TreeView
poate avea un nume/o cheie prin care poate fi identificat în cod

Metoda astfel schimbată va fi aceasta:

```
private void Form1_Load(object sender, EventArgs e)
{
        tvdateleturelor.Nodes.Add("t8h", "Tura de 8 ore");
        tvdateleturelor.Nodes.Add("t12h", "Tura de 12 ore");
        tvdateleturelor.Nodes.Add("t2xh", "Tura de 2x ore");
}
```

Am atribuit un nume/o cheie fiecărui element adăugat:
De exemplu, elementul marcat "Tura de 8 ore" a primit numele/cheia "t8h".
De acum putem adăuga elemente la cele astfel identificabile, prin numele/cheia lor.

Înapoi în modul de proiectare/Design al *Form1*: tastele Shift+F7.

Să începem a adăuga la *TreeView* ture de diverse feluri, ca Noduri pe/sub ramurile corespunzătoare:

Dublu-click pe butonul "badaugaretura":

În metoda generată automat a Evenimentului *Click* al acestui buton scriem o instrucţiune pentru adăugarea unei ture:

```
private void badaugaretura_Click(object sender, EventArgs e)
{
        TreeNode tn_turanoua =
                new TreeNode(
                        dtpdateleturelor.Value.ToString("dddd, dd MMMM yyyy")
                                );
        if (rbtura8.Checked == true)
        {
                tvdateleturelor.Nodes["t8h"].Nodes.Add(tn_turanoua);
        }
        else if (rbtura12.Checked == true)
        {
                tvdateleturelor.Nodes["t12h"].Nodes.Add(tn_turanoua);
        }
        else if (rbtura2x.Checked == true)
        {
                tvdateleturelor.Nodes["t2xh"].Nodes.Add(tn_turanoua);
        }
}
```

Citim linie cu linie instrucţiunile:

```
TreeNode tn_turanoua =
        new TreeNode(
                dtpdateleturelor.Value.ToString("dddd, dd MMMM yyyy")
                );
```

Creăm un element/Nod ca variabilă de tipul *TreeNode* –
obiect care poate fi adăugat la colecţia *Nodes* a unui *TreeView*)
care va avea textul:
data selectată în Controlul "dtpdateleturelor" de tipul *DateTimePicker*
care dată va fi formatată pentru afişare astfel:
dddd – numele, în cuvinte, al zilei datei selectate – de exemplu, Joi
dd – numărul zilei (de la 1 la 28/29/30/31) din luna datei selectate – de exemplu, 14
MMMM – numele, în cuvinte, al lunii datei selectate – de exemplu, "August"
yyyy – anul datei selectate – de exemplu, 2123

if (rbtura8.Checked == true)
Dacă din grupul de opţiuni asociate Controalelor de tipul *RadioButton* este selectat "rbtura8", atunci:

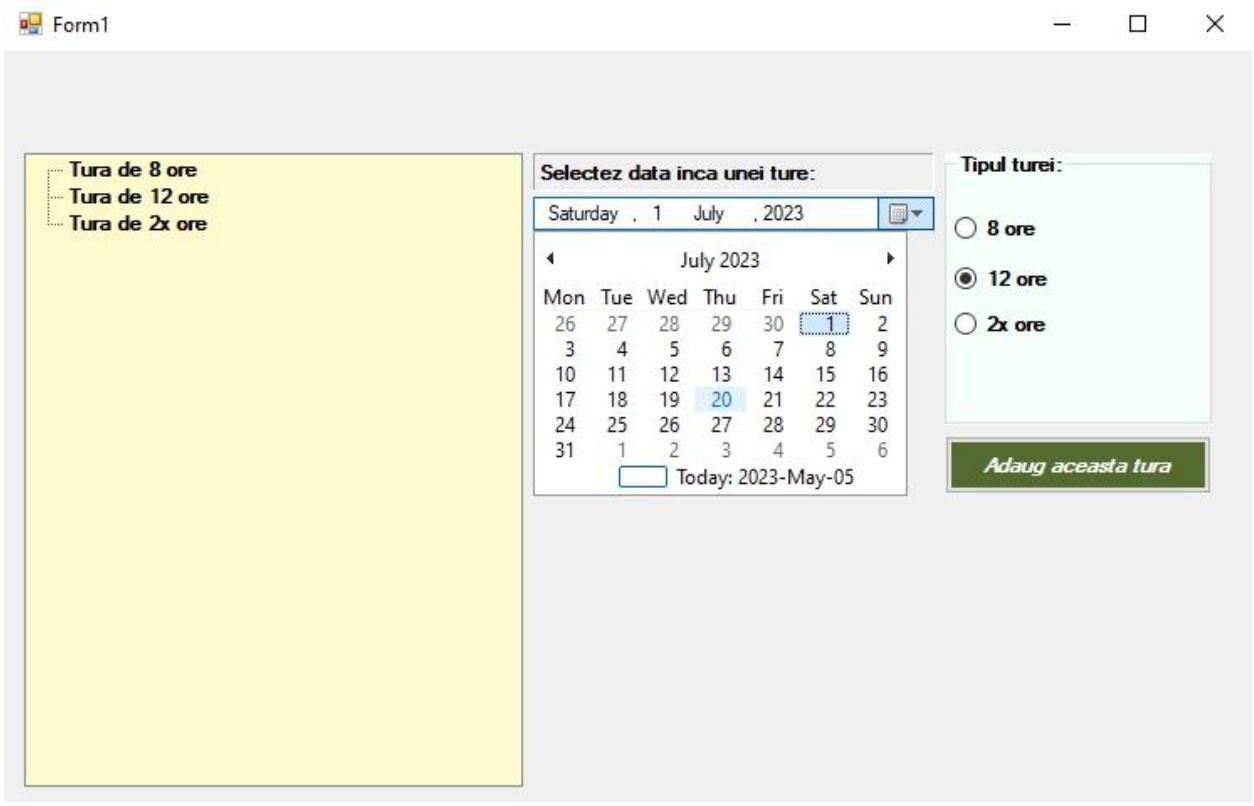
tvdateleturelor.Nodes["t8h"].Nodes.Add(tn_turanoua);
La colecţia de elemente/Noduri a elementuui/ramurii având numele/cheia "t8h"
se adaugă Nodul "**tn_turanoua**" creat anterior

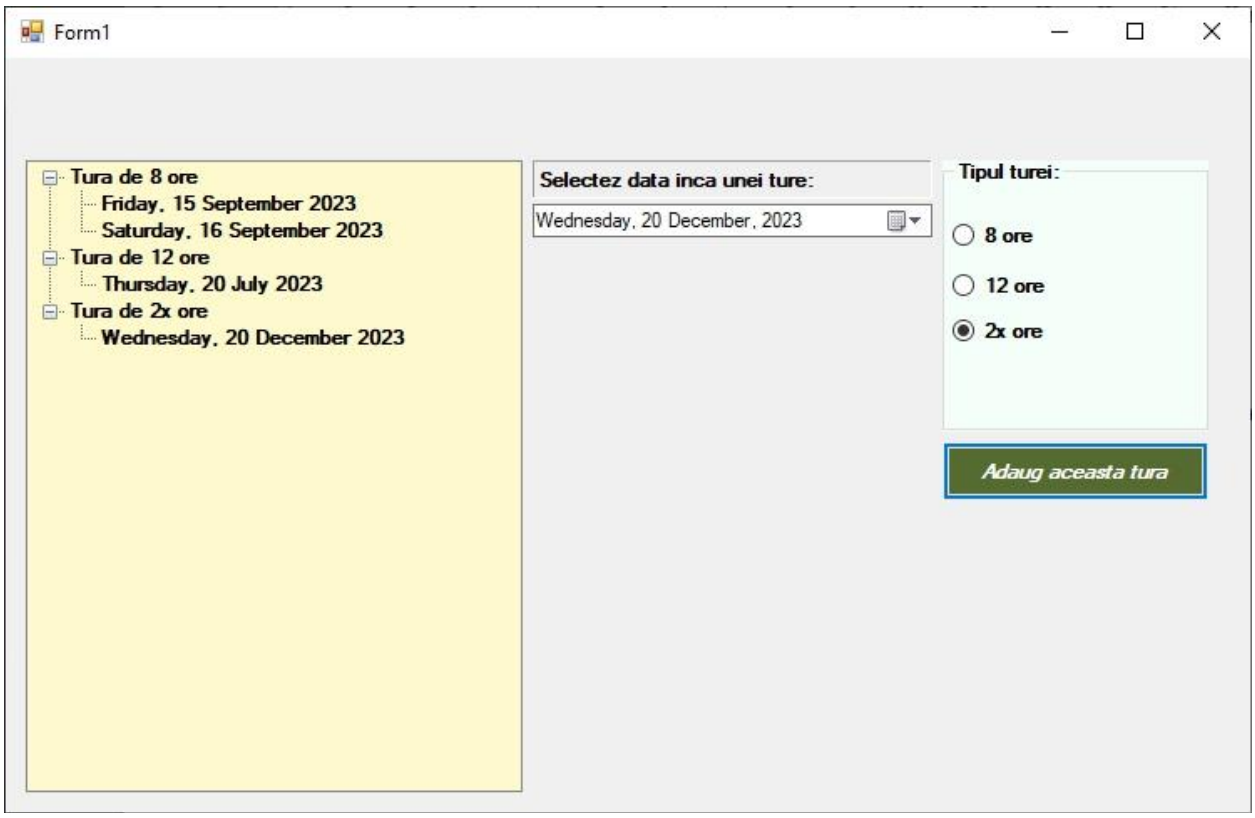
Celelate instrucţiuni ("else if...") – referitoare la cazurile când utilizatorul a selectat
celelalte opţiuni ("rbtura12.Checked == true", "rbtura2x.Checked == true")
sunt similare acesteia

Pornim programul – tasta F5:
Rezultatul după adăugarea mai multor ture ar fi similar acestuia:

- în prima imagine, din "dtpdateleturelor" a fost selectată data 20 iulie 2023 şi a fost bifat "rb12h"
- apoi, a fost adăugată data 15 septembrie 2023 şi tura de tipul "8 ore"
- apoi, a fost selectată data 10 decembrie 2023 şi tura de tipul "2x ore"
- apoi, a fost selectată data 16 septembrie 2023 şi tura de tipul "8 ore": deci pe/sub ramura "8 ore" din *TreeView* vor fi două elemente/Noduri – date calendaristice

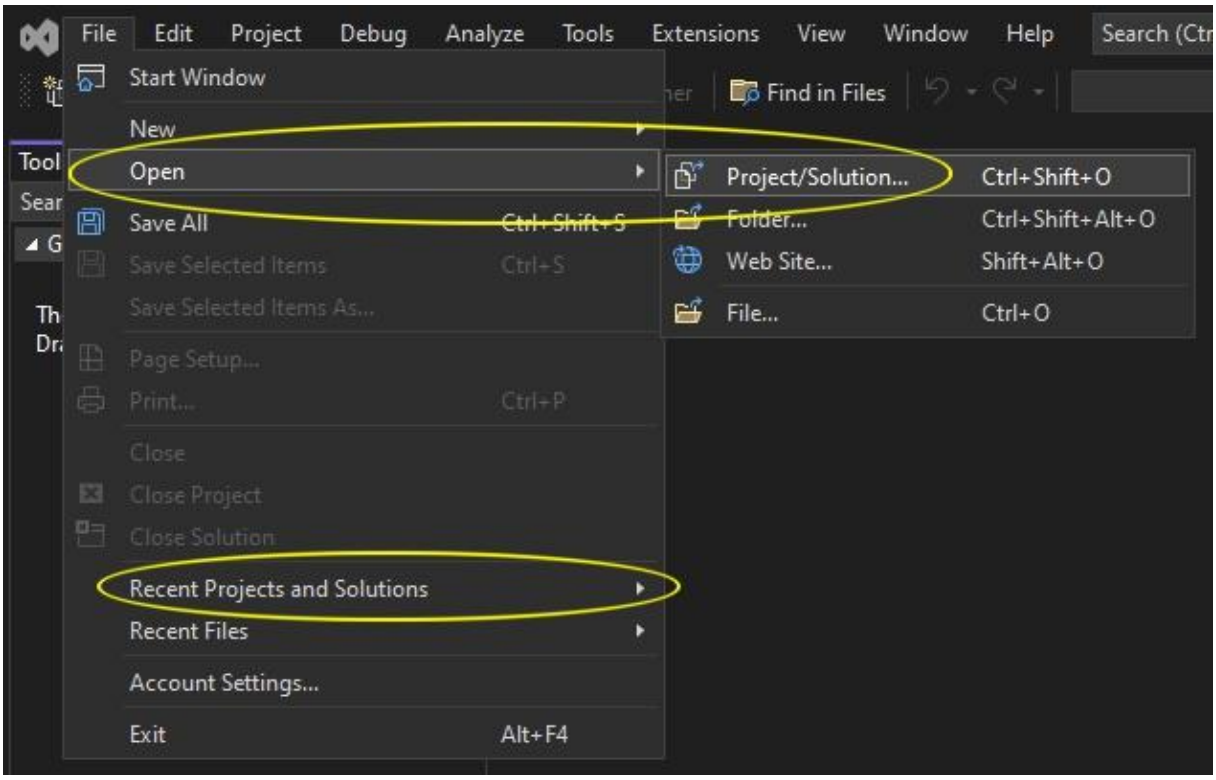
În a doua imagine din continuare se observă rezultatul acestor operaţiuni:


Form1
Tura de 8 ore
Tura de 12 ore
Tura de 2x ore
Selectez data inca unei ture:
Saturday , 1 July , 2023
July 2023
Mon Tue Wed Thu Fri Sat Sun
26 27 28 29 30 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31 1 2 3 4 5 6
Today: 2023-May-05
Tipul turei:
8 ore
12 ore
2x ore
Adaug aceasta tura


Form1
Tura de 8 ore
Friday, 15 September 2023
Saturday, 16 September 2023
Tura de 12 ore
Thursday, 20 July 2023
Tura de 2x ore
Wednesday, 20 December 2023
Selectez data inca unei ture:
Wednesday, 20 December, 2023
Tipul turei:
8 ore
12 ore
2x ore
Adaug aceasta tura

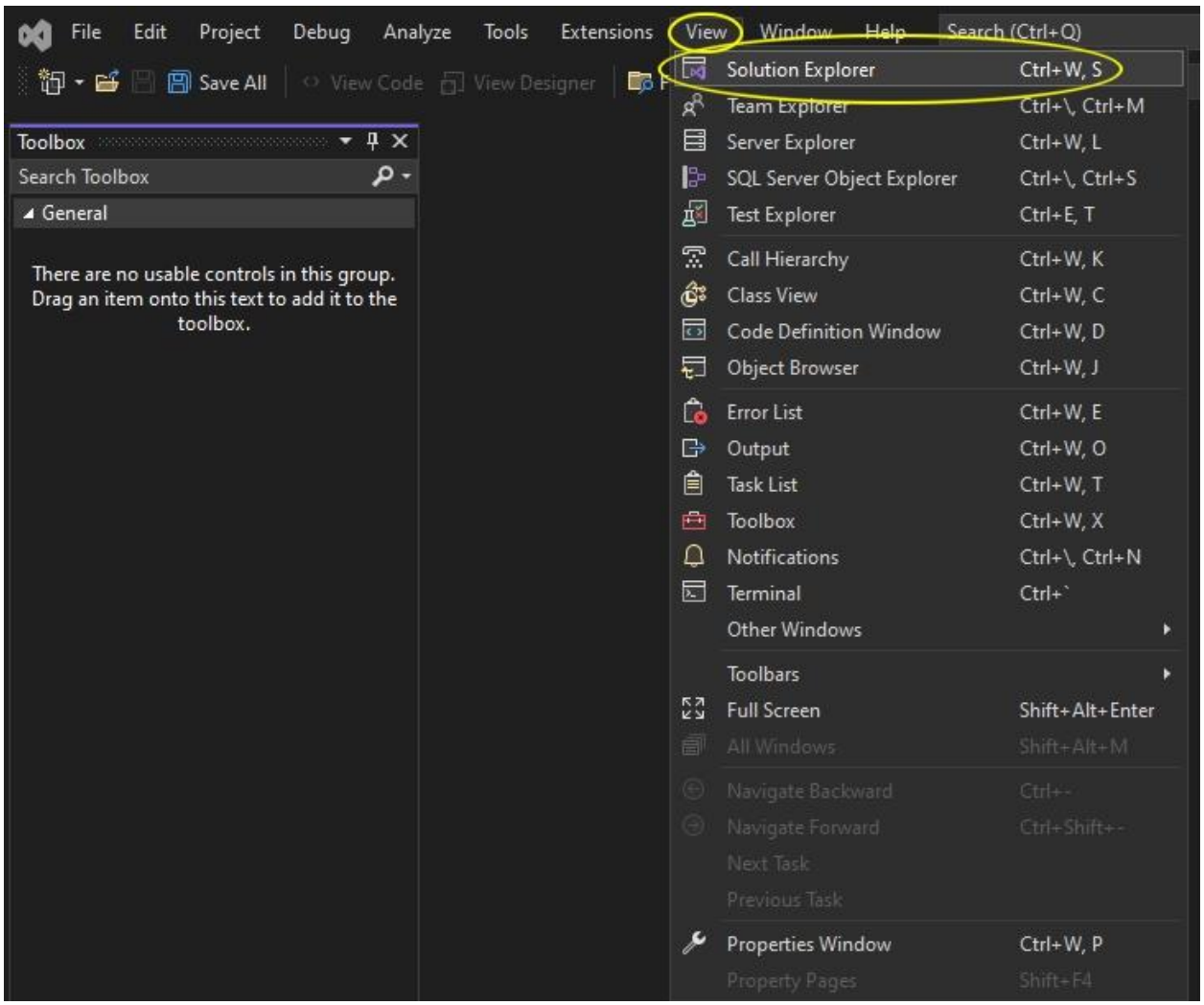
Întrucât intenţionăm ca la alte rulări ale programului nostru să mai adăugăm elemente la cele existente (colecţia .Nodes a *TreeView*-ului) şi eventual să modificăm sau să ştergem din acelea şi apoi să le accesăm iar şi iar:
Şi întrucât cele pe care le-am adăugat până acum se pierd la închiderea programului:
Salvăm într-un loc structura şi elementele arborelui astfel populat:
Bineînţeles, într-un loc de unde pot fi recuperate:
Anume, într-un fişier pe care îl putem transporta împreună cu programul (executabilul .exe) oriunde va fi nevoie de informaţiile din *TreeView*.

Trebuie să modificăm programul nostru.
Astfel că trebuie încărcat în Visual Studio:
- ***fie din meniul File -> Open -> Project/Solution...***
- ***fie din meniul File -> Recent Projects and Solutions:***



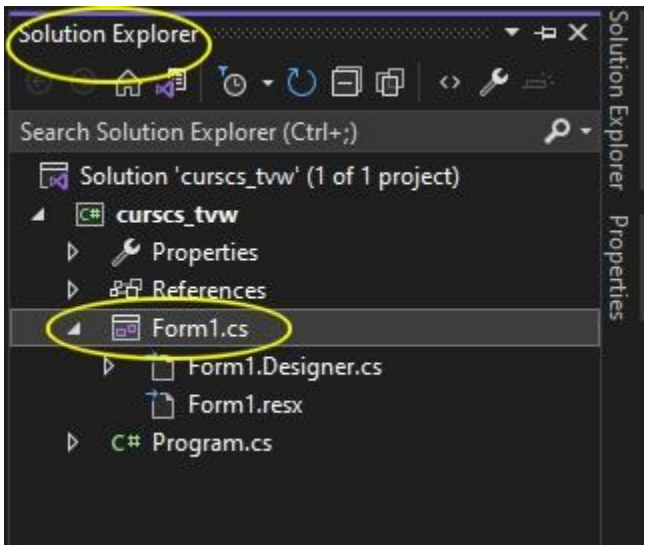
După ce Proiectul programului este astfel încărcat, dacă *Form1* nu este afişată imediat facem asta:

- din meniul "View" selectăm intrarea "Solution Explorer":



La toate cele care în majoritatea cazurilor sunt prezentate în secţiunea care va apărea a mediului de programare ne vom întoarce.

Acum, însă, ne interesează să modificăm interfaţa programului care constă în Dialogul *Form1*:
Aşa că în Solution Explorer dublu-click pe *Form1*:



Ceea ce ne trimite în modul de proiectare/Design al acelui Dialog, unde începem a-l modifica

Aşadar, vom lucra cu fişiere.

Revedem:
- obiectele *StreamWriter* şi *StreamReader*
- Controalele *SaveFileDialog* şi *OpenFileDialog*

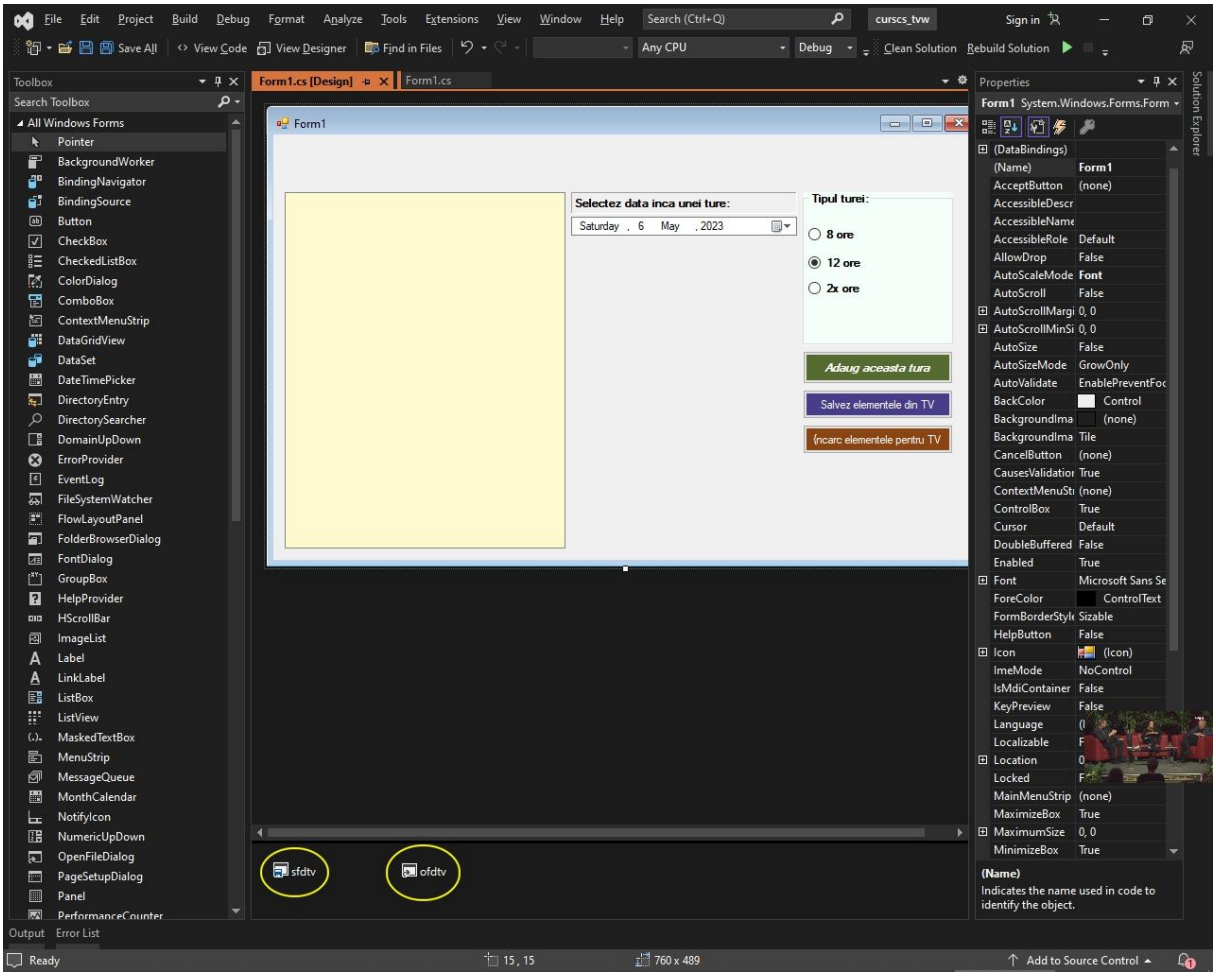
despre care strictul necesar s-a expus sumar în exemplul despre programul-editor de text asemănător lui *Notepad* – ***click aici*** pentru a citi acel exemplu.

Mergem în modul de proiectare/Design al *Form1*, unde se află *TreeView*-ul pe care vrem să-l salvăm: tastele Shift + F7.

Din Toolbox tragem pe Form1:
- un Control de tipul SaveFileDialog: acesta va folosi la salvarea elementelor din *TreeView*
- un Control de tipul OpenFileDialog: aceata va servi la încărcarea/recuperarea elementelor salvate din(tr-un) *TreeView*.
- un Button pe care scriem "Salvez elementele din TV" sau ceva cu acelaşi înţeles
- un Button pe care scriem "Încarc elementele pentru TV" sau ceva cu acelaşi înţeles

Redenumim:
- *SaveFileDialog*-ul: "sfdtv"
  (sau similar, oricum cât de cât sugestiv pentru a şti noi înşine ce anume face acest control)
- *OpenFileDialog*-ul: "ofdtv"
  (aceeaşi observaţie ca mai sus)
- *Button*-ul pentru salvarea într-un fişier a elementelor din *TreeView*: "bsavetv"
- *Button*-ul pentru încărcarea dintr-un fişier a elementelor de plasat în *TreeView*: "bloadtv"

Acum *Form1* ar arăta astfel:



Să scriem codul pentru aceste operaţiuni:

Fiindcă vom lucra cu fişiere (obiecte derivate din obiectul *Stream*), în grupul de directive *using* de pe primele rânduri ale codului *Form1* scriem o directivă pentru accesarea din acest cod a spaţiului de nume **System.IO**:

using System.IO;

Acum scriem codul pentru salvarea într-un fişier a elementelor din *TreeView*:
Dublu-click pe "bsavetv":
În metoda generată automat de Visual Studio asociată Evenimentului implicit *Click* a acestui buton scriem:

În comentarii (liniile marcate "//") scriem ce anume avem de gând să facem şi ce anume se întâmplă în instrucţiunile respective:

Dacă plasaţi acest cod în Visual Studio
După ce dublu-click pe "bsavetv"
liniile goale pot fi şterse:

```
private void bsavetv_Click(object sender, EventArgs e)
{
	//salvam elementele din TreeView-ul "tvdateleturelor"
	sfdtv.Filter = "Fisiere-text|*.txt";
	if (sfdtv.ShowDialog() != DialogResult.Cancel)
	{
		//pregatim fisierul in care vor fi salvate elementele din "tvdateleturelor":
		StreamWriter sw = null;
		//in aceasta versiune de compilator de cod,
		//un obiect de tipul StreamWriter trebuie initializat fie si cu null
		try
		{
			//lucram cu fisiere, astfel ca e obligatoriu sa incadram
			//operatiunile in blocuri try-catch(-finally)
			//
			sw = new StreamWriter(sfdtv.FileName);
			//
			//
			//parcurgem elementele din "tvdateleturelor" si le trimitem in fisierul
			//al carui nume tocmai l-am specificat:
			//
			for (int i = 0; i < tvdateleturelor.Nodes.Count; i++)
			{
				//daca elementul din aceasta iteratie are un Nume/o Cheie
				//in acest caz - unul din cele trei Noduri avand ca Text
				//tipurile turelor:
				//
				if (tvdateleturelor.Nodes[i].Name != null)
				{
					//separam textele cu caracterul #
					//trebuie sa alegem un caracter sau o secventa de caractere
					//care nu exista in Text-ul elementelor
					sw.WriteLine(
						tvdateleturelor.Nodes[i].Name + "#" +
						tvdateleturelor.Nodes[i].Text
						);
					//
					if (tvdateleturelor.Nodes[i].Nodes.Count > 0)
					{
						//daca Nodul din aceasta iteratie
						//are Noduri/elemente-descendent:
```

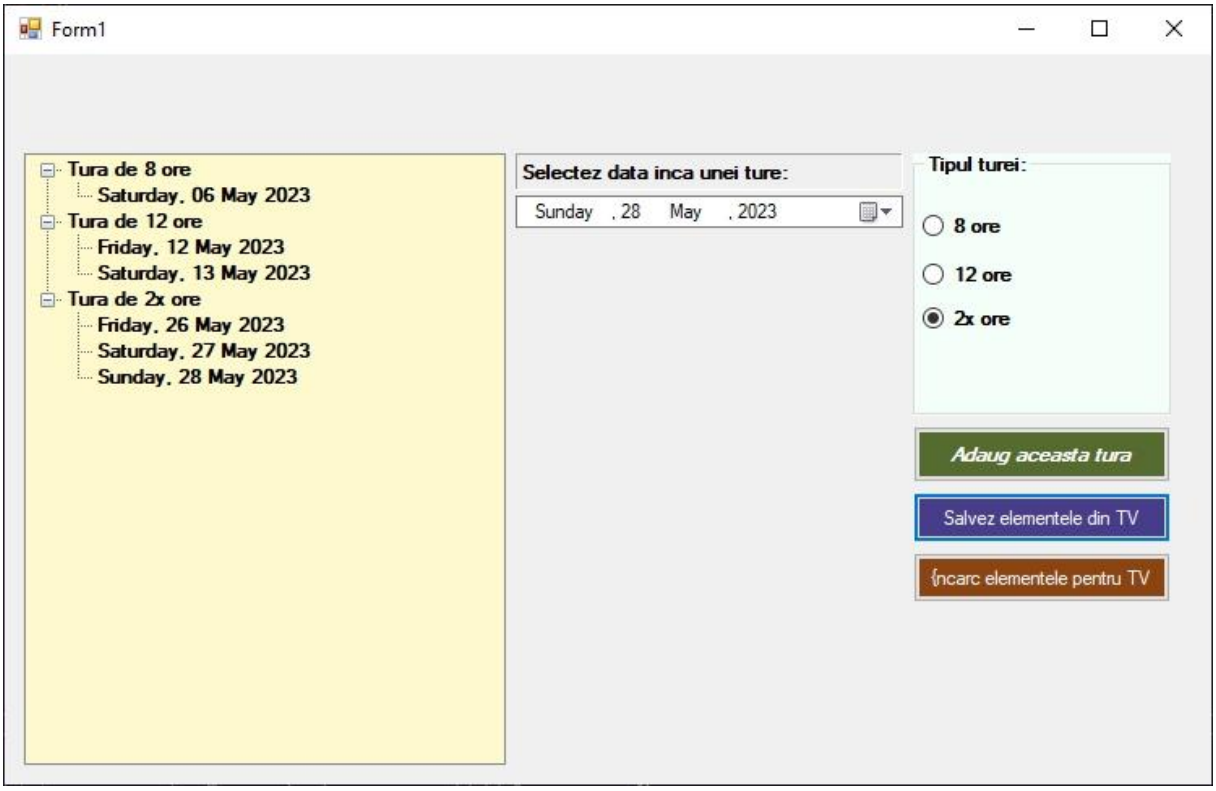
```
                                        //parcurgem colectia acelor Noduri-copil:
                                        //deci parcurgem colectia .Nodes din obiectul
                                        //Node curent avand Name/Cheie aflat la indexul i:

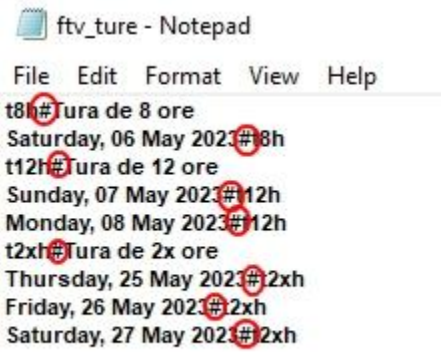
                                        //adica, Nodes[i].Nodes
                                        for (int k = 0;
                                        k < tvdateleturelor.Nodes[i].Nodes.Count; k++)
                                        {
                                                //element simplu, aflat sub unul din Nodurile
                                                //cu Name/Cheie:
                                                //----- Nodul aflat imediat deasupra
                                                //celui curent pe care il si contine -----
                                                //este referit ca .Parent:
                                                sw.WriteLine(
                                                tvdateleturelor.Nodes[i].Nodes[k].Text +
                                                "#" +
                                                tvdateleturelor.Nodes[i].Nodes[k].Parent.Name
                                                        );
                                        }
                                }
                        }
                }
                catch (Exception ex)
                {
                        MessageBox.Show("EROARE la salvarea TreeView:\r\n\r\n" + ex.ToString());
                }
                finally
                {
                        if (sw != null)
                        {
                                sw.Close();
                                //obligatoriu pentru inchiderea si accesarea ulterioara a acestui fisierul
                                sw.Dispose();
                                //eliberarea memoriei ocupate de acest obiect
                        }
                }
        }
}
```

În cele două imagini care urmează se vede cum cu acest cod au fost salvate în fişierul .txt specificat elementele din *TreeView*:

A doua imagine este o parte din captura de ecran surprinzând *Notepad* cu care am deschis acel fişier pentru a-i inspecta conţinutul:
Marcat cu roşu este caracterul # folosit pentru separarea Numelui/Cheii unui Nod de Text-ul său sau al unui element-descendent:

Datele calendaristice nu sunt aceleaşi în cele două imagini, dar funcţionalitatea este cea care contează:





Acum să scriem codul pentru încărcarea/recuperarea acestor informaţii din fişierul tocmai salvat:

Înapoi în modul de proiectare/Design al *Form1*, unde:
Dublu-click pe "bloadtv":
În metoda generată automat de Visual Studio asociată Evenimentului implicit *Click* a acestui buton scriem:

În comentarii (liniile marcate "//") scriem ce anume avem de gând să facem şi ce anume se întâmplă în instrucţiunile respective:

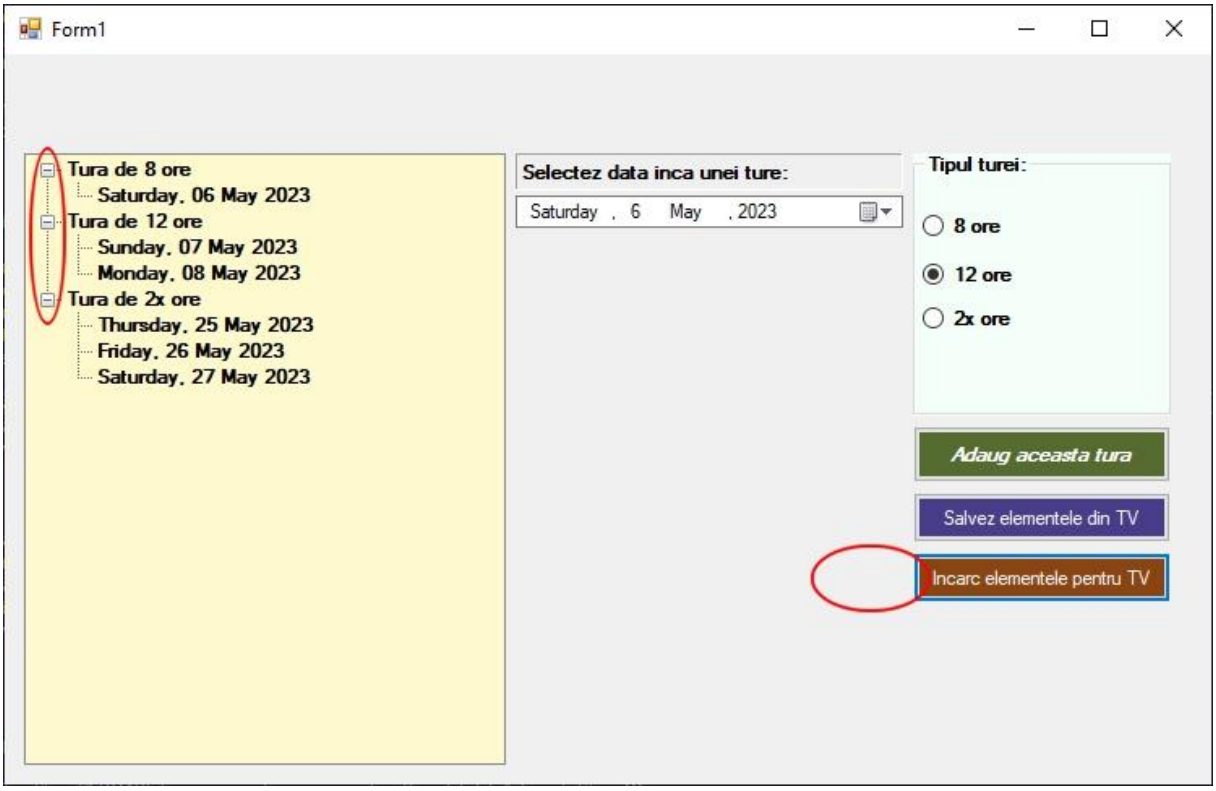
Dacă plasaţi acest cod în Visual Studio
După ce dublu-click pe "bloadtv"
liniile goale pot fi şterse:

```
private void bloadtv_Click(object sender, EventArgs e)
{
        //incarcam elementele pentru popularea TreeView-ului "tvdateleturelor"
        //
        //e bine ca atunci cand prelucram un fisier-text
        //sa punem in cod o portiune din acel text relevanta pentru codul nostru,
        //pentru a confrunta codul cu structura acelui text:
        //de exemplu, in acest caz:
        //
        //
        //t8h,Tura de 8 ore
        //Saturday, 06 May 2023,t8h
        //t12h, Tura de 12 ore
        //Sunday, 07 May 2023,t12h
        //
        //
        ofdtv.Filter = "Fisiere-text|*.txt";
        if (ofdtv.ShowDialog() != DialogResult.Cancel)
        {
                //acesta este fisierul din care vor fi citite elementele
                //de plasat in TreeView,
                //presupuse a fi scrise fiecare pe o linie
                //confruntam codul cu continutul fisierului
                //din care am adus aici in cod portiunea de mai sus:
                //
                StreamReader sr = new StreamReader(ofdtv.FileName);
                //
                //pregatim o variabila pentru liniile ce vor fi citite din fisier:
                //
                string slinie = "";
                //
                //inainte de a popula TreeView-ul stergem toate elementele din el:
                //
                tvdateleturelor.Nodes.Clear();
                //
                try
                {
                        //cat timp mai poate fi citita o linie din fisier:
                        //linie al carei text integral il atribuim (cu semnul = egal)
                        //variabilei slinie declarate mai sus:
```

```
                        while ((slinie = sr.ReadLine()) != null)
                        {
                                //din linia curenta cream un tablou de string-uri
                                //sectionand linia dupa caracterul # sau dupa secventa de caractere
                                //folosit/a cand am salvat elementele din TreeView:
                                string[ ] sa_linie = slinie.Split('#');
                                //
                                if (sa_linie.Length > 1)
                                {
                                        //verificam pentru a nu fi executate inutil instructiuni:
                                        //daca tabloul acestei linii contine
                                        //2 string-uri separate prin caracterul #
                                        //verificam daca primul string de pe aceasta linie
                                        //este un Nume/o Cheie, de exemplu "t8h":
                                        if (sa_linie[0].StartsWith("t") == true)
                                        {
                                                //acesta este un nod sub care se afla altele:
                                                //are Nume/Cheie care incepe cu "t":
                                                tvdateleturelor.Nodes.Add(sa_linie[0], sa_linie[1]);
                                        }
                                        else if (sa_linie[1].StartsWith("t") == true)
                                        {
                                                //acesta este un nod aflat sub altul:
                                                //nu are Nume/Cheie, insa are Parent-ul sau,
                                                //anume string-ul de la pozitia/indexul 1
                                                //in tabloul creat prin sectionarea liniei dupa caracterul #
                                                tvdateleturelor.Nodes[sa_linie[1]].Nodes.Add(sa_linie[0]);
                                        }
                                }
                        }
                }
                catch (Exception ex)
                {
                        MessageBox.Show("EROARE la incarcarea/recuperarea TreeView:\r\n\r\n" +
                                        ex.ToString());
                }
                finally
                {
                        //inchidem fisierul citit si distrugem obiectul corespunzator:
                        if (sr != null)
                        {
                                sr.Close();
                                sr.Dispose();
                        }
                }
        }
}
```

Astfel am recuperat elementele salvate mai devreme ale TreeView-ului.

Desigur, codul poate fi presărat cu *MessageBox*-uri care să afişeze mesaje despre stadiul programului în acea zonă.
De fapt, chiar e recomandat să se facă astfel; cel puţin până când vom folosi *Breakpoint*.

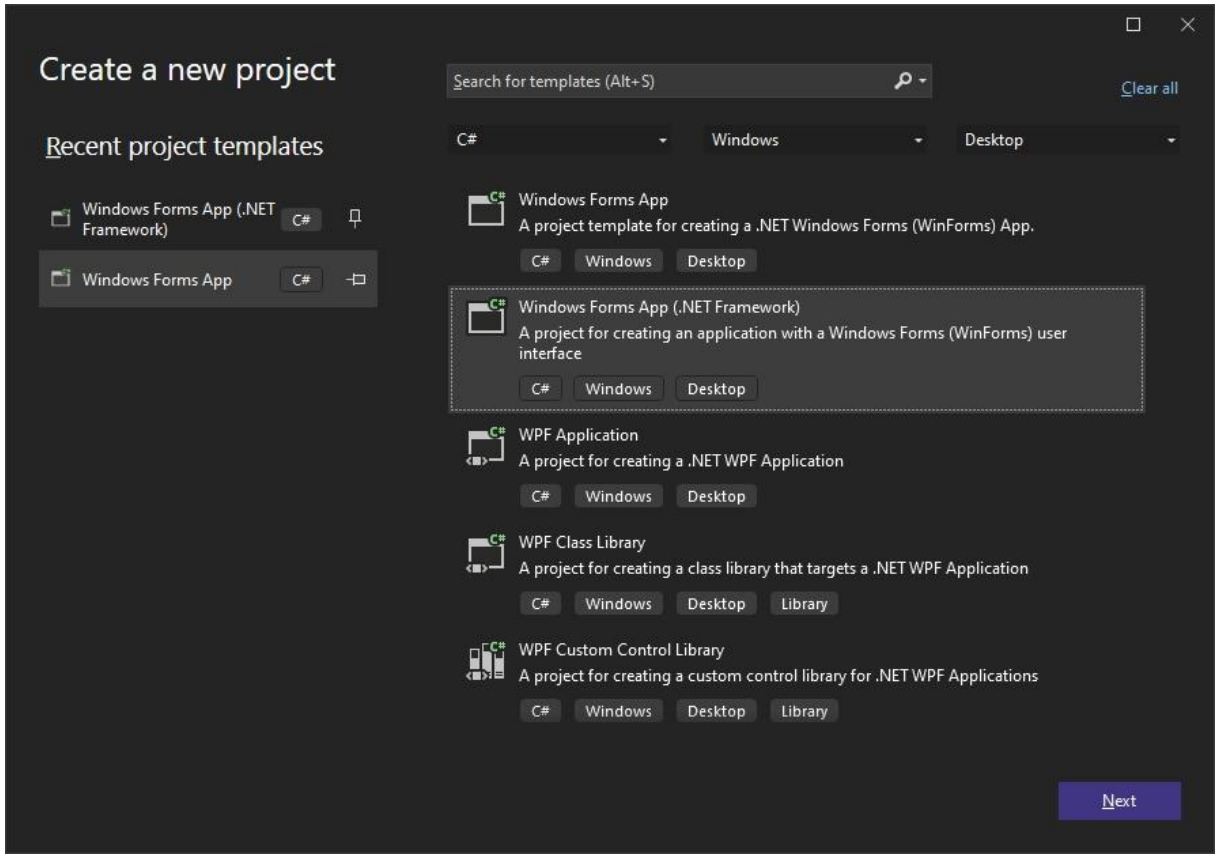

Form1
Tura de 8 ore
Saturday, 06 May 2023
Tura de 12 ore
Sunday, 07 May 2023
Monday, 08 May 2023
Tura de 2x ore
Thursday, 25 May 2023
Friday, 26 May 2023
Saturday, 27 May 2023
Selectez data inca unei ture:
Saturday , 6 May , 2023
Tipul turei:
8 ore
12 ore
2x ore
Adaug aceasta tura
Salvez elementele din TV
Incarc elementele pentru TV

Modificarea interfeţei

În acest exemplu se prezintă cum aspectul unei *Form*-e poate fi ***schimbat în timpul rulării programului***.

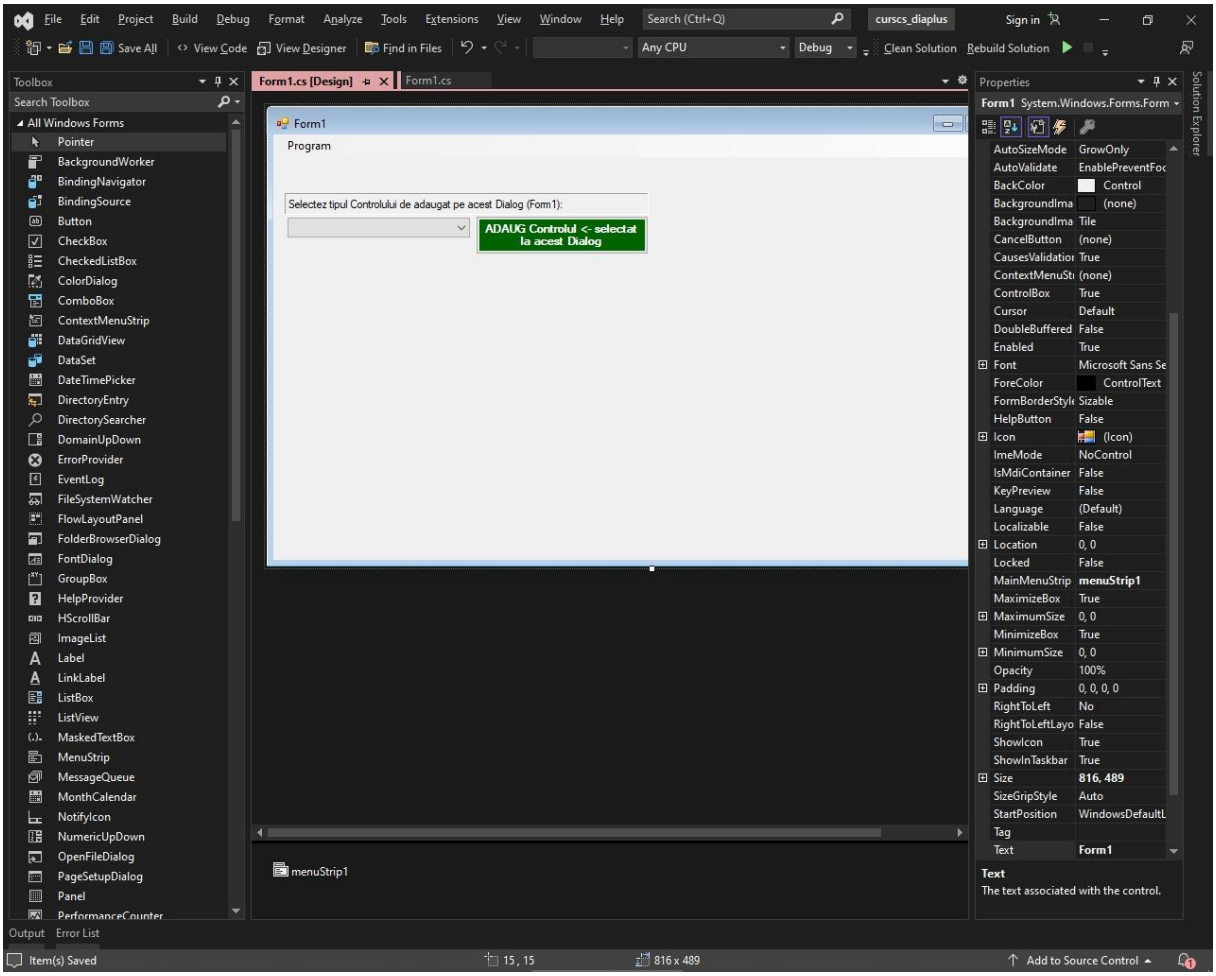
Aceasta se va face astfel:
- creând Controale care vor fi plasate pe Dialog
- stabilind unele Proprietăţi atât ale noilor Controale, cât şi ale unor Controale deja existente

Pornim Visual Studio, *File -> New... Project...*



Pe *Form1* tragem din Toolbox câteva Controale:
După ce fiecare Control este plasat pe *Form1*, apăsăm tasta F4 -> secţiunea Properties:
unde ***stabilim acum, când proiectăm Form1***, câteva Proprietăţi ale acelui Control, respectiv:
- o *MenuStrip* unde creăm pe nivelul principal al meniului elementul "Program" şi sub acesta intrarea de meniu "Exit" sau un text care să indice operaţiunea de închidere a programului ce va fi iniţiată la selectarea acestei opţiuni/intrări; denumim această intrare de meniu "miprogramexit"
- o casetă *TextBox* de editare a textului. *Name*: "tbstart"; *Multiline*: True; *ScrollBars*: Vertical
- un Button. *Name*: "badaugare1control"
- o casetă/listă derulantă *ComboBox*. *Name*: "cbtipulcontroluluideadaugat"; *DropDownStyle*: DropDownList
- imediat în preajma acesteia, o etichetă *Label* pe care scriem ceva despre rostul listei derulante

Interfaţa ar putea arăta astfel:



În lista "cbtipulcontroluluideadaugat" vom pune denumirile tipurilor de Control din care vom putea selecta câte unul pentru a adăuga un astfel de Control pe *Form1*.

Facem aceasta chiar la pornirea programului care, în acest caz, coincide cu afişarea *Form1*:

Aşadar, dublu-click pe suprafaţa *Form1* – ***nu*** în interiorul unui Control:
Ceea ce ne trimite în modul de scriere a codului metodei generate automat de către Visual Studio asociate Evenimentului *Load*,
cel implicit pentru o *Form*-ă:
unde scriem:

```
private void Form1_Load(object sender, EventArgs e)
{
        cbtipulcontroluluideadaugat.Items.Add("un TextBox");
        cbtipulcontroluluideadaugat.Items.Add("un TreeView");
        cbtipulcontroluluideadaugat.Items.Add("o ListBox");
}
```

Acum, înapoi în modul de proiectare/Design al *Form1*: simultan, tastele Shift şi F7:
Unde vrem să instruim butonul pentru a executa cod pentru adăugarea la *Form1* a unui Control de tipul selectat în *ComboBox*:

Deci, dublu-click pe butonul " badaugare1control":
Ceea ce, după cum deja ştim, ne trimite în modul de scriere a codului, în metoda generată automat asociată Evenimentului Click la care răspunde *numai* acest buton:

Unde scriem codul care începe pe următoarea pagină:
pentru înlesnirea Copy de aici – Paste în cod
unde **trebuie ca înainte să solicităm cu dublu-click pe butonul "badaugare1control"**
generarea de către Visual Studio a metodei asociate Evenimentului *Click* al acestui buton:

```
private void badaugare1control_Click(object sender, EventArgs e)
{
        switch (cbtipulcontroluluideadaugat.SelectedItem.ToString())
        {
                case "un TextBox":
                        TextBox tb_noua = new TextBox();
                        tb_noua.Name = "tbonouacasetadetext";
                        tb_noua.Multiline = true;
                        tb_noua.Width = badaugare1control.Width;
                        tb_noua.Height = 140;
                        tb_noua.Text =
                "Acesta este textul\r\ndin caseta de editare a textului\r\n\r\ntocmai adaugata";
                        tb_noua.Location = new Point(
                                badaugare1control.Left,
                                badaugare1control.Top + badaugare1control.Height + 20);
                        //in cele doua instructiuni care urmeaza, "this" inseamna
                        //"aceasta Form-a", deci Form1
                        tb_noua.Parent = this;
                        //^ altfel nu se va sti unde anume exista acest nou Control
                        tb_noua.Visible = true;
                        //^ altfel acest nou Control nu va fi vizibil
                        break;
                case "un TreeView":
                        //similar cream un Control de tipul TreeView
                        //la care adaugam cateva Noduri dupa modelul:
                        //tv_nou.Nodes.Add("numelenodului1","Textul Nodului 1");
                        //tv_nou.Nodes.Add("numelenodului2", "Textul nodului 2");
                        //
                        //fiind valabile aceleasi observatii despre instructiuni
                        //ca pentru noul TextBox
                        //
                        //tv_nou.Parent = this;
                        //tv_nou.Visible = true;
                        break;
                case "un ListBox":
                        //similar cream un Control de tipul ListBox
                        //la care adaugam cateva elemente dupa modelul:
                        //lb_noua.Items.Add("elementul 1 din lbox);
                        //lb_noua.Items.Add("elementul 2 din lbox);
                        //
                        //fiind valabile aceleasi observatii despre instructiuni
                        //ca pentru noul TextBox
                        //
                        //lb_noua.Parent = this;
                        //lb_noua.Visible = true;
                        break;
                default:
                        break;
        }
}
```

Să citim instrucţiunile care stabilesc unele Proprietăţi ale TextBox-ului creat cu acest cod:

TextBox tb_noua = new TextBox();
Declarăm/initializam/Propunem un nou Control de tipul TextBox
ca variabila *tb_noua* de tipul TextBox

De acum, această variabilă (*tb_noua*) / acest obiect
are toate atributele – Proprietăţi, Evenimente –
tipului său – anume, pe cele ale unui Control de tipul TextBox
Astfel că pot fi accesate din codul ulterior declarării sale:

tb_noua.Name = "tbonouacasetadetext";
Denumim astfel noul *TextBox*: Proprietatea *Name* a acestui Control

tb_noua.Multiline = true;
Stabilim Proprietatea MultiLine a acestui Control: noul TextBox va putea afişa text pe mai multe râdnuri

tb_noua.Width = badaugare1control.Width;
Stabilim Proprietatea *.Width* – Lăţimea noului *TextBox*:
anume, aceeaşi cu cea a butonului "badaugare1control"

tb_noua.Height = 140;
Stabilim Proprietatea *.Height* – Înălţimea noului *TextBox*:
anume, 140 pixeli

tb_noua.Text =
"Acesta este textul\r\ndin caseta de editare a textului\r\n\r\ntocmai adaugata";
Stabilim Proprietatea *.Text* – desigur, textul afişat iniţial în noul *TextBox*:
anume, aceeaşi cu cea a butonului "badaugare1control"

tb_noua.Location = new Point(
badaugare1control.Left,
badaugare1control.Top + badaugare1control.Height + 20
);
Stabilim Proprietatea *.Location* – Poziţia pe Form1 a noului TextBox:
anume,
o structură de tip *Point*
declarată ca având:

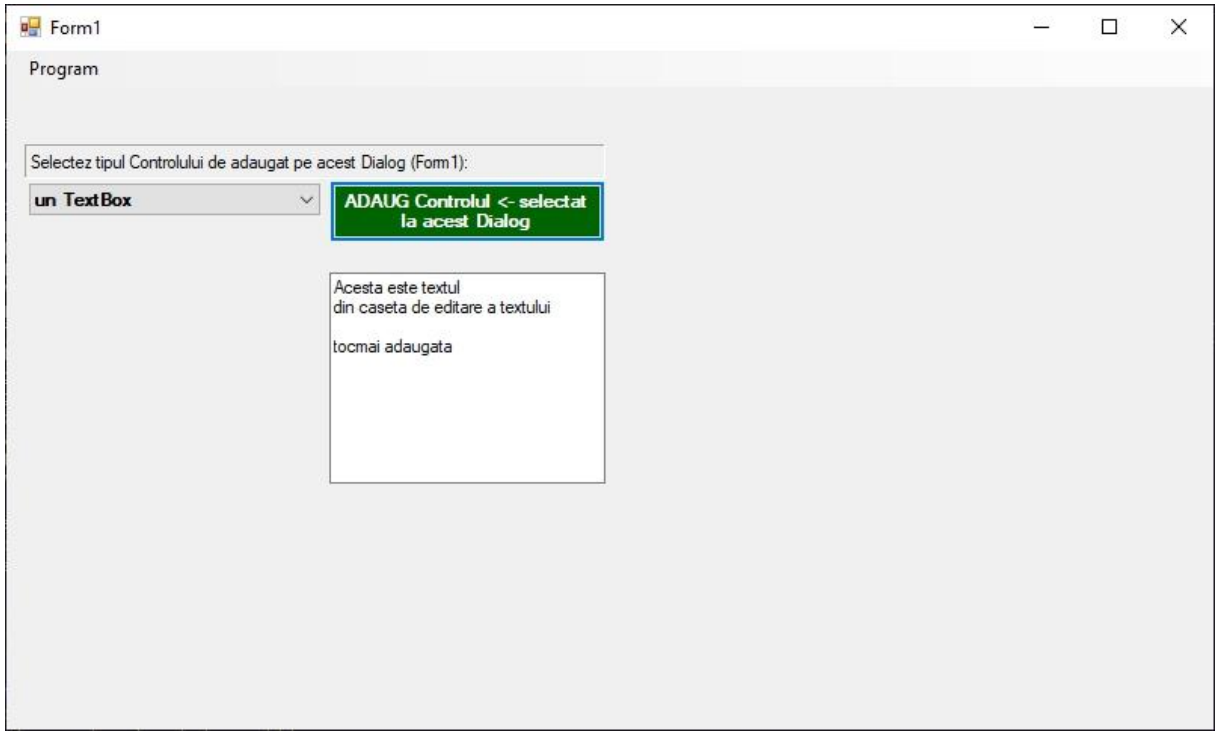
- coordonata *Ox*/pe abscisă aceeaşi cu cea a butonului "badaugare1control":
  adică, noul TextBox va fi aliniat la stânga cu butonul "badaugare1control"
- coordonata *Oy*/pe ordonată – suma dintre:
  coordonata *Oy* a butonului "badaugare1control" (.Top) +
  ,înălţimea butonului "badaugare1control" (.Height) +
  20 pixeli

Adică, noul *TextBox* va fi:

- aliniat la stânga cu butonul "badaugare1control"
- mai jos cu 20 pixeli decât butonul "badaugare1control"

Pornim programul: tasta F5:

Rezultatul după crearea unui Control de tipul TextBox ca în codul de mai sus ar fi asemănător acestuia:



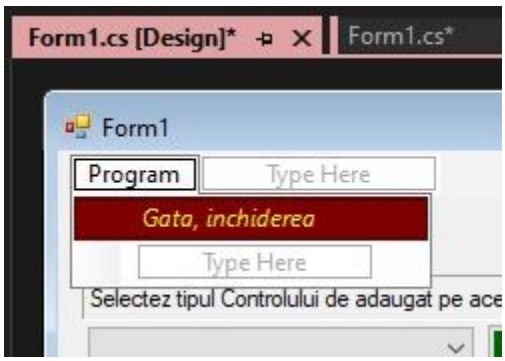
Fiind accesibile ***şi din cod*** toate Proprietăţile accesibile din secţiunea *Properties* a unui Control,
putem scrie instrucţiuni care să modifice mai departe Controalele nou adăugate,
de exemplu:

```
tb_noua.ForeColor = System.Drawing.Color.Red;
//aici se vede cum obiectul "Color"
//se accesează prin spatiul de nume (namespace-ul) System.Drawing
//
//in loc de a scrie System.Drawing.Color
//putem scurta instructiunile ca aceasta, unde este folosit un obiect dintr-un spatiu de nume,
//specificand la inceputul codului, in secventa declaratiilor using,
//faptul ca vor fi accesate entitatile din spatiul de nume System.Drawing, anume:
//using System.Drawing;
//apoi, in cod ne vom putea referi direct la entitati din acel spatiu de nume, de exemplu:
//tb_noua.ForeColor = Color.Red;
```

În codul programului, la fel cum pot fi stabilite Proprietăţile unor Controale inexistente până atunci
la fel pot fi schimbate Proprietăţi ale unor controale existente, ***în afară*** *uneori* ***de Proprietatea*** *"Name"*
de exemplu, Text-ul de pe butonul "badaugare1control":

```
badaugare1control.Text = "ADAUG UN CONTROL";
```

Intrarea de meniu "Gata" sau similar, din "Program", odată acţionată/selectată solicită confirmare pentru închiderea programului:



În modul de proiectare/Design al *Form1*:
dublu-click pe acel text:
apoi, în codul metodei asociate Evenimentului *Click* al acelei intrări de meniu scriem:

```
private void miprogramexit_Click(object sender, EventArgs e)
{
        if (MessageBox.Show(
                "Gata?",
                "Confirmare",
                MessageBoxButtons.YesNo,
                 MessageBoxIcon.Question,
                MessageBoxDefaultButton.Button2) ==
        DialogResult.Yes)
        {
                Application.Exit();
        }
}
```

Recapitulare (Reamintire)

Pentru a putea purcede cu încredere la crearea unui program cu pretenţia de a fi distribuit – folosit de alţi oameni decât de noi înşine care l-am făcut:

Din cele prezentate până acum ***trebuie*** reţinute-avute în vedere acestea:

1. Orice operaţiuni cu fişiere:
    a. necesită directiva **using System.IO;** în grupa directivelor *using* de la începutul codului oricărei *Form*-e/clase
    b. se pun în blocuri *try-catch*(*-finally*)
    c. obligă la specificarea căii spre un fişier separând directoarele cu secvenţa de caractere \\ - două slash-uri "inverse". De exemplu: fişierul "untext.txt" trebuie furnizat programului ca "D:\\teste\\untext.txt". *Dacă nu este specificată calea către un fişier, atunci programul presupune că fişierul se află în acelaşi director cu programul (cu fişierul .exe)*.

2. ***Cel mai / foarte frecvent***:
    a. din fişiere se citeşte folosind un obiect *StreamReader*
    b. din fişiere-text se citeşte cu *ReadLine*(), *ReadToEnd*()
    c. în fişiere se scrie folosind un obiect *StreamWriter*
    d. în fişiere-text se scrie cu *Write*(... şi *WriteLine*(...
    e. pentru a putea fi accesat, un StreamWriter (fişier în care s-a scris) trebuie închis cu metoda *Close*(), de exemplu: **sw.Close();**
    f. de asemenea, după ce s-a citit din conţinutul său (al fişierului asociat), un *StreamReader* trebuie închis folosind metoda *Close()*, de exemplu: **sr.Close();**
    g. pentru a elibera memoria ocupată de aceste două obiecte se foloseşte metoda *Dispose*(), comună amândurora, de exemplu: **sw.Dispose();** şi **sr.Dispose();**

3. Secvenţa \r\n este echivalentul apăsării tastei *Enter* după textul care o precede.
    a. astfel, \r\n\r\n reprezintă două taste *Enter*, adică o linie/un rând gol după textul care precede această secvenţă.
    b. în unele sisteme de operare este folosită numai: \n
    c. *System.Environment.NewLine* este acelaşi lucru, depinzând de sistemul de operare unde rulează programul; deci, nu trebuie să ştim care anume secvenţă denotă sfârşitul de linie

4. Programele lansate cu tasta F5 sunt transformate în fişiere cu extensia .exe - programe executabile care pot fi distribuite-transportate:
    a. locul unde se află acel fişier executabil (programul propriu-zis) se află în directorul (folderul) unde am creat Proiectul/Soluţia respectiv/ă şi pe care l-am specificat când în Visual Studio am ales tipul programului pe care vrem să-l construim, de exemplu:
    b. dacă în directorul "D:\teste" am creat un Proiect/Soluţie cu denumirea "programulmeu", atunci executabilul se află în directorul "D:\teste\programulmeu\bin", într-unul din subdirectoarele lui "bin": "*Debug*" şi "*Release*" (fiecare, prezentă dacă în Visual Studio am selectat acea versiune de executabil care trebuie compilată)

5. O casetă de dialog *MessageBox* întrerupe interacţiunea programului cu utilizatorul până când acesta din urmă reacţionează la acel dialog, închizându-l.

6. Blocul *switch-case* este util şi recomandat când este evaluată îndeplinirea mai multor condiţii decât numai a câtorva care pot fi evaluate cu blocuri *if...-else if... -else if...-else...*, de exemplu este recomandat pentru cazul când o variabilă numerică sau de tip *String* poate avea 15 valori preconizate/acceptabile şi mult mai multe valori necunoscute.

7. Cronometrul - Controlul *Timer* solicită periodic, la fiecare *Tick*-ăit (Evenimentul *Tick*), executarea unor instrucţiuni, inclusiv afişarea unor *MessageBox*-uri (atenţie la situaţia aceasta!)

8. Nu orice program necesită un Meniu; deseori, un Meniu este inutil şi poate fi înlocuit cu doua-trei Controale care însă nu trebuie a aglomera interfaţa şi incomoda utilizatorul; de exemplu, în loc de Meniu cu două intrări şi două comenzi (deci, patru texte pe "bara de sus" a *Form*-ei respective) pot fi plasate pe *Form*-ă două (deci 2 în loc de 4) butoane – unul pentru închiderea programului şi celălalt pentru afişarea unei MessageBox cu informaţii.

9. Controlul *TreeView* oferă posibilitatea de a prezenta informaţii grupate după caracteristici comune într-o structură arborescentă/ramificată, unde pe "trunchi" există "ramuri" şi "Noduri". De exemplu, ramurile ar putea fi denumirile dinastiilor din Egiptul antic şi Nodurile fiecărei ramuri - numele faraonilor; sau, ramurile ar putea fi titlurile unor cărţi şi Nodurile fiecărei ramuri – numele autorului, editura, anul ediţiei, codul ISBN.

10. Controlul *ListView* oferă posibilitatea de a prezenta informaţii grupate după caracteristici comune într-o structură tabelară, unde pe coloane se află "capul de tabel" şi pe linii se află "înregistrările". De exemplu, o *ListView* este adecvată prezentării cărţilor menţionate mai sus, la *TreeView*.

11. Accesarea unor informaţii despre sistemul de operare (Windows) pe care va rula programul se face folosind obiecte/membri (Proprietăţi, metode) din spaţiul de nume (*namespace*) **Environment**. Adică, în grupa directivelor *using* de la începutul codului oricărei *Form*-e/clase trebuie scrisă directiva **using Environment;** dacă nu se face acest lucru, atunci entităţile din acest spaţiu de nume vor fi accesibile scriind în instrucţiunile respective calea spre membrii folosiţi, de altfel ca în cazul oricărui membru al unui spaţiu de nume; de exemplu: *Environment.SpecialFolder*

12. Pentru afişarea imaginilor se foloseşte controlul *PictureBox*

13. Operaţiuni frecvente cu text (cu *String*-uri) sunt:
    a. identificarea (poziţiei) unui caracter folosind metodele *.IndexOf(...* şi *.LastIndexOf(...*
    b. extragerea/separarea unei particule dintr-un text folosind metoda *.Substring(...*
    c. trunchierea textului iniţial la o secvenţă de caractere care nu este nici precedată, nici urmată de spaţii libere (*blank*-uri) – folosind metoda *.Trim()*

14. Pentru organizarea unor informaţii în colecţii se pot folosi:
    a. tablourile (*Array*) de valori fie de acekaşi tip, fie de diverse tipuri (atunci, tabloul va fi declarat **object[]** tabloudediversevalori=...)
    b. *NameValueCollection*, unde sub o *cheie* se află mai multe *valori*

Construirea unui text folosind particule alese la întâmplare

Vom face un program care din cuvinte aparent disparate vor fi alese la întâmplare unele care, odată înlănţuite, vor alcătui un text dacă nu logic, atunci măcar corect din punct de vedere sintactic.

Pentru aceasta, vom folosi:

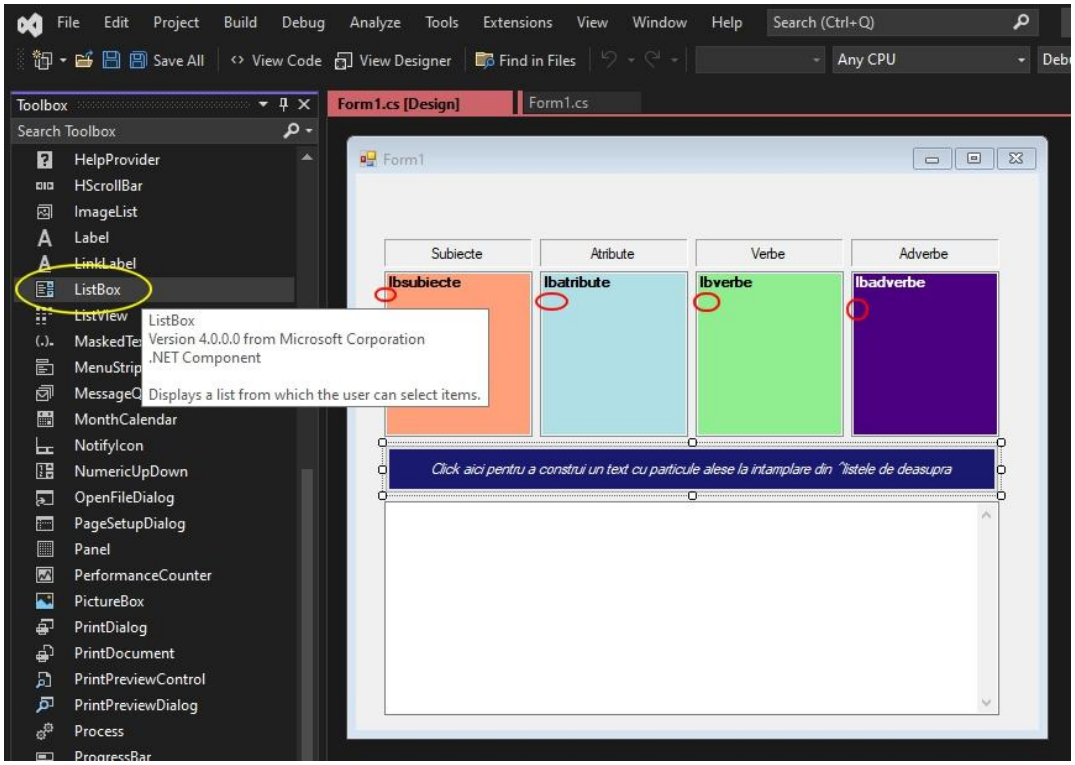
- tablouri (*Arrays*) de *String*-uri, anume patru (4) tablouri dedicate unul subiectelor (ori, substantivelor) de interes, altul atributelor, altul verbelor (ori, predicatelor), altul adverbelor
- Controlul ***ListBox***. Acesta este un control vizibil pe interfaţă (pe Dialog/Form-ă) şi care poate susţine/afişa în listă un număr oarecare de texte (*String*-uri), de exemplu cuvinte care să denote opţiuni selectabile ca într-o *ComboBox*.
  Acest Control este util atunci când utilizatorul trebuie să aibă la îndemână informaţii care pot fi structurate într-o listă şi vizibile mai multe odată - deci nu numai una (elementul selectat) sau niciuna (niciun element selectat), cum este cazul *ComboBox*.

Să pregătim interfaţa, adică Dialogul *Form1*:
Din ToolBox tragem pe *Form1* patru Controale *ListBox* pe care:

- le redimensionăm
- le aliniem cumva
- le colorăm după bunul plac, dar nu strident încât să deranjeze

Pe *Form1* mai plasăm:

- etichete *Label* care să indice rostul fiecărei *ListBox*
- un *Button* cu care vom solicita crearea textului din elemente ***alese aleator, la întâmplare*** din cele patru *ListBox*-uri



Vrem ca la pornirea programului – care în acest caz coincide cu afişarea *Form1*, *ListBox*-urile să fie populate cu elementele care vor fi disponibile în program:
Astfel că ne gândim la Evenimentul *Load*, implicit al unui Dialog/al unei *Form*-e:
Deci *ListBox*-urile vor fi populate în metoda acestui Eveniment:
Aşa că invocăm metoda asociată acestui Eveniment:
Dublu-click pe *Form1*, ceea ce ne duce în modul de scriere a codului, unde scriem:
Întâi declarăm tablourile cuvintelor – pe acelaşi nivel cu metoda "**private void Form1_Load(...**" (sau similar), pentru a fi disponibile pe toate nivelurile de cod inferioare:, inclusiv în această metodă:

Nu copiaţi (Copy) în intregime acest cod pentru a-l plasa (Paste) în cel din Visual Studio:

Puteţi copia numai de la comentariul "//declaram tablourile...

Respectaţi întocmai nivelurile instrucţiunilor prezentate aici:

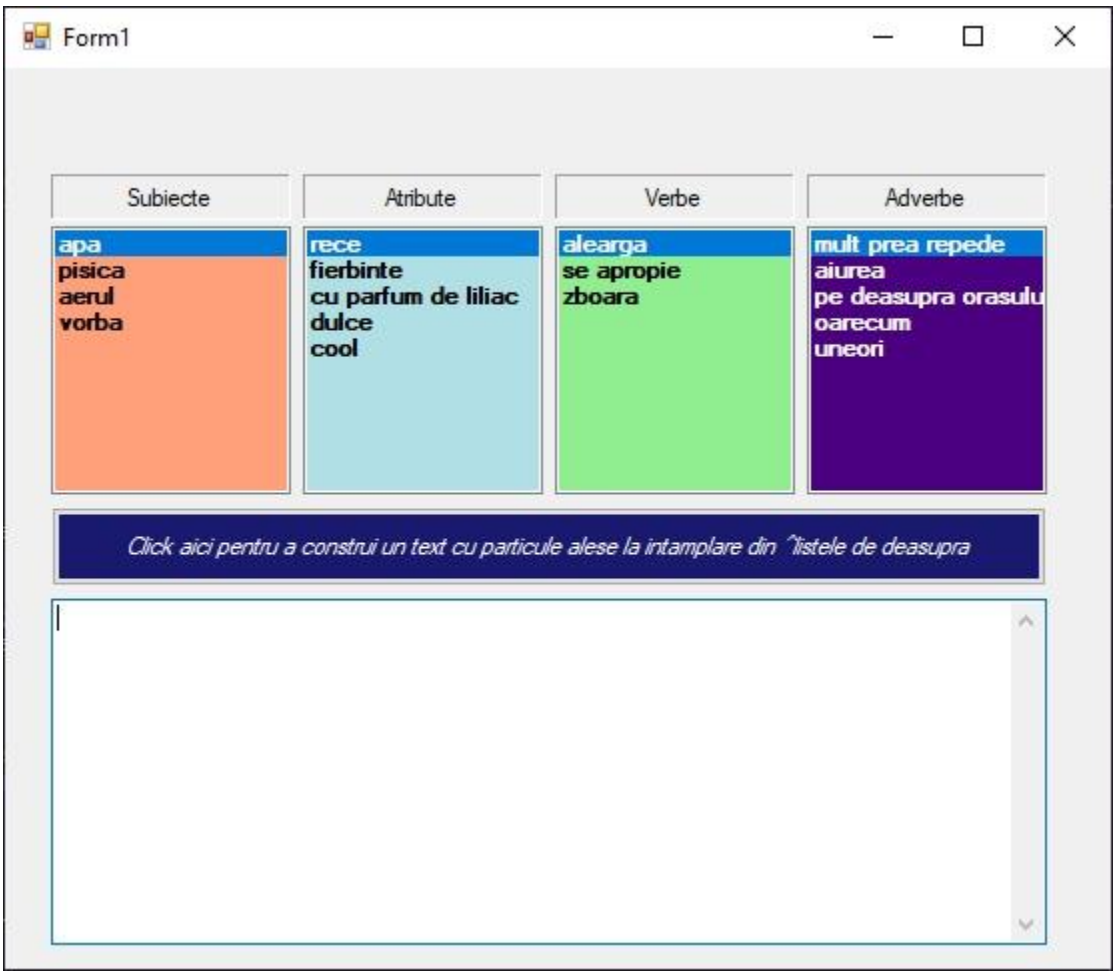
```
namespace curscs_rand
{
        public partial class Form1 : Form
        {
                //declaram tablourile de String-uri cu cuivntele/termenii/particulele/expresiile
                //din care vor fi alese la intamplare elemente cu care va fi alcatuit un text:
                //se observa cum fiecare tablou are un numar de elemente distinct
                //diferit ce cel al celorlaltor tablouri:
                string[ ] sa_subiecte =
                                { "apa", "pisica", "aerul", "vorba" };
                string[ ] sa_atribute =
                                { "rece", "fierbinte", "cu parfum de liliac", "dulce", "cool" };
                string[ ] sa_verbe =
                                { "alearga", "se apropie", "zboara" };
                string[ ] sa_adverbe =
                                { "mult prea repede", "aiurea", "pe deasupra orasului",
                                "oarecum", "uneori", "adesea" };
```

După ce am declarat tablourile în acest fel, în metoda **Form1_Load** scriem:

```
private void Form1_Load(object sender, EventArgs e)
{
        lbsubiecte.DataSource = sa_subiecte;
        lbatribute.DataSource = sa_atribute;
        lbverbe.DataSource = sa_verbe;
        lbadverbe.DataSource = sa_adverbe;
}
```

Proprietatea *.DataSource* a Controlului *ListBox* asociază o *sursă de elemente* acelui *ListBox*:
În acest caz, *.DataSource* a fiecărei *ListBox* o stabilim la tabloul cu elemente cu care vrem să fie populată acea listă.

Pentru a verifica, lansăm programul: tasta F5.
Rezultatul de până acum:



Acum închidem programuil şi ne întoarcem în modul de proiectare/Design al *Form1*:

Unde instruim butonul ca la fiecare apăsare a sa să fie generat un text din elemente alese la întâmplare din cele patru *ListBox*-uri.

Deci, alegerea nu va fi făcută de către utilizator:
Elementele vor fi alese **din cod**:

Pentru aceasta, dublu-click pe buton, ceea ce ne trimite iarăşi în modul de scriere a codului, în metoda asociată Evenimentului *Click* – implicit al unui Control de tipul *Button*:

Metodă în care vor fi aceste instrucţiuni care de aici pot fi copiate şi plasate în Visual Studio:

```
private void button1_Click(object sender, EventArgs e)
{
        //din fiecare tablou de string-uri selectam
        //la intamplare
        //cate un element
        //si cu cele 4 elemente astfel alese incropim un text
        //presupus a avea o structura logica --- data bineinteles de ordinea elementelor ---
        //
        //1 - alegem un subiect:
        Random rand_subiect = new Random(DateTime.Now.Millisecond);
        int i_indexulelementului_subiect = rand_subiect.Next(sa_subiecte.Length);
        string s_subiect = sa_subiecte[i_indexulelementului_subiect];
        //
        //2 - alegem un atribut:
        Random rand_atribut = new Random(DateTime.Now.Millisecond);
        int i_indexulelementului_atribut = rand_atribut.Next(sa_atribute.Length);
        string s_atribut = sa_atribute[i_indexulelementului_atribut];
        //
        //3 - alegem un verb (poate fi predicatul asociat subiectului):
        Random rand_verb = new Random(DateTime.Now.Millisecond);
        int i_indexulelementului_verb = rand_verb.Next(sa_verbe.Length);
        string s_verb = sa_verbe[i_indexulelementului_verb];
        //
        //4 - alegem un adverb (asociat verbului):
        Random rand_adverb = new Random(DateTime.Now.Millisecond);
        int i_indexulelementului_adverb = rand_adverb.Next(sa_adverbe.Length);
        string s_adverb = sa_adverbe[i_indexulelementului_adverb];
        //
        //
        string sTextulConstruitLaIntamplare =
                s_subiect + " " +       //dupa fiecare cuvant punem un spatiu/blank
                s_atribut + " " +
                s_verb + " " +
                s_adverb;
        //
        //curatam caseta de editare a textului in care plasam textul tocmai construit:
        tbtext.Clear();
        //
        //si plasam textul acela:
        tbtext.Text = sTextulConstruitLaIntamplare;
}
```

Citim linie cu linie:

Random rand_subiect = new Random(DateTime.Now.Millisecond);

Random este un obiect din spaţiul de nume/*namespace*-ul ***System*** şi care foloseşte la crearea/generarea unor numere *aleatoare*.
Iniţializarea sa poate fi făcută:

- fie fără argumente/parametri, astfel:
  - **Random r= new Random();**
- fie cu un parametru numeric în funcţie de care va fi generat un număr, astfel:
  - **Random r= new Random(valoaredecarevadepindenumarulgenerat);**

Am creat o variabilă de tipul ***Random***:

- pe care o şi iniţializăm ca fiind o valoare dependentă de numărul milisecundelor din data-ora curente. Dacă nu furnizăm un astfel de argument numeric, generatorul *Random* de numere aleatoare va produce de fiecare dată aceleaşi numere. Încercaţi.

int i_indexulelementului_subiect = rand_subiect.Next(sa_subiecte.Length);
Am creat o variabilă de timp numeric întreg:

- care va fi folosită la alegerea unui element din tabloul *sa_subiecte*
- şi a cărei valoare o stabilim la numărul elementelor din acel tablou; numerele generate vor fi mai mici decât acel număr, deci cel mai mare număr generat aleator va fi egal cu numărul elementelor din acel tablou

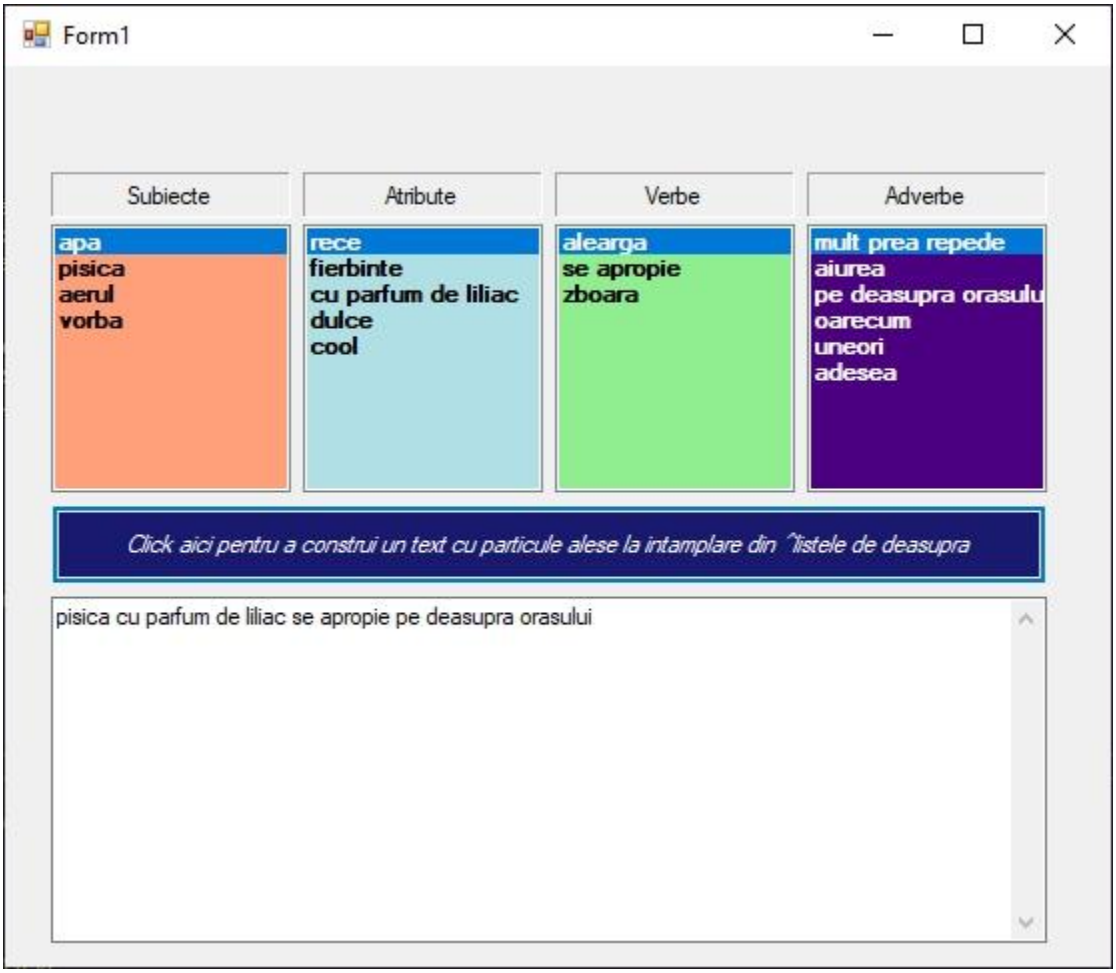
string s_subiect = sa_subiecte[i_indexulelementului_subiect];
Am folosit numărul generat cu instrucţiunea precedentă pentru a alege elementul aflat la acea poziţie/index în tabloul respectiv
Astfel am definit "subiectul" din textul final.drept valoarea variabilei ***s_subiect***

şamd.

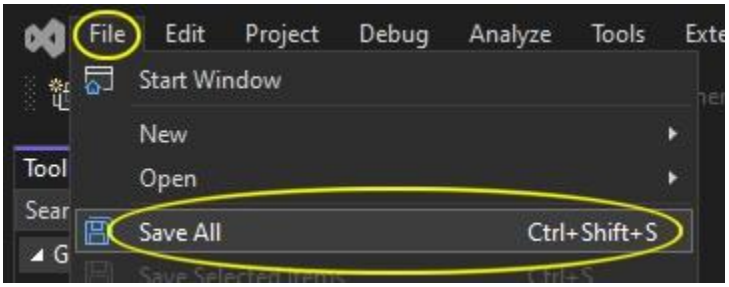
Pornim iarăşi programul: tasta F5.
Rezultatul, după apăsarea iar şi iar a butonului, va fi asemănător celui din imaginea următoare:

Bineînţeles, textul din *TextBox* poate fi copiat, modificat, şters la fel ca orice text dintr-un astfel de Control.

Construiţi alte tablouri cu elemente care pot fi înlănţuite-înşirate-dispuse într-o expresie complexă-ish.


Form1
Subiecte
Atribute
Verbe
Adverbe
apa
pisica
aerul
vorba
rece
fierbinte
cu parfum de liliac
dulce
cool
alearga
se apropie
zboara
mult prea repede
aiurea
pe deasupra orasulu
oarecum
uneori
adesea
Click aici pentru a construi un text cu particule alese la intamplare din ^listele de deasupra
pisica cu parfum de liliac se apropie pe deasupra orasului

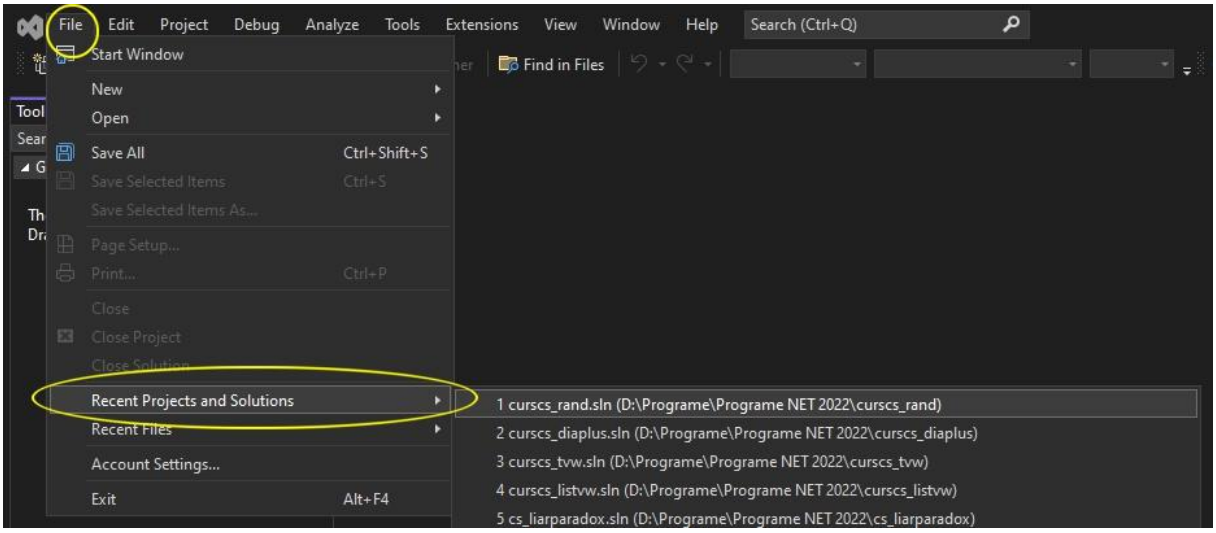
După ce considerăm că am terminat cu programul (cu orice program la care am lucrat) aşa cum este, ***salvăm toată munca din meniul File -> Save All***:



După un timp, vrem să schimbăm ceva la programul acesta; sau la unul *recent*.

Pentru asta, trebuie desigur să deschidem tot Proiectul/Soluţia programului:

Aceasta se face selectând intrarea de meniu
File -> Recent Projects and Solutions
şi Proiectul/Soluţia respectiv/ă:



Colecţii de String-uri: *StringCollection*

Aici se prezintă sumar funcţionalitatea obiectului *StringCollection*, încât să putem crea un program simplu, funcţional unde va fi folosit.

Obiectul *StringCollection* este un membru al/face parte din spaţiul de nume
System.Collections.Specialized
deci, este în acelaşi spaţiu de nume cu obiectul *NameValueCollection* prezentat sumar mai devreme.

Deşi acest obiect, ori o variabilă de acest tip poate susţine o mulţime (în sensul de ansamblu) de *String*-uri la fel ca un tablou *string[]*, diferenţa dintre aceste două entităţi constă în aceea că:

- un tablou *string[]* trebuie redimensionat pe măsură ce la el se adaugă elemente – programatorul măriudu-i, deci, dimensiunea cu o unitate (1) înainte de a adăuga un element, pe când:
- *StringCollection* este redimensionată automat pe măsură ce la ea se adaugă elemente

Astfel, *StringCollection* poate susţine ***un număr nespecificat*** de *String*-uri (de texte).

Metoda .**Add** a unui obiect de acest tip este cea care plasează un *String* în colecţie.
De exemplu: se adaugă *String*-ul: "textul 15" într-un *StringCollection* denumit *sctexte*:

```
sctexte.Add("textul 15");
```

Memoria .**Clear** eliberează memoria ocupată de un astfel de obiect - ştergând toate elementele aceluia, astfel:

```
sctexte.Clear();
```

Să vedem *StringCollection* în acţiune.

Pornim Visual Studio, *File -> New... Project...*

Deja ştim a proiecta o interfaţă în mod logic, sugestiv şi comod pentru utilizatorii programului nostru.
Aşadar, pe *Form1* plasăm – bineînţeles, trăgându-le din Toolbox – aceste Controale:

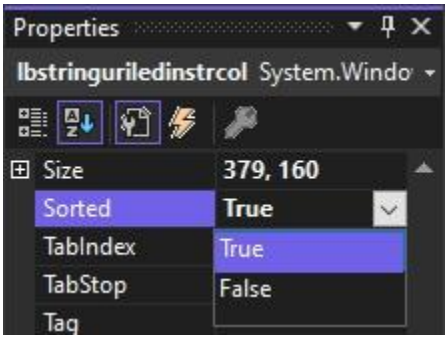
- un *Button* pe care îl redenumim: "badaugaretextlastrcol".
  În secţiunea Properties a sa (click-stânga pe el, apoi tasta F4):
  Stabilim Proprietatea *Enabled* la: False. Vom scrie cod încât acest buton să fie activ numai dacă există un text nenul de adăugat la *StringCollection*.
  Aceasta se poate face şi din cod, la pornirea programului (la afişarea *Form1*), în metoda ***Form1_Load***(..., astfel:
  badaugaretextlastrcol.Enabled = false;

Ne amintim că orice Proprietate
accesibilă în secţiunea Properties (pentru un Control) din Visual Studio
poate fi modificată în cod după modelul:
DenumireaControlului.DenumireaProprietăţii = ValoareaProprietăţii;

Astfel pot fi stabilite/modificate unele Proprietăţi
în timpul rulării programului

Fiecare *Click* pe acest buton va solicita adăugarea unui *String* (text) la obiectul *StringCollection* ale cărui elemente vor fi afişate în:

- o *ListBox* pe care o redenumim: "lbstringuriledinstrcol"
  Această casetă va afişa în listă elementele/*String*-urile/textele din obiectul *StringCollection*
  În secţiunea Properties a sa (click-stânga pe ea, apoi tasta F4):
  Stabilim Proprietatea *Sorted* la: True



  Elementele din această listă vor fi sortate alfabetic crescător (de la A la Z).
  Aceasta se poate face şi din cod, astfel:
  lbstringuriledinstrcol.Sorted = true;

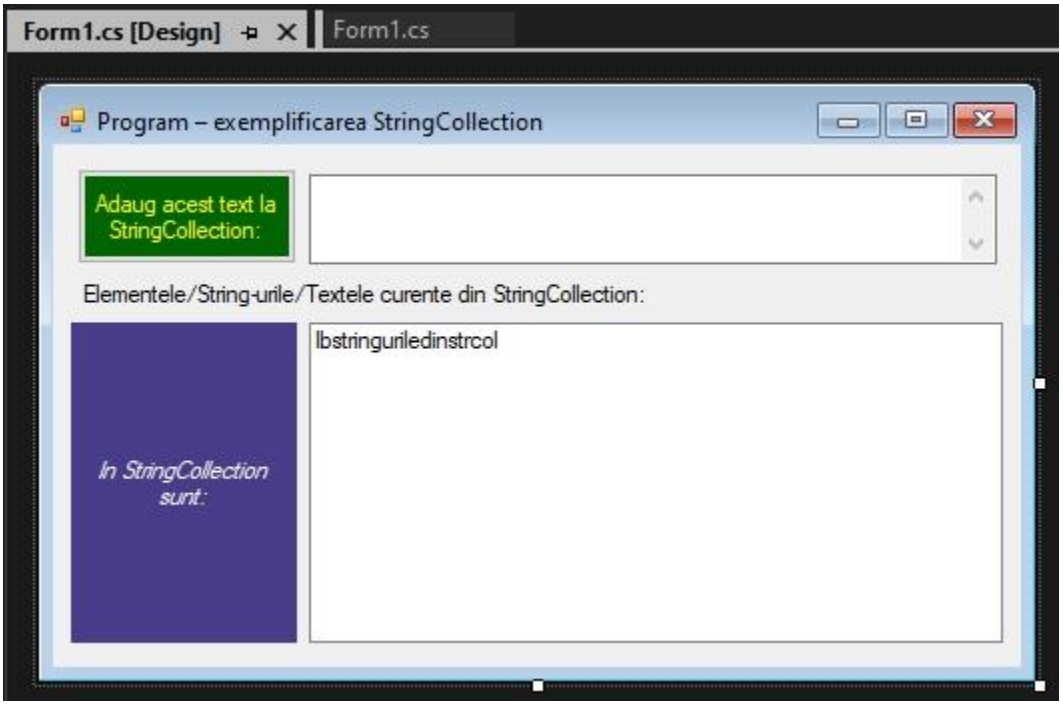
- o *TextBox* pe care o redenumim: "tbtextuldeadaugatlastrcol"
  În această casetă de editare a textului vom scrie textele de adăugat la *StringCollection*.
  Putem stabili Proprietatea sa *MultiLine* la True; nu contează pe câte rânduri este întins textul trimis în S*tringCollection* – fiindcă este un text "generic/obişnuit"

- etichete *Label* cu *Text* explicativ plasate logic imediat în apropierea celorlaltor Controale
  De obicei, nu trebuie să redenumim etichetele – de la denumirile lor implicite; asta fiindcă de obicei etichetele rămân aşa cum sunt afişate dintru bun început şi astfel redenumirea lor e la alegerea programatorului.

- o etichetă *Label* pe care o redenumim la: "labinfostrcol".
  Pe aceasta vom scrie câte elemente se află în *StringCollection*.
  Mai mult, în imaginea următoare se vede cum cu această etichetă am umplut spaţiul gol de pe interfaţă - în stânga *ListBox*-ului.
  Aşa ceva nu este recomandat – adică, nu se ocupă cu o Label, sau cu un Button, 10% sau mai mult din spaţiul unei Form-e.

- în secţiunea Properties a *Form1* (click-stânga pe *Form1*, apoi tasta F4):
  Stabilim Proprietatea *Text* a sa la: "Program – exemplificarea StringCollection".
  Se observă cum am pus numele unei entităţi .NET într-un *String*; este OK, *String*-ul este viabil.
  Aceasta se poate face şi din cod, astfel:
  Form1.Text = "Program – exemplificarea StringCollection";

Acum, interfaţa (Dialogul/*Form*-a *Form1*) ar putea arăta astfel:



Să creăm un *StringCollection* şi să adăugăm texte la el.

Deci, în acest exemplu uităm că există tablourile *string[]*.
Lucrăm numai cu tipul de colecţie *StringCollection*, mai flexibil şi de altfel mai accesibil.

Mergem în modul de scriere a codului: tasta F7.

Înainte de blocul care iniţializează clasa *Form1*
declarăm obiectul *StringCollection* cu care vom lucra:

```
System.Collections.Specialized.StringCollection sc =
                                        new System.Collections.Specialized.StringCollection();
```

Acum, acea porţiune de cod este:

```
System.Collections.Specialized.StringCollection sc =
                                        new System.Collections.Specialized.StringCollection();

public Form1()
{
        InitializeComponent();
}
```

Să instruim *TextBox*-ul "tbtextuldeadaugatlastrcol" unde propunem textul de adăugat la *StringCollection* încât să activeze butonul care adaugă acel text:

Dublu-click pe *TextBox*-ul "tbtextuldeadaugatlastrcol" în a cărui metodă generată automat pentru Evenimentul implicit *TextChanged* al acestuia scriem:

```
private void tbtextuldeadaugatlastrcol_TextChanged(object sender, EventArgs e)
{
        badaugaretextlastrcol.Enabled = (tbtextuldeadaugatlastrcol.Text.Trim() != "");
}
```

Acum scriem codul pentru adăugarea unui text la obiectul/variabila *sc* de tipul *StringCollection*, declarată şi iniţializată cum s-a prezentat.

Dublu-click pe butonul "badaugaretext" în a cărui metodă generată automat pentru Evenimentul implicit *Click* al acestuia scriem (se poate face *Copy-Paste*) codul de pe pagina următoare.

Comentariile (liniile marcate //) oferă detalii despre unele instrucţiuni:

```
private void badaugaretextlastrcol_Click(object sender, EventArgs e)
{
        sc.Add(tbtextuldeadaugatlastrcol.Text.Trim());
        //
        //am trimis un element/String/text in StringCollection, asa ca
        //trebuie sa il si afisam in ListBox:
        //afisam in ListBox toate elementele din StringCollection:
        //
        //inainte de a popula un Control cu o colectie integrala de entitati/texte
        //trebuie sters continutul acelui Control:
        //astfel ca inainte de a popula ListBox-ul cu toate elementele din StringCollection
        //stergem toate elementele din ListBox:
        //
        lbstringuriledinstrcol.Items.Clear();
        //
        //acum trimitem in ListBox elementele din StringCollection:
        //iterarea (for...) se face pe toate elementele din StringCollection:
        //acel numar este Proprietatea .Count a unui StringCollection:
        //
        for (int s = 0; s < sc.Count; s++)
        {
                lbstringuriledinstrcol.Items.Add(sc[s]);
        }
        //
        //actualizam textul de pe eticheta
        //care prezinta numarul curent al elementelor din StringCollection:
        //se observa folosirea secventei \r\n (echivalentul efectului tastei Enter)
        //pentru a face mai lizibil textul relativ lung de pe eticheta:
        //
        labinfostrcol.Text = "Acum, in StringCollection (variabila sc)\r\nsunt:\r\n\r\n" +
                             sc.Count.ToString() +
                             "\r\n\r\nelemente";
}
```

Pornim programul – tasta F5:
Rezultatul după adăugarea unor texte la *StringCollection* – adică, la variabila *sc* va fi asemănător acestuia:



Odată ce am terminat treaba cu un obiect/ o variabilă de tipul *StringCollection*,
putem folosi metoda ***Clear()*** de curăţare/ştergere a întreg conţinutului său:
în codul de mai sus, al acestui program,
eventual înainte de instrucţiunea **Application.Exit();**
am scrie instrucţiunea: **sc.Clear();**

Fişiere conţinând text în format XML

XML (Extended Markup Language) este un model de structurare a unor informaţii de tip text
(deci, *String*-uri) într-o formă asemănătoare celei afişate într-un Control *TreeView* (v. mai sus).

Diverse porţiuni de informaţie caracterizând un anumit concept sunt stocate/înregistrate în fişierul XML pe niveluri, primul nivel fiind aşa-zisa rădăcină a întregului Document XML (acesta fiind întreg conţinutul fişierului XML), din care derivă Noduri/Elemente care conţin acele secvenţe de informaţie despre nodul imediat superior lor.

De exemplu, pe rădăcina numită "Biblioteca mea"
pot exista elemente care pot fi duplicate şi identificabile de asemenea ca elemente unice sau nu
prin grupele de particule de informaţie caracterizând fiecare element care denotă o carte:
toate elementele individuale ar putea fi numite "carte"
şi fiecare astfel de element ar avea Atribute: titlul cărţii, autorul, anul ediţiei ş. a.

Astfel, fişierul va avea structură arborescentă, cu:
- o rădăcină: XmlRoot
- elemente: XmlElement, XmlNode

La data acestui document (mai, 2023 A. D.),
într-un fişier formatat (ori, de tipul) XML poate fi stocat numai text.

Astfel, un fişier XML conţinând informaţii ca text simplu poate servi ca o bază de date referitoare la colecţii de caracteristici ale unor diverse entităţi.

După alcătuirea unui fişier XML, elementele pot fi accesate pentru folosire şi modificare.

Vom alcătui un program care:
- prezintă un formular care trebuie completat cu informaţii despre cel puţin trei persoane cu: nume, prenume, localitatea de domiciliu
- salvează într-un fişier XML informaţiile din formularul completat
- accesează informaţiile salvate în fişierul XML

Pornim Visual Studio, *File -> New... Project...*

Pe Form1 plasăm:
- casete *TextBox* de editare a textului, corespunzătoare acestor porţiuni de informaţie: numele, prenumele, localitatea de reşedinţă
- etichete *Label* cu *Text* descriptiv pentru fiecare *TextBox*
- un *Button* cu care se va solicita reţinerea în memorie a informaţiilor scrise, în vederea înregistrării lor toate odată într-un fişier XML. Stabilim Proprietatea sa *Enabled* la: False.
- un *Button* cu care se va solicita salvarea informaţiilor într-un fişier XML. Stabilim Proprietatea sa *Enabled* la: False.
- un *Button* cu care se va solicita recuperarea informaţiilor dintr-un fişier XML
- un *Button* cu care se va solicita afişarea informaţiilor recuperate dintr-un fişier XML. Stabilim Proprietatea sa *Enabled* la: False. Îl redenumim "bciclareprinelemxmldinfisier".

- un Control *SaveFileDialog* de salvare a unui fişier; il redenumim "sfd"

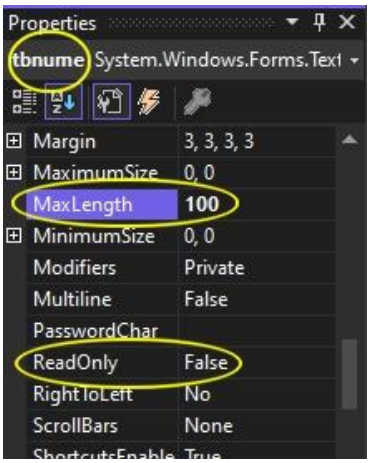
- un Control *OpenFileDialog* de selectare pentru deschidere a unui fişier; îl redenumim "ofd"
  În secţiunea Properties stabilim Proprietatea "*Filter*" a celor două Controale *xFileDialog* la (fără ghilimele): Fisiere XML |*.xml
- un *Button* cu care se va solicita afişarea informaţiilor despre /următoarea serie de informaţii recuperate (incărcăte) dintr-un fişier XML
- o etichetă *Label* pe care va fi afişat numărul elementelor de înregistrat în fişierul XML

Denumim (Proprietatea *Name*) fiecare din aceste Controale în acord cu rolul său.

Pentru fiecare *TextBox* unde utilizatorul programului va scrie ceea ce este solicitat acolo (nume etc.):
Stabilim Proprietatea sa *MaxLength* la un număr care reprezintă:
Lungimea maximă acceptabilă a textului trimis cumva (fie scris, fie cu comanda *Paste*, fie în alt fel) într-o casetă *TextBox* de editare a textului:
Click-stânga pe acea *TextBox*, apoi tasta F4 pentru a afşa secţiunea Properties din Visual Studio:
Acolo modificăm proprietatea *MaxLength* a TextBox-ului:

În imaginea următoare se observă cum pentru *TextBox*-ul denumit "tbnume" am stabilit la 100 lungimea maximă a textului acceptat de această casetă:
Se observă şi cum ne-am asigurat că în "tbnume" se poate **scrie**, deci că textul poate fi editat:
Aceasta este Proprietatea "*ReadOnly*" şi bineînţeles are două valori de adevăr:

- *False* – textul din *TextBox* poate fi editat/modificat
- *True* – textul va ramâne aşa cum a fost plasat în *TextBox* înainte de stabilirea la *True* a acestei Proprietăţi.



Interfaţa ar putea arăta astfel:



Butonul dedicat înregistrării în memorie a unei înregistrări va fi activ numai dacă
în toate casetele *TextBox* există un text nenul (adică, diferit de ""):

Pentru aceasta se monitorizează Evenimentul *TextChanged* – cel implicit al unui Control *TextBox*:

Astfel, dublu-click pe caseta denumită aici "tbnume" şi în care utilizatorul va scrie numele persoanei:

```
private void tbnume_TextChanged(object sender, EventArgs e)
{
        binregmemorieinfoafisate.Enabled =
                (
                (tbnume.Text.Trim() != "")
                &
                (tbprenume.Text.Trim() != "")
                &
                (tblocalitate.Text.Trim() != "")
                );
}
```

Procedăm la fel cu celelalte două *TextBox*-uri, de asemenea denumite sugestiv pentru rolul în program:

```
private void tbprenume_TextChanged(object sender, EventArgs e)
{
	binregmemorieinfoafisate.Enabled =
		(
		(tbnume.Text.Trim() != "")
		&
		(tbprenume.Text.Trim() != "")
		&
		(tblocalitate.Text.Trim() != "")
		);
}

private void tblocalitate_TextChanged(object sender, EventArgs e)
{
	binregmemorieinfoafisate.Enabled =
		(
		(tbnume.Text.Trim() != "")
		&
		(tbprenume.Text.Trim() != "")
		&
		(tblocalitate.Text.Trim() != "")
		);
}
```

Acesta nu este un mod elegant şi nici cât de cât profesional de a scrie cod,
ci este aşa-zis "cod spaghetti" şi un semn de diletantism.

Astfel că: în loc de a scrie de trei ori aceleaşi instrucţiuni
se poate crea o metodă în care să existe acele instrucţiuni, de exemplu:

```
private void ActivareaButonului_DeInregistrareInMemorie()
{
	binregmemorieinfoafisate.Enabled =
		(
		(tbnume.Text.Trim() != "")
		&
		(tbprenume.Text.Trim() != "")
		&
		(tblocalitate.Text.Trim() != "")
		);

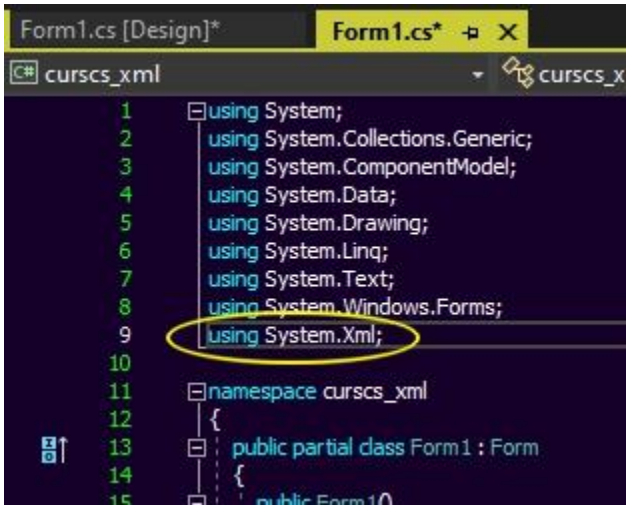
}
```

Apoi, acea metodă va fi apelată/invocată în metodele Evenimentului *TextChanged* de mai sus, în locul scrierii repetate a acelui blocul relativ lung de evaluare a unor condiţii pe care l-am pus în metodă, astfel:

```
private void tbnume_TextChanged(object sender, EventArgs e)
{
        ActivareaButonului_DeInregistrareInMemorie();
}
```

Acum în secvenţa **using** de la începutul codului *Form1* adăugăm această instrucţiune care va deschide accesul spre membrii spaţiului de nume dedicat entităţilor referitoare la lucrul cu XML:

```
using System.Xml;
```

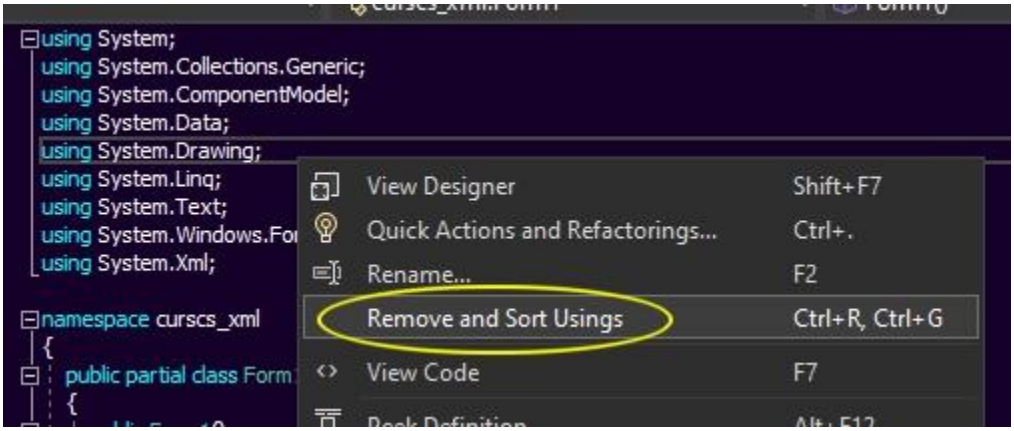


Vedem aici spaţii de nume despre care ori nu ştim nimic, ori nu ştim dacă membri ai lor vor fi folosiţi în programul nostru:

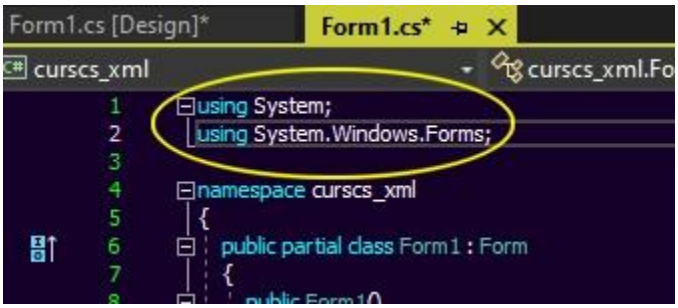
Secvenţa iniţială a spaţiilor de nume care vor putea fi accesibile din program este generată automat de către Visual Studio.

Dacă vrem să reducem acea secvenţă la spaţiile de nume folosite acum în codul programului:

Click-dreapta în cod, apoi selectăm "*Remove and Sort Usings*":



Comandă după care secvenţa **using** va putea arăta astfel:



Vedem că instrucţiunea tocmai scrisă despre spaţiul de nume *System.Xml* a fost ştearsă.

Aceasta s-a întâmplat fiindcă Visual Studio nu a detectat nicio referire explicită sau implicită la vreun membru al acelui spaţiu de nume, ci numai la spaţiile de nume *System* şi *System.Windows.Forms*.

Dar întrucât ştim că vom apela membri ai spaţiului de nume *System.Xml*, în secvenţa **using** adăugăm iarăşi instrucţiunea care ne va asigura accesul la membrii aceluia:

using System.Xml;

Acum scriem cod pentru înregistrarea în memorie a informaţiilor furnizate în *TextBox*-uri:

În vederea înregistrării/salvării lor într-un fişier XML, vom stoca în memorie fiecare secvenţă de informaţii furnizate în TextBox-uri:

Pentru aceasta folosim un obiect/o variabilă de tipul StringCollection (v. mai sus) la care vom adăuga acele trei porţiuni/particule de informaţie alăturate într-un *String* distinct şi desigur delimitate printr-un separator.

Pentru a accesa obiectul *StringCollection* trebuie să "importăm" în cod spaţiul de nume care conţine acest membru, anume *System.Collections.Specialized*:
Aşadar, în secvenţa **using** de la începutul codului scriem:

using System.Collections.Specialized;

Acum definim acel obieect de tipul *StringCollection* care va susţine toate informaţiile furnizate de către utilizatorul programului:
Scriem această instrucţiune înainte de prima metodă *_TextChanged* a unui *TextBox*, pe acelaşi nivel cu acea metodă:

StringCollection scinfo_persoanc_memorie = new StringCollection();

Înapoi în modul de proiectare/Design al *Form1* unde:
Dublu-click pe butonul marcat în acest exemplu "Inregistrare":

Comentariile detaliază instrucţiunile respective:

La *Paste*, după *Copy*-erea codului din continuare, liniile libere pot fi şterse:

```
private void binregmemorieinfoafisate_Click(object sender, EventArgs e)
{
        string snume = tbnume.Text.Trim();
        string sprenume = tbprenume.Text.Trim();
        string slocalitate = tblocalitate.Text.Trim();
        //
        //pentru a delimita/separa aceste particule de informatie
        //folosim un separator complex, de exemplu "#@#"
        //acesta ar fi o secventa de caractere
        //a carei prezenta este foarte improbabila
        //in oricare din particulele astfel delimitate separate:
        //
        string ssep = "#";
        //
        //String-ul, textul de adaugat la colectia StringCollection
        //din memorie
        //cu informatii despre /persoane
        //va fi construit din cele trei particule de informatie (tot String-uri, texte):
        //
        string sinfo_persoana_pentru_sc_memorie =
                snume + ssep +
                sprenume + ssep +
                slocalitate;
        //
        //trimitem/adaugam acest String la variabila de tip StringCollection definita mai devreme:
        //
        scinfo_persoane_memorie.Add(sinfo_persoana_pentru_sc_memorie);
        //
```

```
        //odata plasate in memorie, stergem textele din TextBox-uri:
        //
        tbnume.Clear(); tbprenume.Clear(); tblocalitate.Clear();
        //

        //actualizam Text-ul din eticheta care afiseaza numarul elementelor din StringCollection:
        labinfonumelemsc.Text = "Acum in StringCollection sunt:\r\n" +
                                        scinfo_persoane_memorie.Count.ToString() +
                                                "\r\nelemente";
        //am adaugat o secventa de informatii/texte despre o /persoana
        //care acum se afla in memorie:
        //in variabila scinfo_persoane_memorie
        //si care vor putea fi inregistrate intr-un fisier XML
}
```

Întrucât în fişierul XML vrem să înregistrăm mai multe astfel secvenţe de informaţii odată, nu câte una pe rând, monitorizăm numărul elementelor din obiectul *StringCollection scinfo_persoane_memorie*
Astfel încât să activăm butonul de salvare în fişier XML numai dacă este nimerit, dacă avem ce:

Deci, vrem să fie înregistrată o colecţie de texte de tipul *StringCollection* având mai mult de 2 elemente.

Astfel, după fiecare adăugare la variabila noastră *StringCollection*
trebuie obţinut numărul elementelor sale:
Aşa că la sfârşitul metodei de mai sus, înainte deci de acolada sa de închidere scriem:

```
//
//activam butonul pentru salvarea intr-un fisier XML a informatiilor
//daca StringCollection folosita aici are mai mult de 1 element:
//
bfisierxmlinregistrare.Enabled = (scinfo_persoane_memorie.Count > 2);       //sau, .Count >= 3
```

Acum scriem codul pentru salvarea într-un fişier XML a elementelor din obiectul *StringCollection*:

Pentru aceasta, înapoi în modul de proiectare/Design al *Form1*, unde:
Dublu-click pe butonul marcat "Salvare in fisier XML" sau cu un *Text* descriptiv sugestiv pentru rolul acestui buton:
În metoda asociată Evenimentului *Click* al său scriem:

/urmează un cod lung; poate fi *Copy*-at şi plasat (*Paste*) în cod
numai după ce, desigur, dublu-click pe butonul "bfisierinregistrare"
pentru ca Visual Studio să genereze metoda asociată Evenimentului *Click* al acestui buton.

Comentariile la instrucţiuni prezintă detaliat operaţiunea respectivă:

```
private void bfisierxmlinregistrare_Click(object sender, EventArgs e)
{
        //pregatim un obiect de tipul XmlDocument
        //in care vom trimite fiecare secventa de informatii corespunzatoare /unei persoane
        //acest obiect va fi stocat in memorie
        //dupa ce va fi populat cu Elemente XML, continutul sau va fi salva intr-un fisier XML:
        //
        XmlDocument xd_persoane = new XmlDocument();
        //
        //definim Elementul-radacina al acestui document XML
        //sub acest Element vor fi celelalte Elemente ale documentului XML:
        XmlNode xn_radacina =
                        xd_persoane.CreateNode(XmlNodeType.Element, "persoane", "");
        //
        //iteram prin StringCollection si definim cate un Element XML
        //pentru fiecare astfel de /persoana:
        //fiecare astfel de Element va fi adaugat la Elementul-radacina al documentului XML
        //la sfarsitul iteratiei se adauga Elementul-radacina la Documentul XML:
        for (int s = 0; s < scinfo_persoane_memorie.Count; s++)
        {
                //cu fiecare particula-text citita din StringCollection
                //se defineste un Element XML pentru acea /persoana:
                //
                XmlNode xnpersoana =
                        xd_persoane.CreateNode(XmlNodeType.Element, "persoana", "");
                //
                //pentru a defini pentru XML caracteristicile/atributele acestei /persoane,
                //secventa curenta de trei particule de informatie despre acea /persoana
                //este sectionata dupa separatorul "#"
                //folosit mai devreme, la crearea unui text pentru StringCollection:
                //
                string[ ] scaracteristicilepersoaneicurente = scinfo_persoane_memorie[s].Split('#');
                //
                //daca in tabloul astfel creat din particulele de informatie
                //din care este formata secventa curenta
                //exista trei texte - numele, prenumele, localitatea:
                if (scaracteristicilepersoaneicurente.Length == 3)
                {
                        //cream un Atribut al acestui Element XML creat pentru o /persoana:
                        //acest Atribut va fi cunoscut in documentul XML "xd_persoane"
                        //sub denumirea/textul: "nume":
                        //
                        XmlAttribute xa = xd_persoane.CreateAttribute("nume");
                        //
                        //stabilim Valoarea (proprietatea .Value) acestui Atribut
                        //la prima particula din secventa de informatii despre persoana curenta:
                        //--- desigur --- presupunand --- ca prima particula este numele persoanei:
                        //aceasta particula se afla la
                        //indexul 0 in tabloul "scaracteristicilepersoaneicurente:
                        //
```

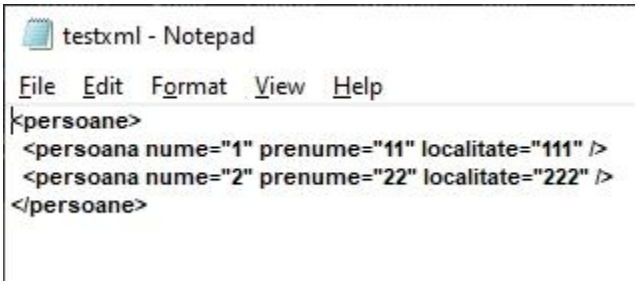
```
                xa.Value = scaracteristicilepersoaneicurente[0];
                //adaugam acest Atribut la elementul "xnpersoana" curent
                //din documentul XML "xd_persoane":
                //
                xnpersoana.Attributes.Append(xa);
                //
                //procedam similar cu celelalte doua particule de informatie
                //din secventa de informatii despre persoana curenta:
                //
                //un Atribut pentru prenume:
                //--- desigur --- presupunand --- ca a doua particula este prenumele:
                //aceasta particula se afla la
                //indexul 1 in tabloul "scaracteristicilepersoaneicurente:
                //
                //variabila "xa" este deja definita rin XmlAttribute xa =...
                //astfel ca de acum inainte doar ii atribuim alta valoare:
                //
                xa = xd_persoane.CreateAttribute("prenume");
                xa.Value = scaracteristicilepersoaneicurente[1];
                xnpersoana.Attributes.Append(xa);
                //
                //
                //un Atribut pentru localitate:
                //--- desigur --- --- presupunand --- ca a treia particula este localitatea:
                //aceasta particula se afla la
                //indexul 2 in tabloul "scaracteristicilepersoaneicurente":
                //
                //variabila "xa" este deja definita rin XmlAttribute xa =...
                //astfel ca de acum inainte doar ii atribuim alta valoare:
                //
                xa = xd_persoane.CreateAttribute("localitate");
                xa.Value = scaracteristicilepersoaneicurente[2];
                xnpersoana.Attributes.Append(xa);
                //
                //
                //
                //
                //odata definite toate Atributele elementului "xnpersoana"
                //de inregistrat in fisierul XML,
                //adaugam acest element "xnlpersoana"
                //la elementul-radacina "xn_radacina" (cel de mai sus, cu textul "persoane")
                //al documentului XML "xd_persoane":
                //
                xn_radacina.AppendChild(xnpersoana);
                //
                xd_persoane.AppendChild(xn_radacina);
        }
}
```

```
        //^ am incheiat iterarea prin elementele din StringCollection
        //pe care astfel le-am plasat in documentul XML xd_persoane
        //pe care il vom salva/scrie intr-un fisier XML
        //
        //
        //solicitam numele fisierului XML in care salvam documentul XML "xd_persoane":
        if (sfd.ShowDialog() != DialogResult.Cancel)
        {
                xd_persoane.Save(sfd.FileName);
                //
                //se observa cum pentru a scrie in fisier textul "formatat" XML
                //nu este folosit vreun obiect de tipul System.IO.StreamWriter,
                //ci obiectul de tip XmlDocument se ocupa de operatiunile Input/Output:
                //deschiderea, scrierea si inchiderea fisierului
        }
}
```

După ce am adăugat la *StringCollection* două secvenţe de informaţie, completând deci formularul pentru două /persoane şi înainte de a o înregistra pe a treia, programul ar putea arăta astfel:
Se observă că butonul pentru salvarea în fişier XML este activ, Proprietatea *Enabled* = true:



După salvarea într-un fişier XML, conţinutul acestuia ar fi asemănător celui din imaginea următoare unde fişierul a fost deschis cu *Notepad*:



De-acum, când vom vrea să recuperăm/accesăm /extragem o parte din conţinutul fişierului astfel salvat – de exemplu, după ce am plasat pe alt computer:

- şi programul ca fişier executabil ("*denumireaprogramuluioriaproiectuluisauasolutiei.exe*")
- şi fişierul XML cu informaţii

va trebui să folosim obiecte, entităţi specifice acestui scop şi care se află
tot în spaţiul de nume *System.Xml*.

Acest exemplu, unde s-au folosit:

- ***metodologia prezentată***

De obicei în limbajele de programare de nivel înalt,
oricare operaţiune poate fi efectuată în mai multe feluri.

Metodologia prezentată aici,
unde se foloseşte un obiect StringCollection
populat cu String-uri care pot fi secţionate în particule distincte
care sunt trimise în tablouri de String-uri
este una consumatoare de memorie. incomodă şi neelegantă
şi este adecvată numai pentru scopul prezentării unui mod
în care folosind noţiuni simple se poate ajunge la un rezultat anume

- ***un număr mic de Elemente XML – numai trei elemente***

nu este unul bun pentru cazurile când trebuie prelucrate, de exemplu,
mai mult de 100 secvenţe de informaţie
şi când astfel vor fi folosite alte metode –
mai rapide, mai economice cu memoria şi posibil mai elegante.

Fiindcă avem la dispoziţie fişierul cu toate secvenţele de informaţie înregistrate mai devreme, vrem ca:

- să citim pe rând, de la primul la ultimul,
  cele trei Elemente XML unde se află secvenţele de informaţie pe care
- să le afişăm în casetele *TextBox* corepunzătoare unde, însă, nu vor putea fi modificate
- odată afişată ultima secvenţă de informaţie, să se revină la prima

Pentru citirea Elementelor XML de interes, ori a Nodurilor din structura arborescentă a fişierului/formatului XML vom folosi obiectul:

XMLNodeList

În acest obiect/variabilă pot fi plasate Elementele/Nodurile XML care îndeplinesc o anumită condiţie sa/şi care se află pe o anumită cale (*Path*)/ramură în structura arborescentă a documentului *XMLDocument* creat dintr-un fişier XML.

Astfel, după separarea Elementelor/Nodurilor de interes se poate efectua citirea Atributelor sau a altoe caracteristici ale fiecăruia.

Şi chiar asta intenţionăm să facem:
- să extragem Elementele/Nodurile numite "persoana" din fişierul XML salvat anterior şi posibil pe alt computer
- să ciclăm prin acea colecţie de Elemente/Noduri XML încât
- să afişăm fiecare particulă de informaţie stocată în Atributele acelui Element

Înainte de asta trebuie să citim conţinutul fişierului XML cu informaţiile de interes.

Pentru a cicla prin Elementele de interes separate din acel fişier:
Declarăm o variabilă de tip *XmlNodeList* pe acelaşi nivel cu declaraţia:
StringCollection scinfo_persoane_memorie = new StringCollection();

... anume:

XmlNodeList xnl_elementeleincarcatedinfisierulxml;

Mai departe, fiindcă informaţiile asociate unui Element/Nod XML vor fi afişate la apăsarea unui buton, trebuie să ţinem cumva evidenţa poziţiei în acea colecţie de Elemente/Noduri XML a celui curent:
Aşa că declarăm o variabilă de tip numeric întreg unde să fie reţinută acea poziţie curentă:

int i_poselemxml = 0;

Acum, variabilele declarate înainte de orice alt cod se află pe acelaşi nivel cu metoda:
public Form1()
...

... anume:

public Form1()
...

StringCollection scinfo_persoane_memorie = new StringCollection();
XmlNodeList xnl_elementeleincarcatedinfisierulxml;
int i_poselemxml = 0;

Începem să scriem cod.

Întâi, codul pentru citirea fişierului XML care conţine Elementele/Nodurile de interes,
numite aici "persoana":

Înapoi în modul de proiectare/Design al *Form1*: tastele Shift + F7

Dublu-click pe butonul " bfisierxmlincarcare":
Metoda asociată Evenimentului *Click* al său va fi astfel:

```
private void bfisierxmlincarcare_Click(object sender, EventArgs e)
{
	if (ofd.ShowDialog() != DialogResult.Cancel)
	{
		XmlDocument xd_incarcatinmemorie = new XmlDocument();
		xd_incarcatinmemorie.Load(ofd.FileName);
		try
		{
			//Elementele cu numele "persoana" pot fi accesate cu comanda
			//SelectNodes a oricarui obiect XmlDocument, precedate de "//":
			//
			xnl_elementeleincarcatedinfisierulxml =
				xd_incarcatinmemorie.SelectNodes("//persoana");
			//
			//afisam numarul Elementelor astfel identificate:
//
			MessageBox.Show("In fisierul XML selectat:\r\n\r\n" +
				ofd.FileName +
				"\n\r\nau fost identificate:\r\n" +
				xnl_elementeleincarcatedinfisierulxml.Count.ToString() +
				"\r\nelemente de interes");
			//
			//activam butonul de prezentare a secventelor de informatie
			//stocate in Atributele Elementele/Nodurile XML de interes ("persoana")
			//tocmai citite daca numarul acelor Elemente/Noduri este nenul:
			//
			bciclareprinelemxmldinfisier.Enabled =
				(xnl_elementeleincarcatedinfisierulxml.Count > 0);
		}
		catch (Exception ex)
		{
			MessageBox.Show("Eroare la citirea continutului fisierului XML:\r\n\r\n" +
				ofd.FileName +
				"\r\n\r\n" +
				ex.ToString(),
				"EROARE");
		}
	}
}
```

Acum scriem codul pentru:

- ciclarea prin Elementele XML identificate/separate/preluate
  din fişierul XML selectat şi pentru
- ***afişarea Valorilor Atributelor sale, Atribute ale căror nume le cunoaştem şi astfel putem plasa fiecare astfel de Valoare în TextBox-ul corespunzător***:

Dublu-click pe butonul "bciclareprinelemxmldinfisier":
În metoda asociată Evenimentului *Click* al său scriem codul următor (continuă pe pagina următoare):
Unde comentariile detaliază instrucţiuni:

```
private void bciclareprinelemxmldinfisier_Click(object sender, EventArgs e)
{
        //apasam acest buton pentru a afisa in TextBox-urile corespunzatoare
        //particulele de informatie din fiecare astfel de secventa
        //stocata in Atributele fiecarui Element XML
        //
        //astfel, intentionand a citi urmatorul Element XML,
        //verificam daca s-a ajuns la numarul Elementelor de interes
        //stocate acum in acel obiect XmlNodeList:
        //daca --- acum, la aceasta apasare pe buton --- este afisat ultimul Element,
        //atunci valoarea/pozitia se reseteaza la 0 – se revine la primul Element:
        //
        if (i_poselemxml == xnl_elementeleincarcatedinfisierulxml.Count)
        {
                i_poselemxml = 0;
        }
        //
        //citim Atributele Elementului de la aceasta pozitie in XmlNodeList:
        //
        if (i_poselemxml < xnl_elementeleincarcatedinfisierulxml.Count)
        {
                //primul Element: indexul 0; al doilea: indexul 1; al treilea: indexul 2 samd.
                //
                //retinem Valorile Atributelor acestui Element in String-uri:
                //aceasta NU este necesar; ocupam degeaba memorie cu un String duplicat
                //DAR este util pentru programator, pentru a nu scrie
                //iar si iar ce e in dreapta semnului =
                //
                string snume_cititdinelemxmlinfisier =
                xnl_elementeleincarcatedinfisierulxml[i_poselemxml].Attributes["nume"].Value;
                string sprenume_cititdinelemxmlinfisier =
                xnl_elementeleincarcatedinfisierulxml[i_poselemxml].Attributes["prenume"].Value;
                string slocalitate_cititdinelemxmlinfisier =
                xnl_elementeleincarcatedinfisierulxml[i_poselemxml].Attributes["localitate"].Value;
                //
                //afisam aceste Valori ale Atributelor in TextBox-urile corespunzatoare:
                //
```

```
                tbnume.Text = snume_cititdinelemxmlinfisier;
                tbprenume.Text = sprenume_cititdinelemxmlinfisier;
                tblocalitate.Text = slocalitate_cititdinelemxmlinfisier;
                //
                //pregatim citirea urmatorului Element:
                //incrementam cu 1 variabila de ciclare/iterare:
                //
                i_poselemxml++;
        }
}
```

Pornim programul: tasta F5

Selectăm pentru citirea Elementelor XML de interes
fişierul XML pe care l-am salvat anterior,
apoi solicităm afişarea informaţiilor despre fiecare din acestea.

Biblioteca XML

Vom crea un program pentru inventarul cărţilor din bibliotecă.
Biblioteca va putea fi:
- salvată într-un fişier XML
- încărcată dintr-un fişier XML pentru a o extinde şi salva iarăşi şamd.

Despre fiecare carte pe care vrem să o adăugăm la bibliotecă ştim:
- titlul
- autorul

Despre unele cărţi cunoaştem şi/sau anul ediţiei respective şi/sau editura şi/ sau traducătorul ş. a.

Informaţiile despre fiecare carte vor fi:
- furnizate de catre utilizatorul programului in casete *TextBox* de editare a textului. Acestea vor fi: titlul, autorul/autorii, opţional: anul ediţiei
- afişate pe o linie a unui Control de tipul *ListView*

Pe *Form1* plasăm aceste Controale:
- o *ListView*, redenumit la "lvcartiledinbiblioteca"
    - Stabilim Proprietatea sa "*ViewMode*" la "Details"
    - Adăugăm trei coloane (*Columns*), Text-ul lor fiind sugestiv pentru ce particulă de informaţie despre acea carte va fi pe acea coloană a tabelului din *ListView*: prima coloană va fi intitulată "Titlul cartii", a doua - "Autorul/Autorii cartii", a treia - "Anul editiei";

- un *Button*, redenumit la "bcartenoua_adaugarelalista" care va iniţia plasarea/adăugarea în ListView a informaţiilor despre o carte
    - Stabilim Proprietatea sa *Enabled* la: False. Va fi activ numai dacă utilizatorul a specificat titlul şi auorul cărţii de adăugat la ListView

- 3 *RadioButton* care vor indica cele trei TIPURI de cărţi
    - *Text*-ul unuia va fi "Meteorologie", al altuia "SF", al altuia "Aventuri" sau alte tipuri
    - Numele lor vor fi, respectiv:
        - "rbcartetip_meteo"
        - "rbcartetip_sf"
        - "rbcartetip_aventuri"
        -
- un *GroupBox* în care vom plasa aceste Controale de tipul *RadioButton*

- 3 *TextBox*-uri unde utilizatorul va scrie informaţiile (titlul, autorul/autorii, eventual şi anul ediţiei) despre cartea care va fi adăugată la lista din afişată în *ListView*
    - Numele lor vor fi sugestive pentru textul conţinut – anume, în acest exemplu:
        - "tbcartenoua_titlu"
        - "tbcartenoua_autor"
        - "tbcartenoua_ anuleditiei"

- un *MenuStrip* cu două intrări: "Program" şi "Biblioteca":
  - sub "Program" se pune opţiunea "Exit" sau un text care să indice închiderea programului. Redenumim aceasta la: "mi_gataexit"
  - sub "Biblioteca" se pun opţiunile:
    - "Salvez intr-un fisier XML cartile din lista curenta" – redenumit la "miblio_fisier_salvare"
    - "Incarc biblioteca dintr-un fisier XML" – redenumit la "miblio_fisier_incarcare"

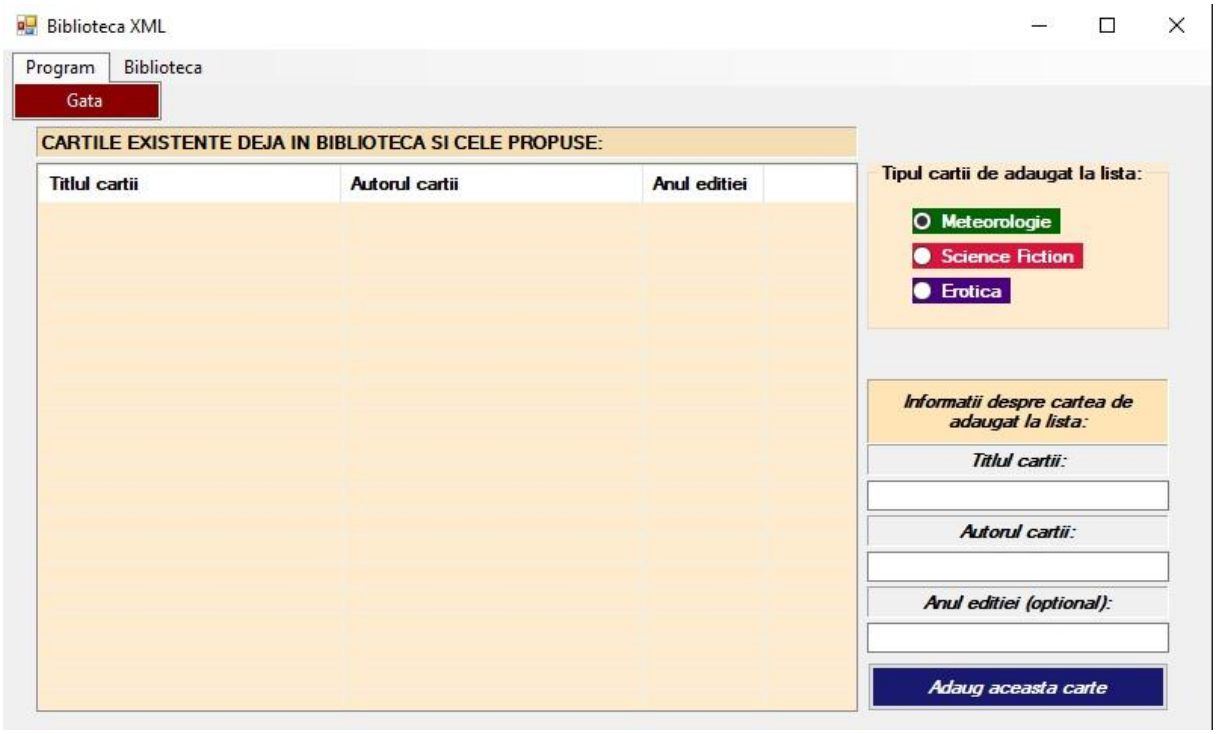
- un *SaveFileDialog* care va folosi la salvarea informaţiilor din *ListView* despre cărţi într-un fişier XML. Redenumim acest Control la: "sfd_fisierulbiblio_salvare"

- un *OpenFileDialog* care va folosi la încărcarea dintr-un fişier XML a informaţiilor despre cărţi care vor fi afişate în *ListView*. Redenumim acest Control la: " ofd_fisierulbiblio_deschidere"

Întrucât vom folosi entităţi XML, la secvenţa **using** de la începutul codului Form1 adăugăm o referire la spaţiul de nume *System.Xml*:

using System.Xml;

Interfaţa (*Form1*) ar putea arăta astfel:



Să scriem codul programului.

Întâi creăm metoda pentru activarea butonului dedicat adăugării la *ListView* a informaţiilor despre o carte de adăugat la lista din *ListView*:

Pe acelaşi nivel cu metoda

public Form1()
...

scriem acest cod (*Copy* de aici – *Paste* în cod):

```
private void CarteNoua_ActivareaButonuluiDeInregistrare()
{
        //butonul pentru adaugarea la lista din ListView
        //a cartii propuse
        //va fi activ numai daca am specificat Titlul si Autorul cartii
        //
        //nu verificam daca s-a completat anul editiei
        //fiindca nu este necesara neaparat -
        //aceasta conditie poate fi adaugata la cele doua dintre parantezele
        //intre care evaluam celelalte conditii:
        //
        bcartenoua_adaugarelalista.Enabled =
                (
                (tbcartenoua_titlu.Text.Trim() != "")
                &
                (tbcartenoua_autor.Text.Trim() != "")
                );
}
```

Înapoi în mediul de proiectare/Design al *Form1* – tastele Shift + F7 – unde solicităm de la Visual Studio generarea metodei asociate Evenimentului *TextChanged* al *TextBox*-urilor unde vor fi scrise titlul şi autorul cărţii de adăugat la listă:

Dublu-click pe *TextBox*-ul pentru titlul cărţii
Şi în cod găsim metoda generată automat pentru Evenimentul *TextChanged* al acestui *TextBox*.

Înapoi în modul de proiectare/Design al *Form1*,
Şi în cod găsim metoda generată automat pentru Evenimentul *TextChanged* al acestui *TextBox*.

Aceste metode vor fi (*Copy* de aici – *Paste* în cod, desigur înlocuind metodele generate automat):
(observăm apelarea metodei de mai sus prin care se activează sau nu butonul de adăugare la listă)

```
private void tbcartenoua_titlu_TextChanged(object sender, EventArgs e)
{
        //se schimba Textul despre titlul cartii
        CarteNoua_ActivareaButonuluiDeInregistrare();
}

private void tbcartenoua_autor_TextChanged(object sender, EventArgs e)
{
        //se schimba Textul despre autorul cartii
        CarteNoua_ActivareaButonuluiDeInregistrare();
}
```

Scriem codul pentru adăugarea la lista din *ListView* a informaţiilor despre o carte:

Acele instrucţiuni vor fi asociate butonului dedicat acestei proceduri
(acest buton este cel albastru, din dreapta-jos a imaginii precedente)

Înapoi în modul de proiectare/Design al *Form1*:
Dublu-click pe acel buton, aici denumit "bcartenoua"

Metoda asociată Evenimentului său *Click* va fi aceasta (*Copy* de aici – *Paste* în cod):

```
private void bcartenoua_adaugarelalista_Click(object sender, EventArgs e)
{
        try
        {
                //am specificat Titlul si Autorul cartii de adaugat la lista din ListView
                //--- eventual si Anul Editiei ---
                //deci cream un obiect
                //ListViewItem
                //constand din doua particule de text necesare: Titlul, Autorul
                //sau din cele trei particule de text: Titlul, Autorul, Anul Editiei
                //care ListViewItem va fi trimis in ListView
                //
                ListViewItem lvi_cartenoua = new ListViewItem();
                lvi_cartenoua.Text = tbcartenoua_titlu.Text.Trim();
                lvi_cartenoua.SubItems.Add(tbcartenoua_autor.Text.Trim());
                //anul editiei este informatie optionala, dar ne asiguram
                //ca trimitem un text chiar nul ("") si in a 3-a coloana din ListView:
                //
                lvi_cartenoua.SubItems.Add(tbcartenoua_anuleditiei.Text.Trim());
                //
                //ca informatie despre intregul acest ListViewItem
                //stabilim valoarea Proprietatii sale Tag
                //la TIPUL cartii pe care tocmai o adaugam:
                //anume, in functie de care RadioButton am bifat:
                //in fisierul XML, acestea vor fi ramurile pe care se vor afla
                //Elementele-cartile individuale:
                //
                if (rbcartetip_meteo.Checked == true)
                {
                        lvi_cartenoua.Tag = "meteorologie";
                }
                else if (rbcartetip_sf.Checked == true)
                {
                        lvi_cartenoua.Tag = "sf";
                }
                else if (rbcartetip_aventuri.Checked == true)
                {
                        lvi_cartenoua.Tag = "aventuri";
                }
                //
                //trimitem acest ListViewItem in ListView:
                //
                lvcartiledinbiblioteca.Items.Add(lvi_cartenoua);
                //
                //curatam TextBox-urile de ceea ce am scris in ele:
                //
                tbcartenoua_titlu.Clear();
                tbcartenoua_autor.Clear();
                tbcartenoua_anuleditiei.Clear();
        }
```

```
        catch (Exception ex)
        {
                MessageBox.Show("Eroare la adaugarea la lista a cartii propuse:\r\n\r\n" +
                                        ex.ToString(),
                                        "EROARE");
        }
}
```

Să scriem codul pentru salvarea într-un fişier XML a listei cu informaţii despre cărţile astfel adăugate:
Acest cod va fi invocat la selectarea intrării de meniu respective din meniul "Biblioteca":

Înapoi în modul de proiectare/Design al *Form1*:

Dublu-click pe intrarea de meniu având Numele "miblio_fisier_salvare":
Apoi, în cod, înlocuim metoda generată automat pentru Evenimentul *Click* al său cu:
(urmează trei pagini cu codul acestei metode):

```
private void miblio_fisier_salvare_Click(object sender, EventArgs e)
{
	//aceasta intrare de meniu:
	//"Biblioteca -> Salvez intr-un fisier XML cartile din lista curenta"
	//initiaza salvarea intr-un fisier XML informatiile despre cartile afisate in ListView
	//
	//
	//folosim SaveFileDialog-ul de pe Form1
	//pentru a specifica in ce fisier vrem sa salvam lista curenta:
	//
	//intai putem restabili Proprietatea Filter a SaveFileDialog-ului:
	sfd_fisierulbiblio_salvare.Filter = "Fisiere XML|*.xml";
	//
	//de obicei,
	//cand --- se stie --- ca Filter nu va fi schimbata in timpul rularii programului
	//aceasta Proprietate se stabileste:
	//- fie pe un nivel superior instructiunilor unde va fi folosita
	//- fie in metoda Evenimentului Load al Form-ei care contine acel SaveFileDialog
	//
	//in acest exemplu, ne-am asigurat ca Filter este cel necesar
	//pentru accesarea --- implicit --- numai a fisierelor cu extensia .xml
	//
	//se solicita numele fisierului XML in care va fi salvata lista curenta de carti:
	if (sfd_fisierulbiblio_salvare.ShowDialog() != DialogResult.Cancel)
	{
		//incepem a lucra cu fisiere
		//deci plasam toate acele instructiuni in bloc try - catch:
		try
		{
			//initializam un Document XML in care
			//vom plasa ca Elemente/Noduri XML
			//cartile detaliate in ListView:
			//
			XmlDocument xd_biblioteca = new XmlDocument();
			//
			//cream elementul-radacina al structurii XML arborescente
			//pe care vom plasa Elementele despre carti:
			XmlNode xn_radacina = xd_biblioteca.CreateElement("biblioteca");
			xd_biblioteca.AppendChild(xn_radacina);
			//
			//pregatim tot atatea Elemente/Noduri XML
			//cate tipuri de carti sunt indicate de cele (3) RadioButton:
			//se presupune ca stim cate astfel de tipuri sunt
			//si cum sa le denumim in consecinta:
			//
			XmlNode xn_carte_tip = xd_biblioteca.CreateNode(
									XmlNodeType.Element,
									"meteorologie",
									"");
```

```
//adaugam acest nod la radacina Documentului XML:
xn_radacina.AppendChild(xn_carte_tip);
xn_carte_tip = xd_biblioteca.CreateNode(XmlNodeType.Element, "sf", "");
//adaugam acest nod la radacina Documentului XML:
xn_radacina.AppendChild(xn_carte_tip);
xn_carte_tip = xd_biblioteca.CreateNode(
                                                XmlNodeType.Element,
                                                "aventuri",
                                                "");
//adaugam acest nod la radacina Documentului XML:
xn_radacina.AppendChild(xn_carte_tip);
//
//
//parcurgem lista de carti detaliate in ListView
//extragand TIPUL fiecarei carti
//din Proprietatea Tag a fiecarui ListViewItem:
//
//vom crea Elemente/Noduri XML
//toate denumite la fel: "carte"
//dar fiecare avand Atribute diferite
//(desi pot fi inregistrate Elemente/Noduri identice)
//
for (
        int ilvitem_carte = 0;
        ilvitem_carte < lvcartiledinbiblioteca.Items.Count;
        ilvitem_carte++
        )
{
        //retinem intr-o variabila de tipul ListViewItem
        //linia curenta din ListViewItem
        //adica, ListViewItem-ul de la aceasta pozitie in iteratie:
        //
        ListViewItem lvi_cartedinlista =
                                lvcartiledinbiblioteca.Items[ilvitem_carte];
        //
        //TIPUL cartii acesteia se afla in
        //Proprietatea Tag a acestui ListViewItem:
        //in functie de String-ul din Tag cream un Element/Nod XML
        //pe care il vom plasa/pe ramura denumita corespunzator
        //din Documentul XML:
        //
        //cream un Element XML denumit "carte"
        XmlNode xn_cartedeadaugatinbiblioteca =
                        xd_biblioteca.CreateNode(XmlNodeType.Element,
                        "carte",
                        "");
```

```
                        //cream Atributele acestui Element XML:
                        //titlul, autorul, anul editiei (acesta poate fi nul)
                        //
                        //adaugam un Atribut a carui Value va fi titlul cartii:
                        XmlAttribute xa = xd_biblioteca.CreateAttribute("titlu");
                        xa.Value = lvi_cartedinlista.Text;  //textul de pe coloana 1
                        xn_cartedeadaugatinbiblioteca.Attributes.Append(xa);
                        //adaugam un Atribut a carui Value va fi autorul cartii:
                        xa = xd_biblioteca.CreateAttribute("autor");
                        xa.Value = lvi_cartedinlista.SubItems[1].Text;
                        //^ textul de pe coloana 2
                        xn_cartedeadaugatinbiblioteca.Attributes.Append(xa);
                        xa = xd_biblioteca.CreateAttribute("anuleditiei");
                        xa.Value = lvi_cartedinlista.SubItems[2].Text;
                        //^ textul de pe coloana 3
                        xn_cartedeadaugatinbiblioteca.Attributes.Append(xa);
                        //
                        //acum plasam Elementul XML "carte" avand aceste Atribute
                        //in Elementul XML care denota TIPUL cartii
                        //si care acelasi text cu Tag-ul acestui ListViewItem:
                        //
                        //deci nu specificam explicit Tag-ul, de exemplu "meteorologie"
                        //ci ne prefacem ca nu stim cum se numeste
                        //asa ca il preluam pur si simplu pentru plasarea
                        //sub Elementul XML avand acelasi text:
                        //
                        xd_biblioteca.SelectSingleNode(
                                "//" +
                                lvi_cartedinlista.Tag.ToString())
                                        .AppendChild(xn_cartedeadaugatinbiblioteca);
                    }
                    //
                    //
                    //acum salvam Documentul XML in fisierul specificat in SaveFileDialog:
                    xd_biblioteca.Save(sfd_fisierulbiblio_salvare.FileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Eroare la salvarea in fisier XML "+
                                    "a listei curente de carti:\r\n\r\n" +
                                    ex.ToString(),
                                    "EROARE");
                }
            }
        }
```

Să scriem codul pentru încărcarea-afişarea informaţiilor despre cărţi salvate cu acest program într-un fişier XML:
Acest cod va fi invocat la selectarea intrării de meniu respective din meniul "Biblioteca":

Înapoi în modul de proiectare/Design al *Form1*:
Dublu-click pe intrarea de meniu având Numele "miblio_fisier_incarcare":
Apoi, în cod, înlocuim metoda generată automat pentru Evenimentul *Click* al său cu:
(urmează două pagini cu codul acestei metode):

```
private void miblio_fisier_incarcare_Click(object sender, EventArgs e)
{
	//aceasta intrare de meniu:
	//"Biblioteca -> Incarc biblioteca dintr-un fisier XML"
	//initiaza incarcarea dintr-un fisier XML salvat in alta sesiune a acestui program
	//si care contine informatiile afisabile in ListView despre cartile existente in biblioteca
	//si initiaza afisarea lor in ListView
	//
	//
	//folosim OpenFileDialog-ul de pe Form1
	//pentru a specifica din ce fisier vrem sa incarcam biblioteca:
	//
	//intai putem restabili Proprietatea Filter a OpenFileDialog-ului:
	ofd_fisierulbiblio_deschidere.Filter = "Fisiere XML|*.xml";
	//
	//pentru Proprietatea Filter se aplica observatiile de la SaveFileDialog-ul de mai sus
	//
	//se solicita numele fisierului XML din care va fi incarcata biblioteca:
	if (ofd_fisierulbiblio_deschidere.ShowDialog() != DialogResult.Cancel)
	{
		//intai curatam ListView:
		//stergem toate inregistrarile - ListViewItem-urile:
		//altfel, ceea ce va fi citit din fisier se va adauga la lista curenta:
		//
		lvcartiledinbiblioteca.Items.Clear();
		//
		//incepem a lucra cu fisiere
		//deci plasam toate acele instructiuni in bloc try - catch:
		try
		{
			//initializam un Document XML in care
			//vom incarca Elementele/Nodurile XML
			//din biblioteca aflata in fisierul XML selectat:
			//
			XmlDocument xd_biblioteca = new XmlDocument();
			//
			//populam acest Document XML cu continutul fisierului XML selectat:
			xd_biblioteca.Load(ofd_fisierulbiblio_deschidere.FileName);
			//
			//din acest Document XML selectam numai
			//Elementele/Nodurile XML denumite "carte":
			//
			//folosim un obiect de tipul XmlNodeList
			//in care vor fi plasate acele Elemente:
			XmlNodeList xnl_carti_toate_din_fisier =
                                                    xd_biblioteca.SelectNodes("//carte");
			//
```

```
                    //parcurgem aceasta colectie XmlNodeList de Elemente/Noduri XML
                    //pentru a le:
                    //- plasa in ListView: titlul, autorul, anul editiei
                    //- recupera Tag-ul din Elementul-parinte asa cum este in fisierul XML:
                    //de exemplu, Elementul avand textul "meteorologie":
                    //
                    //atentie la:
                    //--- ORDINEA in care sunt adaugate SubItem-urile
                    //(textele de pe coloanele cutare din ListView)
                    //--- si SEMNIFICATIA lor care trebuie sa corespunda
                    //Valorilor Atributelor fiecarui Element XML citit
                    //
                    //de exemplu,
                    //Coloana 1 poate fi intitulata "ISBN" si Coloana 2 "Numarul Editiei"
                    //
                    foreach (XmlNode xn_carte_din_fisier in xnl_carti_toate_din_fisier)
                    {
                        ListViewItem lvi_carte_de_plasat_in_lista = new ListViewItem();
                        lvi_carte_de_plasat_in_lista.Tag =
                                            xn_carte_din_fisier.ParentNode.Name;
                        lvi_carte_de_plasat_in_lista.Text =
                                            xn_carte_din_fisier.Attributes["titlu"].Value;
                        lvi_carte_de_plasat_in_lista.SubItems
                            .Add(xn_carte_din_fisier.Attributes["autor"].Value);
                        lvi_carte_de_plasat_in_lista.SubItems
                            .Add(xn_carte_din_fisier.Attributes["anuleditiei"].Value);
                        //
                        //odata ce am creat acest ListViewItem
                        //corespunzand unei carti detaliate in fisierul XML
                        //adaugam acest ListViewItem la ListView:
                        //
                        lvcartiledinbiblioteca.Items.Add(lvi_carte_de_plasat_in_lista);
                        //
                        //
                        //odata incarcata o astfel biblioteca dintr-un fisier XML
                        //unde fusese salvata in alta sesiune a acestui program
                        //poate fi adaugita, salvata iarasi, incarcata iarasi samd.
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Eroare la salvarea in fisier XML "+
                        "a listei curente de carti:\r\n\r\n" + ex.ToString(),
                        "EROARE");
                }
            }
}
```

Acum scriem şi codul asociat intrării de meniu care închide programul: "Program -> Gata"

```
private void mi_gataexit_Click(object sender, EventArgs e)
{
        //programul va fi inchis
        //
        //nu mai cerem confirmare cu un MessageBox:
        //
        Application.Exit();
}
```

Biblioteca – bază de date SQL Server

Am aflat cum poate fi creat un inventar (în lipsa unui termen mai bun) al unor entităţi/obiecte cu diverse proprietăţi – inventar care constă exclusiv din text şi care inventar poate fi:

- salvat într-un fişier XML formatat corespunzător acelor obiecte
- încărcat pentru reutilizare

Această abordare este adecvată unor colecţii relativ mici (de ordinul a zeci şi sute) de obiecte.

Când, însă, avem de-a face cu colecţii de mii până la milioane de elemente, putem trece la pasul următor în gestionarea/administrarea unor astfel de colecţii – anume, la stocarea lor în ***baze de date***.

Un fişier XML este
mai portabil şi
mult mai uşor accesibil (conţinând text simplu)
decât o bază de date conţinând aceeaşi informaţie - în format text

Conceptul de Bază De Date,
la fel cu informaţii despre
sistemul de administrare SQL Server a bazelor de date,
crearea şi structura unei baze de date,
popularea sa cu informaţie şi despre
accesarea informaţiei conţinute
sunt expuse cu exemple în documentului distinct, asociat celui de faţă.

Vrem să facem inventarul cărţilor din biblioteca proprie.

Aceasta înseamnă a înregistra o parte din aceste informaţii despre fiecare carte:

- titlul - aceasta este o informaţie neapărat necesară despre o carte
- autorul sau autorii – aceeaşi observaţie ca pentru titlu, fiindcă pot exista cărţi cu acelaşi titlu scrise de autori diferiţi
- domeniul, sau genul, sau tipul
- anul ediţiei – informaţie opţională
- codul ISBN al ediţiei respective – opţional
- editura la care a apărut ediţia respectivă - opţional
- traducătorul sau traducătorii, dacă este cazul – deci, opţional
- citate – opţional

Numărul cărţilor poate fi de câteva sute sau de câteva mii sau mai multe,
astfel că ne propunem ca pentru a le inventaria să folosim o bază de date.

Întrucât bazele de date şi folosirea lor fie şi ca strictul necesar au fost prezentate,
vom folosi sistemul de administrare a bazelor de date (*RDBMS*-ul) prezentat – anume, SQL Server.

RDBMS –
Relational **D**ata**b**ase **M**anagement **S**ystem – Sistem de administrare a bazelor de date relaţionale

Acest sistem poate fi accesat cu programe create şi compilate sub platforma .NET.

Astfel, din Visual Studio poate fi accesat spaţiul de nume:

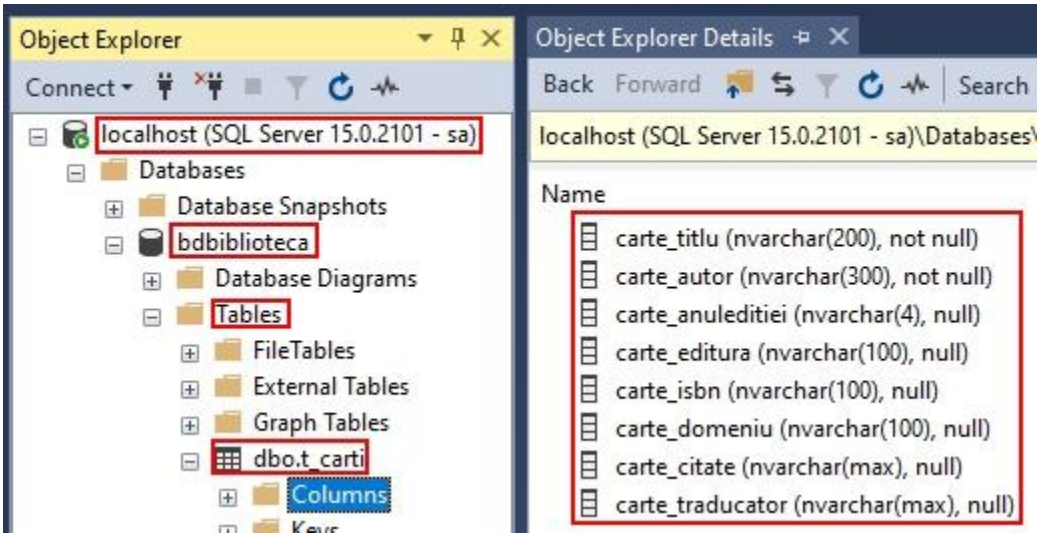
System.Data.SqlClient

ai cărui membri asigură accesul la un RDBMS.

În documentul care prezintă bazele de date şi operaţiunile comune efectuate cu ele
se arată şi cum putem pregăti baza de date care va susţine inventarul bibliotecii noastre.

Anume, folosind programul SQL Server Management Studio (SSMS)
pe care l-am instalat după instalarea sistemului SQL Server,
am creat o bază de date numită "*bdbiblioteca*".
În aceasta se află tabelul numit "*t_carti*"
ale cărui coloane/câmpuri se numesc (desigur, fără ghilimelele de aici):

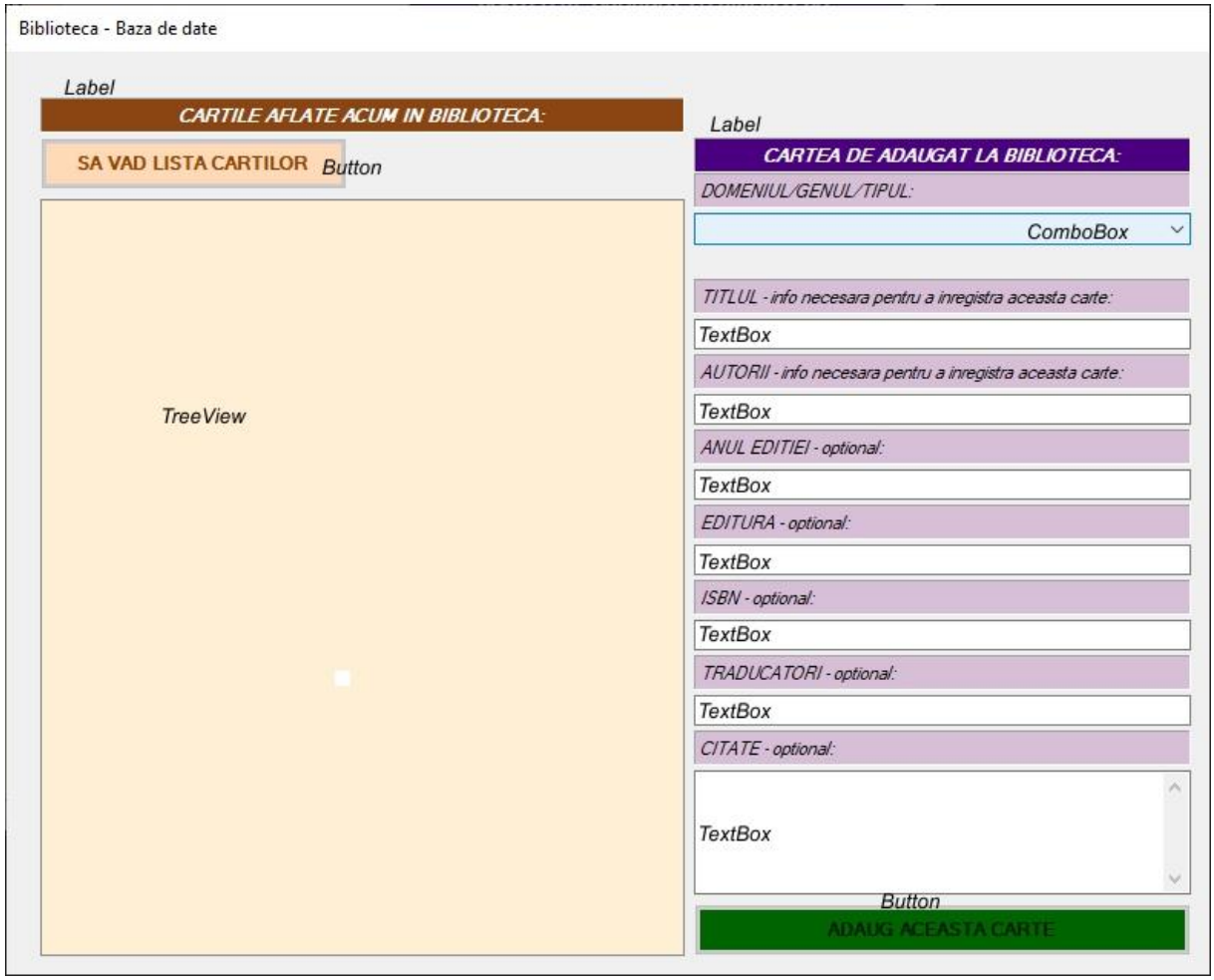
- "carte_titlu"
- "carte_autor"
- "carte_anuleditiei"
- "carte_editura"
- "carte_isbn"
- "carte_domeniu"
- "carte_citate"
- "carte_traducator"



Să începem construirea programului nostru.

Pornim Visual Studio, *File -> New... Project...*

Proiectăm *Form1* astfel încât să conţină Controalele ilustrate în imaginea următoare:
(se observă cum butoanele din *ControlBox* din dreapta-sus lipsesc: Proprietatea *ControlBox* = False)



Această interfaţă asigură strictul necesar pentru:

- înregistrarea informaţiilor despre fiecare carte de adăugat la cele din baza de date
- prezentarea cărţilor aflate în baza de date

Anume, în dreapta este zona dedicată înregistrării unei cărţi noi:

- câte un *TextBox* pentru scrierea textului de trimis în câmpul corespunzător din tabelul unde vor fi înregistrate informaţiile respective despre fiecare carte
    - Proprietatea *Name* a *TextBox*-urilor o stabilim astfel:
    - *TextBox*-ul unde vom scrie titlul cărţii: **tbcartenoua_titlu**
    - *TextBox*-ul unde vom scrie autorul/autorii cărţii: **tbcartenoua_autor**
    - *TextBox*-ul unde vom scrie anul ediţiei cărţii: **tbcartenoua_anuleditiei**
    - *TextBox*-ul unde vom scrie editura unde a fost publicat ă cartea: **tbcartenoua_editura**
    - *TextBox*-ul unde vom scrie codul ISBN al cărţii: **tbcartenoua_isbn**
    - *TextBox*-ul unde vom scrie traducătorul/traducătorii cărţii: **tbcartenoua_traducator**
    - *TextBox*-ul unde vom scrie citate din carte: **tbcartenoua_citate**
- o casetă *ComboBox* care iniţial va conţine domeniile/genurile/tipurile de cărţi din care vom alege unul singur pentru fiecare carte
    - redenumim această casetă la: **cbcartenoua_tip**
- un *Button* pentru iniţierea înregistrării; implicit, acesta are Proprietatea *Enabled* = False. Va fi activat sau dezactivat după cum sunt specificate: **Titlul** şi **Autorii** unei cărţi, adică în timp ce textul din *TextBox*-urile respective se schimbă şi astfel este invocat Evenimentul *TextChanged*
    - redenumim acest Button la: **bcartenoua_adaugare**

În stânga se află aceste Controale:

- un *TreeView* în care vor fi prezentate unele informaţii despre cărţile din baza de date cu biblioteca
    - redenumim acest *TreeView* la: **tvbiblioteca**
- un *Button* pentru iniţierea solicitării acelor informaţii
    - redenumim acest Button la: **bafisarebiblioteca**

Bineînţeles, la *Form1* poate fi adăugat un *MenuStrip* cu intrări pentru:

- închiderea (cunoscută ca "Exit"/"Close") programului: Application.Exit();
- un MessageBox cu informaţii despre (cunoscută ca "About") program

În spaţiul de nume **System.Data.SqlClient** se află membri ale căror proprietăţi şi (metode şi Evenimente) asociate oferă posibilitatea de a interacţiona cu un RDBMS care acceptă o astfel de comunicare –
există RDBMS-uri care au altă structură decât SQL Server şi cu care comunicarea se face în alt fel decât cu acesta).

Câţiva din principalii membri ai acestui spaţiu de nume şi de fapt cei pe care îi vom folosi în acest program sunt:

- **SqlConnection**
  - un obiect de acest tip administrează conexiunea la un server SQL:
    - o deschide, o închide, îi verifică starea ş. a
    - fiind iniţializzat cu un aşa-numit String de Conectare – anume, valoarea Proprietăţii *ConnectionString* a unui obiect/a unei variabile SqlConnection
      - String de Conectare în care programatorul trebuie să specifice:
        - **numele sau adresa IP a serverului SQL sau a computerului unde este instalat serverul SQL de interes**; computerul unde rulează atât programul, cât şi serverul SQL fiind referit prin **localhost** sau prin caracterul punct **.**
        - **numele de utilizator al serverului SQL**; *de regulă, administratorul are numele* **sa**
        - **parola de acces a acelui utilizator la serverul SQL**; parola administratorului "**sa**" este stabilită la instalarea SQL Server (de obicei şi pentru comoditate, parola administratorului este o parolă simplă şi uşor de reţinut, ca **sa123**.
        - **numele/denumirea bazei de date spre care iniţial se va deschide conexiunea**; această procedură este echivalentă directivei **USE *NumeleBazeiDeDate*;** dinaintea oricărei secvenţe de instrucţiuni SQL.

- **SqlCommand**
  - un obiect de acest tip conţine partea de comunicare propriu-zisă din partea utilizatorului şi trimisă către serverul SQL – anume, ce operaţiuni vrem să fie executate; în cel fel – selectare a unor înregistrări, modificarea lor sau a structurii înseşi a bazei de date ş.a.
    - **CommandText**: acesta este textul comenzii (instrucţiunii) sau comenzilor (instrucţiunilor) SQL pe care le trimitem serverului SQL. Poate fi orice secvenţă de instrucţiuni valide sintactic şi baza de date referită fie în *ConnectionString*, fie printr-o directivă USE *NumeleBazeiDeDate* înaintea acelor instrucţiuni.
    - **Connection**: acesta este obiectul de tip SqlConnection asociat acestui SqlCommand – aşadar, conexiunea la capătul căreia vor fi executate instrucţiunile din **CommandText**

Acum începem a scrie codul pentru înregistrarea în baza de date cu inventarul bibliotecii a informaţiilor despre o carte nouă.

În programul nostru, click-stânga pe *Form1* (selectarea sa, la fel cum selectăm orice Control)
şi apoi tasta **F7** pentru a merge în modul de scriere a codului acestei Form-e,
unde în secvenţa **using** de la începutul codului
adăugăm o instrucţiune prin care ne anunţăm intenţia de a folosi
membri ai spaţiului de nume **System.Data.SqlClient**:

```
using System.Data.SqlClient;
```

Acum, înapoi în modul de proiectare/design al *Form1*, unde pregătim câteva Controale:

Vrem ca la afişarea *Form1*, în caseta *ComboBox* cu domeniile/genurile/tipurile de cărţi
să se afle deja câteva astfel de elemente din care va putea fi ales unul pentru fiecare carte.

Aşadar, popularea casetei se va face de îndată ce *Form1* este afişată -
anume, la Evenimentul său implicit *Load*.

Tot la afişarea Form1, vrem să fie stabilite diverse Proprietăţi ale unor Controale
la care în cod, bineînţeles, ne vom referi cu numele *Name* al fiecăruia
şi ale căror Proprietăţi le vom accesa scriind caracterul punct . după acel nume.

Pentru aceasta:
Dublu-click pe *Form1*
Metoda *Form1_Load* o înlocuim cu:

- unde comentariile explică majoritatea instrucţiunilor:

```
private void Form1_Load(object sender, EventArgs e)
{
        //stabilim unele Proprietati ale acestei Form-e:
        //
        //titlul acestei Form-e:
        this.Text = "Biblioteca - Baza de date";
        //
        //vrem ca Form1 sa fie mereu deasupra altor programe cu interfata vizuala
        //pentru a sti ca programul acesta este pornit
        //incat sa nu pornim alta instanta a sa:
        this.TopMost = true;
        //
        //vrem ca Form1 sa nu poata fi inchis/a, minimiizata, maximizata:
        this.ControlBox = false;
        //Form1 va fi inchisa din comanda din meniul principal pentru aceasta operatiune
        //
        //
        //pentru procedura de ADAUGARE a unei carti la biblioteca:
        //
        //caseta ComboBox din care se va alege domeniul/genul/tipul cartii de adaugat:
        //stabilim cateva Proprietati ale sale:
        //(stim ca dintr-un Control de tipul ComboBox poate fi selectat numai un element)
        //
        //o populam cu cateva domenii/genuri/tipuri de carte:
        cbcartenoua_tip.Items.Add("Istorie");
        cbcartenoua_tip.Items.Add("Astronomie/Astrofizica/Cosmologie");
        cbcartenoua_tip.Items.Add("Filosofie");
        cbcartenoua_tip.Items.Add("Arta");
        cbcartenoua_tip.Items.Add("Casa/Atelier/Gradina");
        cbcartenoua_tip.Items.Add("Arhitectura");
        cbcartenoua_tip.Items.Add("Science Fiction");
        cbcartenoua_tip.Items.Add("Poezie");
        cbcartenoua_tip.Items.Add("Roman politist");
```

```
            cbcartenoua_tip.Items.Add("Matematica");
            cbcartenoua_tip.Items.Add("Enciclopedie");
            cbcartenoua_tip.Items.Add("Dictionar");
            //
            //ordonam alfabetic crescator elementele din ComboBox:
            cbcartenoua_tip.Sorted = true;
            //
            //stabilim Proprietatea DropDownStyle
            //incat textul elementului selectat sa NU fie editabil
            //si nici sa nu se poata scrie in ComboBox:
            cbcartenoua_tip.DropDownStyle = ComboBoxStyle.DropDownList;
            //
            //TextBox-ul unde va fi scris anul editiei cartii de adaugat:
            //stabilim Proprietatea MaxLength la 4:
            tbcartenoua_anuleditiei.MaxLength = 4;
}
```

Button-ul dedicat înregistrării unei cărţi va fi activ
numai dacă textul din casetele de editare *TextBox*
pentru **titlul** şi **autorul** cărţii
nu este nul în vreuna din aceste TextBox-uri,
aceste informaţii fiind esenţiale, ori minimumul necesar pentru a se putea înregistra o carte.

La început, la afişarea *Form1*, butonul va fi inactiv:
Click-stânga pe buton (selectare), apoi tasta **F4** (secţiunea Properties – Proprietăţile Controlului selectat)
Stabilim Enabled la False.

Acum solicităm de la Visual Studio generarea metodelor asociate Evenimentului implicit *TextChanged* al celor două *TextBox*-uri, pentru a verifica dacă există un text nenul:

Dublu-click pe *TextBox*-ul pentru **titlul** cărţii, ceea ce ne trimite în modul de scriere a codului, în metoda tocmai generată, asociată Evenimentului *TextChanged* al acestui *TextBox*.

Nu scriem nimic. Înapoi în modul de proiectare a *Form1*: tastele **Shift** şi **F7**

Dublu-click pe *TextBox*-ul pentru **autorul** cărţii, ceea ce ne trimite în modul de scriere a codului, în metoda tocmai generată, asociată Evenimentului *TextChanged* al acestui *TextBox*.

Pe acelaşi nivel cu cele două metode *TextChanged* scriem această metodă care va stabili starea butonului:

```
private void CarteDeAdaugatLaBiblioteca_ActivareaButonuluiDeAdaugare()
{
        //activam butonul de ADAUGARE a unei carti daca:
        //textul scris in TextBox-ul pentru TITLU
        //si cel  scris in TextBox-ul pentru AUTORI
        //sunt nenule:
        bcartenoua_adaugare.Enabled =
                (tbcartenoua_titlu.Text.Trim() != "") &
                (tbcartenoua_autori.Text.Trim() != "");
}
```

Acum apelăm această metodă din cele asociate Evenimentului *TextChanged* al celor două *TextBox*-uri – cea pentru **titlul** si cea pentru **autorul** cărţii:

```
private void tbcartenoua_titlu_TextChanged(object sender, EventArgs e)
{
        CarteDeAdaugatLaBiblioteca_ActivareaButonuluiDeAdaugare();
}

private void tbcartenoua_autori_TextChanged(object sender, EventArgs e)
{
        CarteDeAdaugatLaBiblioteca_ActivareaButonuluiDeAdaugare();
}
```

Acum, înapoi în modul de proiectare/design al *Form1*:

Solicităm metoda pentru Evenimentul implicit Click al butonului de adăugare a cărţii descrise:
Dublu-click pe butonul "*bcartenoua_adaugare*":

Înlocuim metoda generată automat cu:

```
private void bcartenoua_adaugare_Click(object sender, EventArgs e)
{
        //apasarea acestui buton
        //apeleaza metoda de inregistrare in baza de date
        //a informatiilor despre noua carte:
        //
        Biblioteca_Adaugare_Carte();
}
```

Aşadar, va fi apelată metoda **Biblioteca_Adaugare_Carte()** care urmează pe patru (4) pagini:

- unde comentariile explică majoritatea instrucţiunilor:

```
private void Biblioteca_Adaugare_Carte()
{
	//
	//pentru a indica faptul ca se efectueaza operatiuni teoretic de durata
	//stabilim Proprietatea Cursor a Form1 la: WaitCursor:
	//
	this.Cursor = Cursors.WaitCursor;
	//
	//cream o conexiune SQL la baza de date:
	SqlConnection sqlcon = new SqlConnection();
	//
	//textul de initializare a conexiunii cuprinde:
	//- numele serverului SQL; sau numele sau adresa IP a computerului pe care se afla serverul SQL
	//in acest caz, cand rulam acest program pe computerul unde este instalat SQL Server,
	//folosim "localhost" sau punctul .
	//- numele bazei de date catre care vor fi trimise comenzi SQL
	//in acest caz, baza de date este bdbiblioteca
	//- numele utilizatorului care va trimite comenzile SQL
	//folosim numele administratorului __serverului SQL__: sa
	//- parola acelui utilizator
	//folosim parola pe care am stabilit-o la instalarea serverului SQL, aici fiind: sa123
	//
	sqlcon.ConnectionString =
			"server=localhost;initial catalog=bdbiblioteca;user=sa;password=sa123";
	//
	//
	try
	{
		//construim comanda SQL
		//care adauga o inregistrare in tabelul cu informatii despre carti:
		//
		SqlCommand sqlcomm = new SqlCommand();
		//
		//inainte de orice, asociem acestei comenzi o conexiune valida:
		sqlcomm.Connection = sqlcon;
		//
		//
		string stextulcomenziideadaugare_carte = "USE bdbiblioteca;";
		//
		stextulcomenziideadaugare_carte += "INSERT t_carti";
		//
		stextulcomenziideadaugare_carte +=
			"(";
		//am inceput/deschis cu PARANTEZA ROTUNDA lista campurilor de completat
		//
		stextulcomenziideadaugare_carte += "carte_titlu,";
		stextulcomenziideadaugare_carte += "carte_autor,";
		stextulcomenziideadaugare_carte += "carte_anuleditiei,";
		stextulcomenziideadaugare_carte += "carte_editura,";
		//nu adaugam vreun cod ISBN, deci acest camp lipseste din aceasta lista
		stextulcomenziideadaugare_carte += "carte_domeniu,";
		stextulcomenziideadaugare_carte += "carte_citate,";
		stextulcomenziideadaugare_carte += "carte_traducator";
		//
```

```
stextulcomenziideadaugare_carte +=  ")";
//am inchis cu PARANTEZA ROTUNDA lista campurilor de completat
//
//formam clauza VALUES unde trimitem valorile campurilor de mai sus
//in aceeasi ordine, ori punand in corespondenta campul cu valoarea:
//
stextulcomenziideadaugare_carte += " VALUES";
//in clauza VALUES trimitem:
//VALORILE campurilor de completat asteptate de comanda SQL INSERT
//IN ORDINEA CORESPUNZATOARE denumirilor campurilor
//
//deschidem cu PARANTEZA ROTUNDA lista VALORILOR
//campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += "(";
//
//trimitem TITLUL noii carti: inlocuim apostrofurile ' din textul scris cu _ :
stextulcomenziideadaugare_carte +=
                    "'" + tbcartenoua_titlu.Text.Trim().Replace("'", "_") + "'";
//separam prin VIRGULA
//VALORILE campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ", ";
//
//trimitem AUTORII noii carti: inlocuim apostrofurile ' din textul scris cu _ :
stextulcomenziideadaugare_carte +=
                    "'" + tbcartenoua_autor.Text.Trim().Replace("'", "_") + "'";
//separam prin VIRGULA
//VALORILE campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ", ";
//
//trimitem ANUL EDITIEI noii carti: inlocuim apostrofurile ' din textul scris cu _ :
//fiindca putem scrie orice in acest TextBox, nu neaparat un an ca "2010":
stextulcomenziideadaugare_carte +=
                    "'" + tbcartenoua_anuleditiei.Text.Trim().Replace("'", "_") + "'";
//separam prin VIRGULA
//VALORILE campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ", ";
//
//trimitem numele EDITURII noii carti: inlocuim apostrofurile ' din textul scris cu _
stextulcomenziideadaugare_carte +=
                    "'" + tbcartenoua_editura.Text.Trim().Replace("'", "_") + "'";
//separam prin VIRGULA
//VALORILE campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ", ";
//
//trimitem DOMENIUL noii carti: inlocuim apostrofurile ' din textul scris cu _ :
//fiindca e posibil ca elementul selectat sa contina apostrofuri
//
```

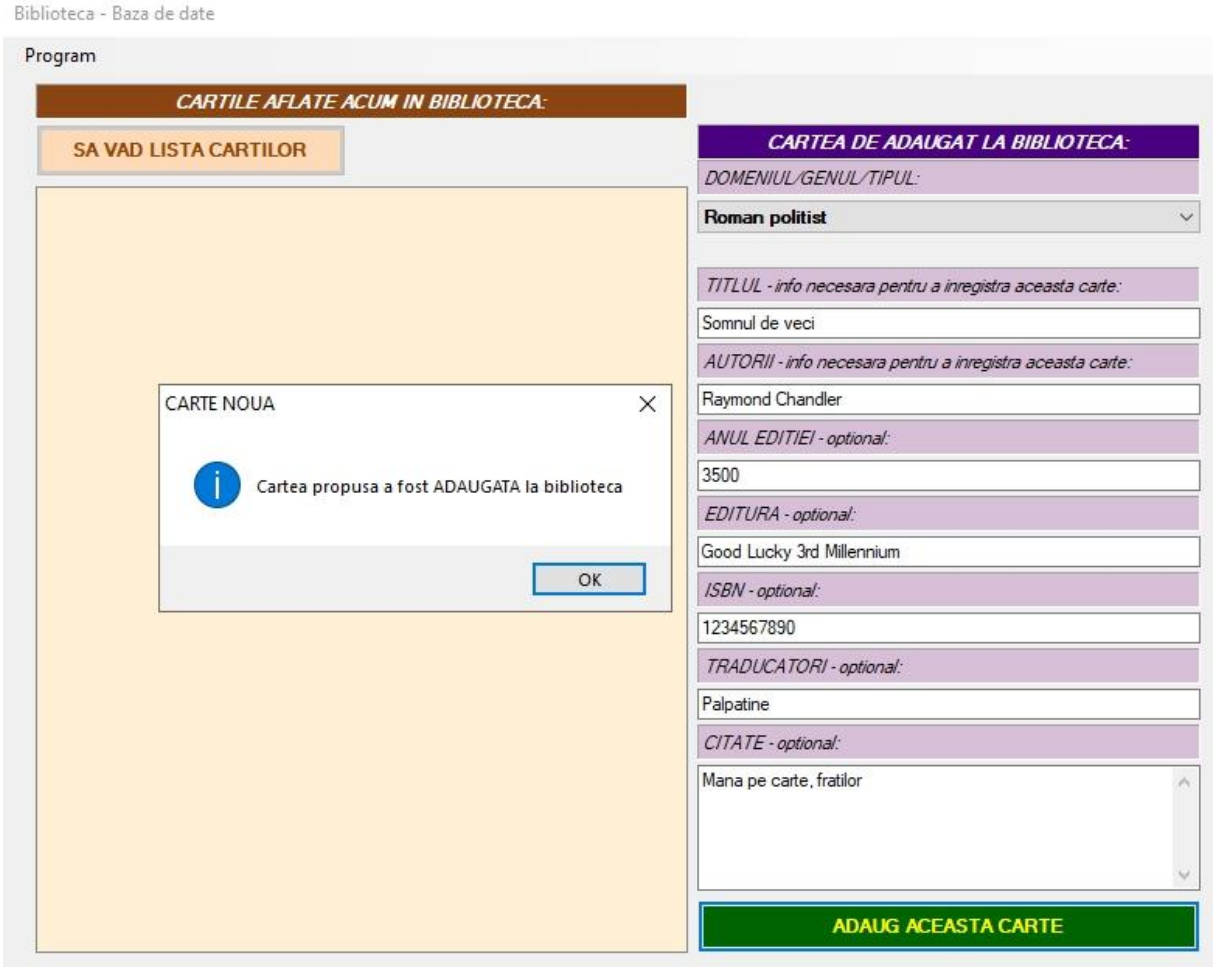
```
//intai verificam daca a fost selectat un domeniu
//daca nu, atunci trimitem textul nul "":
if (cbcartenoua_tip.SelectedIndex > -1)
{
        stextulcomenziideadaugare_carte +=
        "'" + cbcartenoua_tip.SelectedItem.ToString().Trim().Replace("'", "_") + "'";
}
else
{
        stextulcomenziideadaugare_carte += "'" + "" + "'";
}
//separam prin VIRGULA
//VALORILE campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ", ";
//
//trimitem CITATELE din noua carte: inlocuim apostrofurile ' din textul scris cu _ :
stextulcomenziideadaugare_carte +=
                        "'" + tbcartenoua_citate.Text.Trim().Replace("'", "_") + "'";
//separam prin VIRGULA
//VALORILE campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ", ";
//
//trimitem TRADUCATORII noii carti: inlocuim apostrofurile ' din textul scris cu _ :
stextulcomenziideadaugare_carte +=
                        "'" + tbcartenoua_traducator.Text.Trim().Replace("'", "_") + "'";
//aceasta^ este ULTIMA VALOARE a unui camp de completat
//asteptate de comanda SQL INSERT, asa ca inchidem lista lor cu paranteza rotunda
//
//astfel ca nu mai scriem virgula, ci:
//inchidem cu PARANTEZA ROTUNDA lista VALORILOR
//campurilor de completat asteptate de comanda SQL INSERT:
stextulcomenziideadaugare_carte += ")";
//
//
//acum solicitam EXECUTAREA acestei comenzi SQL:
//
//deschidem conexiunea la baza de date:
//
sqlcon.Open();
sqlcomm.CommandText = stextulcomenziideadaugare_carte;
sqlcomm.ExecuteNonQuery();
//
//
//am trimis o noua inregistrare in baza de date
//iar daca inregistrarea a fost acceptata –
//daca procedura aceasta s-a efectuat cu succes, atunci:
//inchidem conexiunea la baza de date:
//
sqlcon.Close();
//
//
```

```
                //inregistrarea cu informatiile despre o carte noua
                //a fost adaugata la tabelul t_carti
                //asa ca anuntam succesul:
                //
                MessageBox.Show(
                        "Cartea propusa a fost ADAUGATA la biblioteca",
                        "CARTE NOUA",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
                //nu s-a reusit adaugarea unei noi carti
                //asa ca anuntam esecul:
                MessageBox.Show(
                        "Cartea propusa NU A FOST ADAUGATA la biblioteca.\r\n\r\nEROARE:\r\n" +
                        ex.ToString(),
                        "CARTE NOUA",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        }
        finally
        {
                //indiferent daca inregistrarea a fost sau nu acceptata:
                //distrugem (si implicit inchidem) conexiunea la baza de date:
                //
                sqlcon.Dispose();
                //
                //pentru a indica faptul ca s-au incheiat operatiunile teoretic de durata
                //(re)stabilim Proprietatea Cursor a Form1 la: Default:
                //
                this.Cursor = Cursors.Default;
        }
}
```

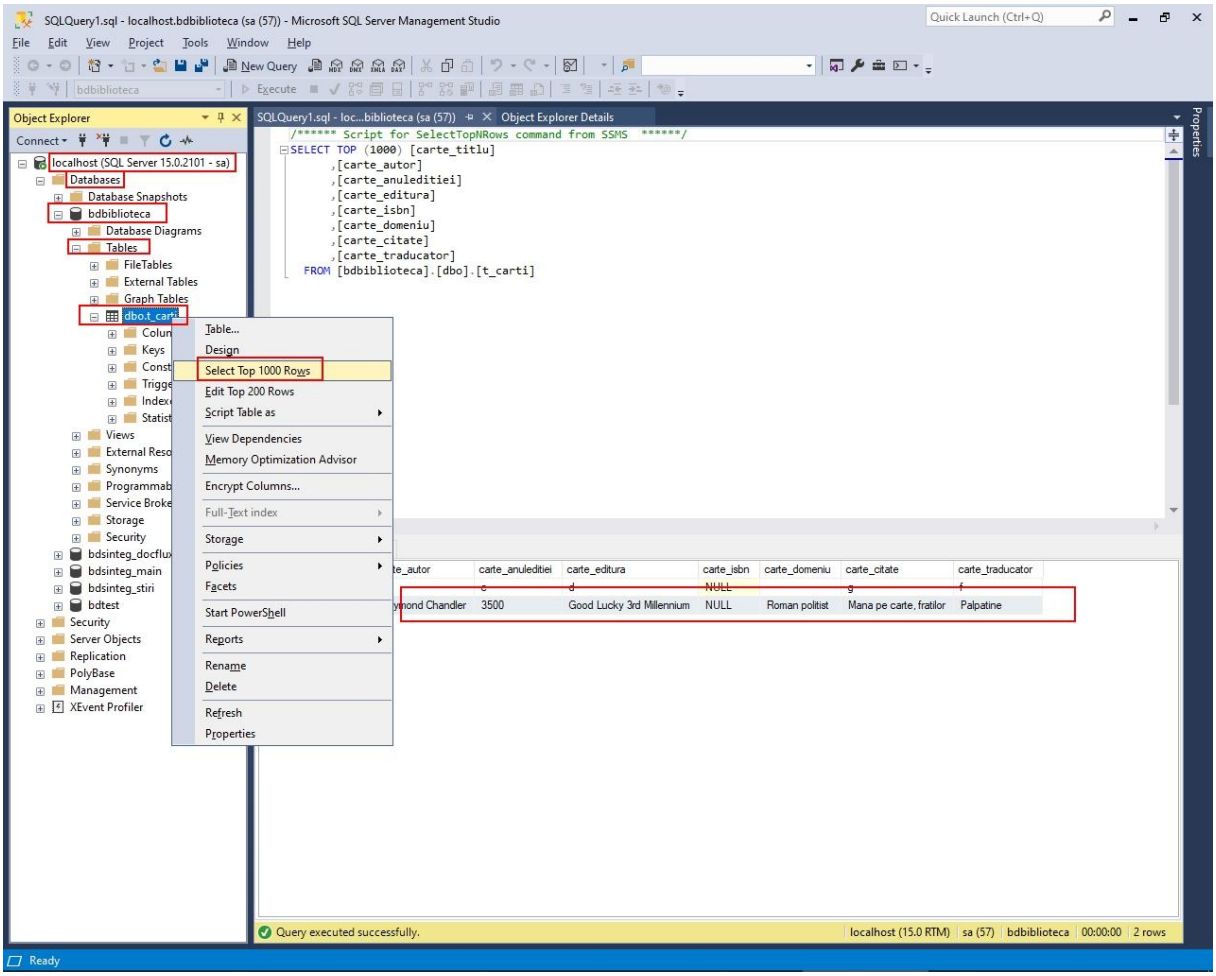
Deja ştim cum să plasăm codul de mai sus în programul nostru:

Nu se face Copy-Paste înainte de
a solicita generarea (automată) a metodelor asociate Evenimentelor implicite
ale Controalelor si Evenimentului *Form1_Load* al *Form1*.

Rezultatul adăugării la inventarul bibliotecii a informaţiilor despre o carte este ilustrat în imaginea de pe pagina următoare:


Biblioteca - Baza de date
Program
CARTILE AFLATE ACUM IN BIBLIOTECA:
SA VAD LISTA CARTILOR
CARTEA DE ADAUGAT LA BIBLIOTECA:
DOMENIUL/GENUL/TIPUL:
Roman politist
TITLUL - info necesara pentru a inregistra aceasta carte:
Somnul de veci
AUTORII - info necesara pentru a inregistra aceasta carte:
Raymond Chandler
ANUL EDITIEI - optional:
3500
EDITURA - optional:
Good Lucky 3rd Millennium
ISBN - optional:
1234567890
TRADUCATORI - optional:
Palpatine
CITATE - optional:
Mana pe carte, fratilor
ADAUG ACEASTA CARTE
CARTE NOUA
Cartea propusa a fost ADAUGATA la biblioteca
OK

Cartea nou adăugată la inventar apare după o interogare SELECT în SQL Server Management Studio:



Vrem şi să fie afişate informaţii despre cărţile pe care ***cu acest program*** le adăugăm la bibliotecă.

Pentru aceasta, desigur, trebuie să scriem cod care să fie invocat la apăsarea butonul de deasupra *TreeView*-ului dedicat afişării unor informaţii despre cărţile adăugate ***cu acest program***.

Astfel că,
ştiind că spaţiile de nume au o granularitate mare a posibilităţilor oferite unui programator,
vom căuta în spaţiul de nume ***System.Data.SqlClient*** entităţi (membri ai săi) care oferă posibilitatea de a parcurge o colecţie de înregistrări dintr-un tabel al unei baze de date.

Membrul sau entitatea din ***System.Data.SqlClient***
şi care poate fi folosit pentru a parcurge o astfel de colecţie este:

SqlDataReader

Un obiect sau o variabilă de acest tip poate fi obţinut ca rezultat a executării unei comenzi SQL definită de un obiect sau o variabilă de tipul ***SqlCommand***.

Anume, între membrii unui obiect ***SqlCommand*** există metoda:

ExecuteReader()

al cărei rezultat este un obiect de tipul ***SqlDataReader***.

SqlDataReader parcurge colecţia/lista înregistrărilor pe rând, de la prima la ultima înregistrare
aşa cum au fost obţinute după o comandă SELECT
din blocul de instrucţiuni specificate în textul ***SqlCommand*** asociate (Proprietatea **CommandText**).

Odată definit şi creat un ***SqlDataReader***,
valorile câmpurilor care alcătuie fiecare înregistare
pot fi obţinute prin apelarea:

- numelui acelui câmp
- indexului/poziţiei în lista câmpurilor, numărând de la 0 încolo.

Ştiind acestea care sunt suficiente pentru ceea ce vrem să facem, să scriem cod pentru extragerea unor informaţii despre cărţile pe care ***cu acest program*** le-am adăugat la bibliotecă.

Repetăm "***cu acest program***" fiindcă la bibliotecă pot fi adăugate cărţi şi în aceste moduri:

- manual, din SQL Server Management Studio
- cu alte programe care pot trimite informaţii despre cărţi în alte feluri (de exemplu, anul poate fi scris altfel decât cu patru_4 cifre; dar vom vrea să **SELECT**-ăm anii şi să-i ordonăm strict/numai ca numere)

Aşadar, mergem în modul de proiectare/Design al Form1, unde:

Din ToolBox, pe *Form1* plasăm o etichetă *Label* pe care:

- după ce redimensionăm *TreeView*-ul trăgând în sus de marginea sa inferioară
  - o plasăm sub *TreeView*
- o redenumim "labnumarulcartilor"
- îi stabilim Proprietatea *Text* la: "[numarul cartilor din biblioteca]"

Dublu-click pe butonul cu care solicităm lista cărţilor din bibliotecă.

Urmează pagini de cod care poate fi *Copy*-at şi apoi plasat-*Paste* în codul programului
înlocuind metoda generată automat, asociată Evenimentului *Click* al acestui buton:

Comentariile explică aproape fiecare instrucţiune, aşa că trebuie citite cu atenţie.

```
private void bafisarebiblioteca_Click(object sender, EventArgs e)
{
        //in aceasta metoda se fac operatiuni care pot dura un timp sesizabil,
        //asa ca stabilim valoarea Proprietatii Cursor a acestei Form-e
        //incat sa indice desfasurarea operatiunilor pana cand acestea se incheie:
        //
        this.Cursor = Cursors.WaitCursor;
        //
        //solicitam popularea TreeView cu
        //informatii despre cartile din biblioteca
        //pe care vor fi grupate dupa domeniul/genul/tipul lor
        //
        //pentru a pregati aceasta grupare ---vizuala--- a cartilor
        //trebuie ca la TreeView sa adaugam
        //noduri/elemente numite la fel cu titlurile cartilor din biblioteca
        //
        //adaugarea acestor noduri la TreeView poate fi facuta astfel:
        //
        //1. folosind elementele din ComboBox,
        //fiindca acolo sunt chiar denumirile domeniilor/genurilor/tipurilor de carte
        //
        //2. extragand acele denumiri din baza de date, cu comanda SELECT DISTINCT
        //
        //
        //aici vom folosi abordarea 1.
        //desi daca ne-am hotarat la asta, operatiunea trebuie facuta
        //imediat dupa popularea ComboBox,
        //de exemplu in metoda de mai sus a Evenimentului Load
        //
        //deci a face asta in acest punct al programului este nerecomandat
        //
        //oricum, asta facem, cu mentiunea ca --- putem muta acest cod in metoda Form1_Load() ---
        //
        //intai stergem tot ce se afla in TreeView
        //pentru a nu adauga noduri la cele existente:
        //
        tvbiblioteca.Nodes.Clear();
        //
        //apoi resetam textul etichetei "labnumarulcartilor":
        //
        labnumarulcartilor.Text = "[numarul cartilor din biblioteca]";
        //
        //in aceasta eticheta vom afisa numarul cartilor din biblioteca
        //dupa ce parcurgem inregistrarile corespunzatoare din tabel
        //
        //acum preluam textul fiecarui element din ComboBox
        //si il plasam ca nod in TreeView
        //care are acel text...
        //- atat ca text al nodului respectiv
        //- cat si ca nume sau Cheie a acelui nod, pentru a sti ca sub nodul astfel --- identificat ---
        //vor fi plasate informatii despre cartile din domeniul/genul/tipul numit astfel
        //
```

```
for (byte b = 0; b < cbcartenoua_tip.Items.Count; b++)
{
        tvbiblioteca.Nodes.Add(
                cbcartenoua_tip.Items[b].ToString(),    //acesta va fi numele sau Cheia acestui nod
                cbcartenoua_tip.Items[b].ToString());
}
//
//retinem numarul domeniilor/genurilor/tipurilor de carte:
//
int inumdomcarte = tvbiblioteca.Nodes.Count;
//
//
//acum accesam tabelul cu informatiile despre carti
//pentru a extrage informatii despre toate cartile
//si a plasa fiecare carte sub nodul identificat prin numele acelui domeniu/gen/tip:
//
//
//cream iarasi o conexiune SQL la baza de date:
//
SqlConnection sqlcon_select_cartiledinbiblioteca = new SqlConnection();
//
//aceasta procedura nu este recomandata
//fiindca pe nivelul superior oricaror metode
//poate fi creat un obiect SqlConnection
//--- care sa fie folosit oriunde este nevoie ---
//
//textul de initializare a conexiunii cuprinde:
//server=
//acesta este numele serverului SQL
//sau numele sau adresa IP a computerului pe care se afla serverul SQL
//in acest caz, cand rulam acest program pe computerul unde este instalat SQL Server,
//folosim "localhost" sau punctul .
//
//initial catalog=
//acesta este numele bazei de date catre care vor fi trimise comenzi SQL
//in acest caz, baza de date este bdbiblioteca
//
//user=
//acesta este numele utilizatorului care va trimite comenzile SQL
//folosim numele administratorului __serverului SQL__: sa
//
//password=
//aceasta este parola acestui utilizator
//folosim parola pe care am stabilit-o la instalarea serverului SQL, aici fiind: sa123
//
sqlcon_select_cartiledinbiblioteca.ConnectionString =
                        "server=localhost;initial catalog=bdbiblioteca;user=sa;password=sa123";
//
//
//construim un obiect SqlCommand care, in acest caz, va trimite o comanda SQL de tip SELECT:
//
SqlCommand sqlcomm_select_cartiledinbiblioteca = new SqlCommand();
//
```

```
//specificam conexiunea pe care vor fi trimise comenzile din SqlCommand:
//
sqlcomm_select_cartiledinbiblioteca.Connection = sqlcon_select_cartiledinbiblioteca;
//
//scriem instructiunile SQL care constituie Proprietatea CommandText a acestui SqlCommand:
//
string scs = "USE bdbiblioteca; SELECT carte_titlu, carte_autor, carte_domeniu FROM t_carti";
//
sqlcomm_select_cartiledinbiblioteca.CommandText = scs;
//
//deschidem conexiunea la baza de date:
sqlcon_select_cartiledinbiblioteca.Open();
//
//
//
//
//cream obiectul de tip SqlDataReader
//care va parcurge inregistrarile obtinute dupa executarea SqlCommand definite anterior:
//
SqlDataReader dr = sqlcomm_select_cartiledinbiblioteca.ExecuteReader();
//
//
//
//
try
{
        //verificam daca a fost creat un SqlDataReader
        //fiindca e posibil ca instructiunile SQL din SqlCommand.CommandText
        //sa nu fi fost executate cu succes:
        if (dr != null)
        {
                //incepem parcurgerea inregistrarilor:
                //atat timp cat mai poate fi citita o inregistrare din dr (acest SqlDataReader)...
                while (dr.Read())
                {
                        //... verificam daca exista campul numit cutare
                        //... si daca exista, atunci citim valoarea sa:
                        //
                        string scarte_titlu =
                                dr["carte_titlu"] != null ? dr["carte_titlu"].ToString() : "";
                        string scarte_autor =
                                dr["carte_autor"] != null ? dr["carte_autor"].ToString() : "";
                        string scarte_gen =
                                dr["carte_domeniu"] != null ? dr["carte_domeniu"].ToString() :
                        "";
                        //
                        //
                        //in TreeView cream un nod care sa aiba textul format astfel:
                        //titlul cartii (numele autorului):
                        //
                        TreeNode tn_carte =
                                        new TreeNode(scarte_titlu + " (" + scarte_autor + ")");
                        //
                        //in TreeView, unde anume sa fie plasata aceasta carte?
                        //
```

```
                        //daca in TreeView exista un nod cu numele/Cheia
                        //acelasi cu valoarea campului carte_domeniu
                        //atunci aceasta carte se plaseaza sub acel nod:
                        //
                        if (tvbiblioteca.Nodes[scarte_gen] != null)
                        {
                                tvbiblioteca.Nodes[scarte_gen].Nodes.Add(tn_carte);
                        }

                        else
                        {
                                //in TreeView nu exista un nod cu numele acelasi
                                //cu valoarea campului carte_domeniu
                                //asa ca adaugam la TreeView acest nod ca unul independent:
                                //
                                tvbiblioteca.Nodes.Add(tn_carte);
                        }
                }
        }
        //
        //acum pentru fiecare nod cu numele unui domeniu/tip/gen de carte
        //numaram elementele-descendent (adica, numarul cartilor din acel domeniu/tip/gen)
        //si afisam acest numar langa textul nod:
        //totodata vrem sa aflam numarul cartilor din biblioteca:
        //
        int inum_cartidinbiblioteca = 0;
        foreach (TreeNode tn in tvbiblioteca.Nodes)
        {
                if (tn.Nodes.Count > 0)
                {
                        tn.Text += " (" + tn.Nodes.Count.ToString() + " carti)";
                        //
                        inum_cartidinbiblioteca += tn.Nodes.Count;
                }
        }
        //
        //am numarat nodurile-descendent ale unor noduir-domeniu
        //dar e posibil ca in TreeView sa fie si noduri corespunzand unor carti
        //al caror domeniu nu a fost specificat
        //astfel ca obtinem numarul real al cartilor:
        //combinand numarul nodurilor cu mai mult de 0 descendenti
        //cu
        //numarul nodurilor fara descendenti DAR diferit
        //de numarul nodurilor care denota domenii
        //
        inum_cartidinbiblioteca += (tvbiblioteca.Nodes.Count - cbcartenoua_tip.Items.Count);
        //
        labnumarulcartilor.Text = "In biblioteca se afla: " +
                                                        inum_cartidinbiblioteca.ToString() +
                                                        " carti din " +
                                                        inumdomcarte.ToString() +
                                                        " domenii";
}
```

```
        catch (Exception ex)
        {
                MessageBox.Show(
"NU S-A REALIZAT extragerea de informatii despre cartile din biblioteca.\r\n\r\nEROARE:\r\n" + ex.ToString(),
                        "CARTILE DIN BIBLIOTECA",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        }
        finally
        {
                //restabilim Proprietatea Cursor a acestei Form-e la Default, pointerul-sageata sau similar:
                //
                this.Cursor = Cursors.Default;
                //
                //inchidem SqlDataReader—ul cu care am parcurs lista cartilor din tabel/din baza de date:
                if (dr != null)
                {
                        dr.Close();
                }
                //
                //distrugem obiectul SqlCommand in care am plasat instructiunile SQL pentru
                //populatea SqlDataReader-ului cu informatii despre toate inregistrarile
                //cu informatii despre toate cartile din biblioteca-baza de date:
                //
                if (sqlcomm_select_cartiledinbiblioteca != null)
                {
                        sqlcomm_select_cartiledinbiblioteca.Dispose();
                }
                //
                //distrugem obiectul SqlConnection
                //asociat acestui SqlCommand care a trimis instructiuni catre serverul SQL
                //unde se afla baza de date cu biblioteca:
                //
                if (sqlcon_select_cartiledinbiblioteca != null)
                {
                        sqlcon_select_cartiledinbiblioteca.Close();
                        sqlcon_select_cartiledinbiblioteca.Dispose();
                }
        }
}
```

Până acum, în baza de date am înregistrat informaţii numai despre două_2 cărţi:

- o carte...
  - înregistrată manual în SQL Management Studio
  - al cărei titlu este "a
  - al cărei autor este "b"
  - al cărei domeniu/tip/gen este nul
- o carte
  - înregistrată cu acest program
  - al cărei domeniu/tip/gen este "Roman politist"

În imaginea următoare

- unde în meniul "*Program*" se află o intrare cu sensul de "*Inchidere/Exit/Close*" care apelează instrucţiunea ***Application.Exit();*** eventual după confirmarea închiderii programului)

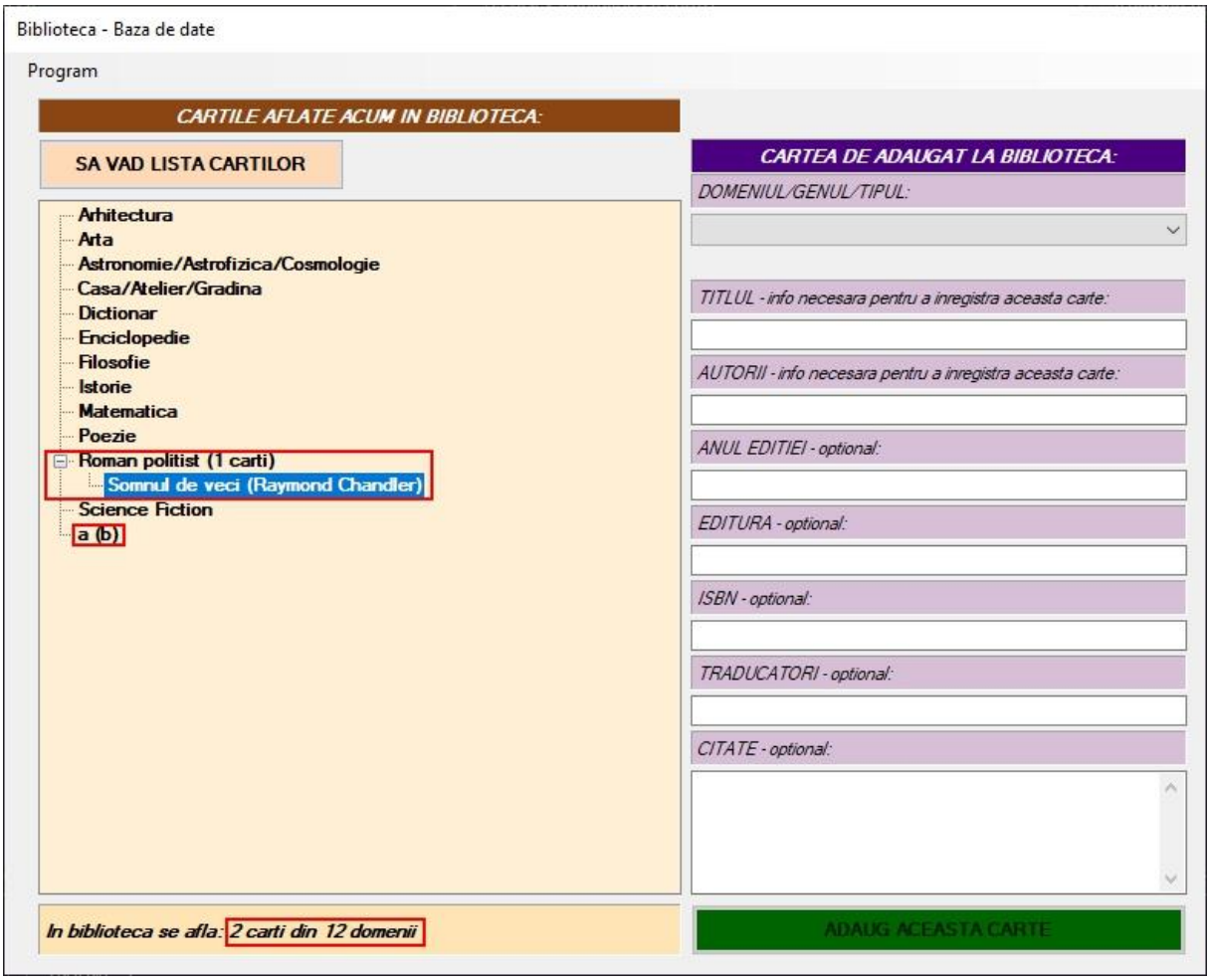
se vede cum sunt afişate aceste două cărţi:

- cea înregistrată fără a avea asociat un domeniu este afişată ca nod independent
- cea înregistrată cu acest program se află sub nodul domeniului/tipului/genului "Roman politist"

Totodată, numărul cărţilor este corect:

- 2 cărţi
- din 12 domenii/tipuri/genuri cunoscute
- dar inclusiv cartea fără domeniu/tip/gen cunoscut, adăugată manual din SSMS

Vom ajusta aceste calcule şi unele mesaje.



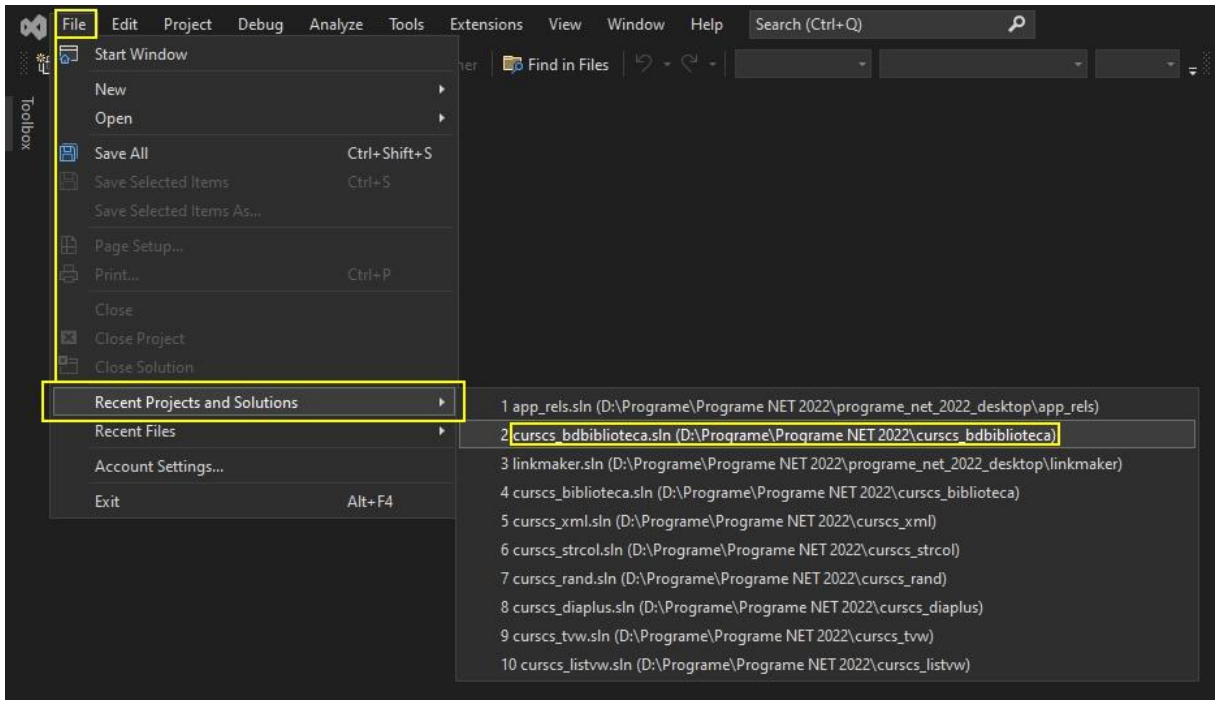
Ne-am mai gândit şi vrem să excludem posibilitatea de a fi înregistrată o carte-duplicat.

Aici, prin Duplicat înţelegem o carte care are unele caracteristici identice cu cele ale altei cărţi.

Diferenţierea cărţilor se poate face prin acest set de caracteristici:
- domeniul/tipul/genul
- numele autorului sau numele autorilor
- titlul
- anul ediţiei
- numele editurii
- codul ISBN
- numărul de pagini

Pentru a implementa o astfel de verificare în program,
bineînţeles că trebuie să modificăm programul,
întâi deschizând Proiectul corespunzător, cu Visual Studio:

Pornim Visual Studio,
apoi din meniul *File* alegem intrarea "*Recent Projects and Solutions*"
în a cărei listă se află listate cele mai recente (de obicei, primele 4 începând cu cel mai recent) Proiecte şi Soluţii, ca în imaginea următoare:



Dacă Proiectul programului nostru se află în acea listă, atunci click pe denumirea sa şi aşteptăm să se încarce în Visual Studio.

Dacă nu se află în listă, atunci click pe intrarea de meniu *File -> Open -> Project/Solution*, navigăm prin directoare/foldere până la cel unde am creat programul şi
selectăm fişierul cu extensia "**.sln**".

Bănuim că o carte adăugată la inventarul din baza de date şi astfel în *TreeView*-ul de pe *Form1* poate fi identificată cu numai o parte din setul menţionat de caracteristici – anume, numai prin acestea:

- anul ediţiei
- numele editurii
- codul ISBN

În cazul cărţilor, pe plan internaţional identificarea unei ediţii se face prin **codul ISBN**.

Acest cod se află, de obicei, fie pe ultima pagină a cărţii, fie pe coperta a patra (spatele cărţii).

Astfel că ne folosim de aceasta şi modificăm în consecinţă programul nostru:

Aşadar, intenţionăm să folosim **codul ISBN** ca identificator unic al unei ediţii.

Dar ne amintim că în program codul ISBN este opţional.

Mai mult, e posibil să nu cunoaştem codul ISBN.

Totuşi, acesta e mai uşor de găsit decât anul ediţiei.

Aşa că vom folosi
o caracteristică obligatoriu de specificat la adăugarea unei cărţi – anume,
domeniul/tipul/genul cărţii.

Deşi această abordare nu este suficientă pentru evitarea duplicatelor,
vom găsi totuşi o soluţie ca măcar utilizatorul să fie avertizat asupra încercării de adăuga o astfel de carte.

Vom marca fiecare nod/element din *TreeView* şi care denotă o carte
cu domeniul/tipul/genul cărţii.

Pentru aceasta vom folosi proprietatea *Tag* a unui *TreeNode*.

Modificăm metoda **bafisarebiblioteca_Click**
în porţiunea unde la *TreeView* se adaugă un *TreeNode* cu titlul şi autorii cărţii celei noi, astfel:

Înlocuim porţiunea de cod:

```
//in TreeView cream un nod care sa aiba textul format astfel:
//titlul cartii (numele autorului):
//
TreeNode tn_carte =
                    new TreeNode(scarte_titlu + " (" + scarte_autor + ")");
//
```

cu aceasta:

```
//in TreeView cream un nod care sa aiba textul format astfel:
//titlul cartii (numele autorului):
//
TreeNode tn_carte =
                    new TreeNode(scarte_titlu + " (" + scarte_autor + ")");
//
//acestui nod ii atribuim, ca valoare a proprietatii Tag,
//domeniul/tipul/genul acestei carti:
//
tn_carte.Tag = scarte_gen.ToUpper();
//
//^ am folosit .ToUpper() care transforma textul respectiv la majuscule
//fiindca vrem sa uniformizam acel text (domeniul/tipul/genul cartii)
//in vederea verificarii existentei unor duplicate ale acestei carti (desi verificarea
//numai dupa domeniu/tip/gen nu este de ajuns, dar acum ne multumim cu asta)
//
```

Astfel, de acum înainte cărţile vor fi adăugate la *TreeView* folosind o caracteristică a lor care va fi stocată în valoarea proprietăţii *Tag* a nodului/elementului respectiv.

Totodată, atunci când verificăm existenţa unui duplicat al cărţii pe care intenţionăm să o adăugăm:

- vom citi valoarea proprietăţii Tag a fiecărui nod din *TreeView*
- şi o vom compara cu domeniul/tipul/genul cărţii selectat în *ComboBox*

Pentru aceasta:
în metoda **Biblioteca_Adaugare_Carte**
înlocuim porţiunea de cod:

```
private void Biblioteca_Adaugare_Carte()
{
        //
        //pentru a indica faptul ca se efectueaza operatiuni teoretic de durata
        //stabilim Proprietatea Cursor a Form1 la: WaitCursor:
        //
        this.Cursor = Cursors.WaitCursor;
```

cu aceasta:

```
private void Biblioteca_Adaugare_Carte()
{
        //
        //
        //inainte de a adauga o carte
        //verificam daca in TreeView exista deja un nod de carte
        //--- deci, ca nod-descendent - sub un nod-domeniu ---
        //cu aceleasi caracteristici ca ale cartii propuse:
        //- titlu
        //- autor(i)
        //- domeniu/tip/gen
        //
        //parcurgem nodurile-parinte, la nivelul 0/radacina TreeView:
        //TreeView poate fi gol, caz in care instructiunile din acest bloc foreach nu se executa:
        foreach (TreeNode tn in tvbiblioteca.Nodes)
        {
                //daca exista astfel de noduri-parinte:
                if (tn.Nodes.Count > 0)
                {
                        //atunci verificam daca fiecare astfel de nod are noduri-descendent:
                        foreach (TreeNode tn_carte in tn.Nodes)
                        {
                                //verificam daca textul acestui nod-descendent este identic cu cel
                                //compus prin combinarea in acelasi fel a titlului si autorului/autorilor
                                //cartii propuse:
                                if
                                (
                                tn_carte.Text.ToUpper() ==
                                        tbcartenoua_titlu.Text.Trim().ToUpper() +
                                        " (" +
                                        tbcartenoua_autor.Text.Trim().ToUpper() +
                                        ")"
                                )
                                //daca am gasit un astfel de nod, mai departe
                                //verificam daca
                                //valoarea proprietatii sale Tag (ca text, ToString()) este aceeasi cu
                                //domeniul/tipul/genul cartii propuse - cel selectat in ComboBox
                                //de asemenea transformat la text, ToString():
                                //pentru compararea celor doua texte
                                //le transformam la majuscule (ToUpper()):
```

```
                                //
                                //si anuntam utilizatorul despre existenta unei astfel de carti:
                                {
                                        if (tn_carte.Tag.ToString().ToUpper() ==
                                                cbcartenoua_tip.SelectedItem.ToString().ToUpper())
                                        {
                                                MessageBox.Show(
                                        "Aceasta carte se afla deja in inventar:\r\n\r\n" +
                                        tn_carte.Text + "\r\n\r\n\r\n" +
                                "Cartile pot fi diferentiate prin aceste caracteristici simultan:\r\n" +
                                        "- titlu\r\n" +
                                        "- autor(i)\r\n" +
                                        "- domeniu/tip/gen",
                                        "Duplicat",
                                        MessageBoxButtons.OK,
                                        MessageBoxIcon.Warning);
                                        //iesim din aceasta metoda:
                                        //
                                        return;
                                        //dupa ce utilizatorului i-am dat indicatii incat sa modifice/schimbe
                                        //cel putin una din cele trei caracteristici
                                        }
                                }
                        }
                }
        }
        //
        //pentru a indica faptul ca se efectueaza operatiuni teoretic de durata
        //stabilim Proprietatea Cursor a Form1 la: WaitCursor:
        //
        this.Cursor = Cursors.WaitCursor;
```

Încă o dată - verificarea trebuie făcută după cât de multe caracteristici este posibil, ştiind aceasta:

cele stocate la adăugarea/înregistrarea unei cărţi

vor fi plasate în TreeView la solicitarea inventarului curent

Grafică– analiza simplă a unor puncte dintr-o imagine

Vom face un program unde se va evalua culoarea unor puncte din aceeaşi locaţie pe mai multe imagini.

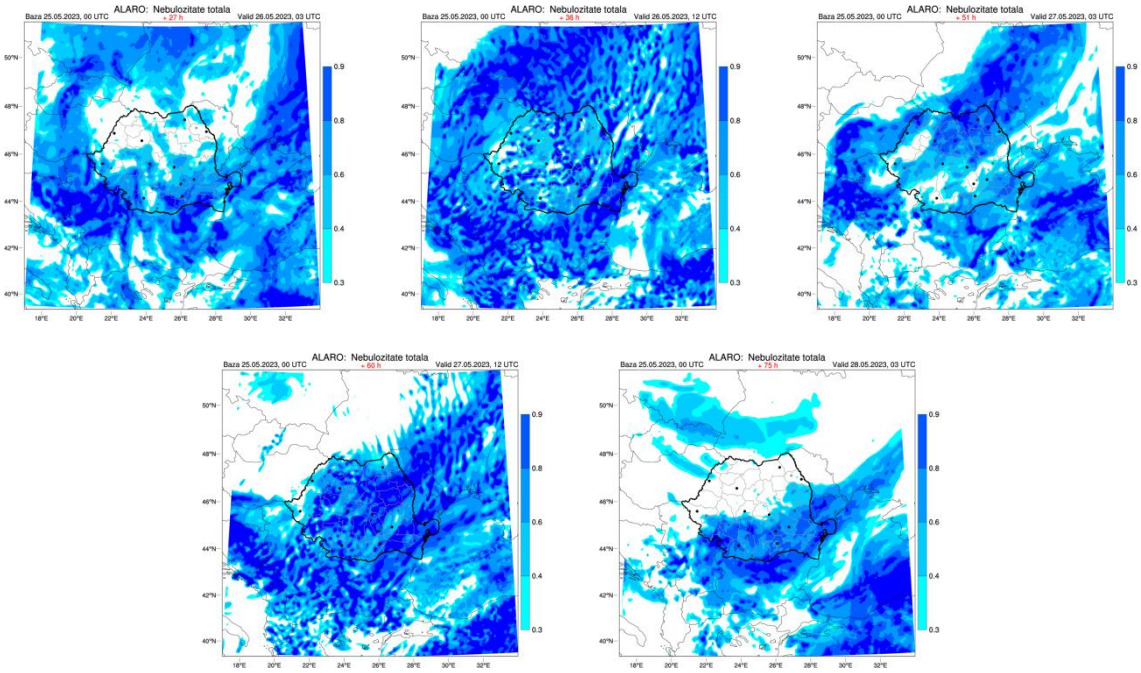
Procedura aceasta e utilă inclusiv în cazul când vrem să aflăm evoluţia în timp a unui parametru din acel punct – parametru care pe imagine este reprezentat cu anumite culori în funcţie de valoarea sa (de exemplu, albastru închis pentru nebulozitate de 7 optimi şi alb pentru cer senin).

În acest program vom evalua nebulozitatea – atât ca mărime (numeric), cât şi ca evoluţie în timp – dintr-un punct aflat la aceleaşi coordonate carteziene (Ox, Oy) pe mai multe imagini.

Pentru aceasta vom folosi:

- câteva imagini...
  - având aceeaşi dimensiune – atât pe orizontală, cât şi pe verticală...
  - şi pe care este reprezentat acelaşi contur / aceeaşi suprafaţă
- entităţi din (membri ai) spaţiul(ui) de nume **System.Drawing**.

Vom folosi aceste 5 imagini pe care este reprezentată nebulozitatea în optimi, la diverse momente deasupra unei zone geografice:



În/pe fiecare imagine vom evalua "valoarea" unui punct situat în acelaşi loc din Moldova şi reprezentat deja de un cerc plin, negru corespunzând poziţiei Iaşiului.

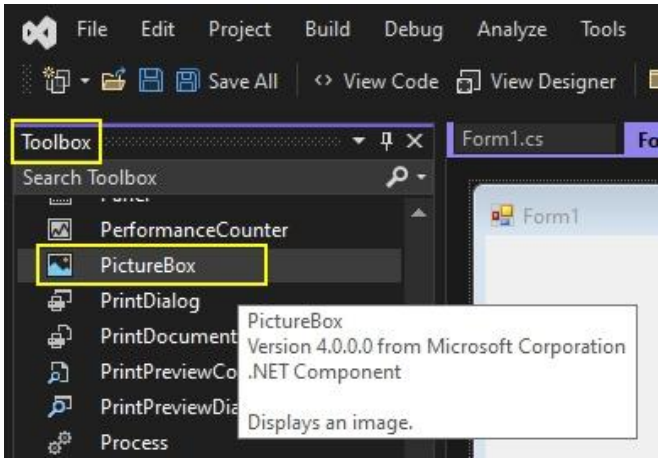
Să facem planul programului nostru:

Vom afla valoarea numerică a culorii dintr-un punct aflat la aceleaşi coordonate pe cele 5 imagini
Vom stoca valorile acestea într-un tablou (Array) pe care îl vom folosi la aflarea evoluţiei în timp;
Vom descrie evoluţia aceasta folosind texte (String-uri) dintr-un tablou.

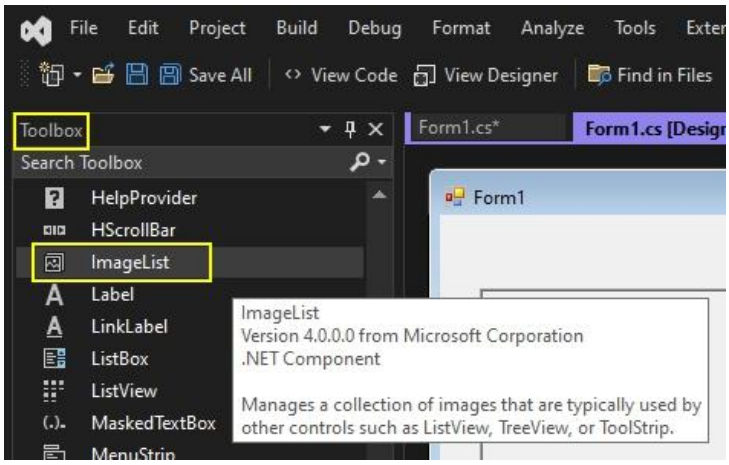
Să începem a alcătui programul.

Pornim Visual Studio, *File -> New... Project...*

Din *Toolbox*, pe *Form1* tragem un Control de tipul ***PictureBox***
pe care îl/o şi redenumim (Proprietatea *Name*) "pbharta":



Imaginile le stocăm într-un Control de tipul *ImageList*: tragem un astfel de Control pe *Form1*:

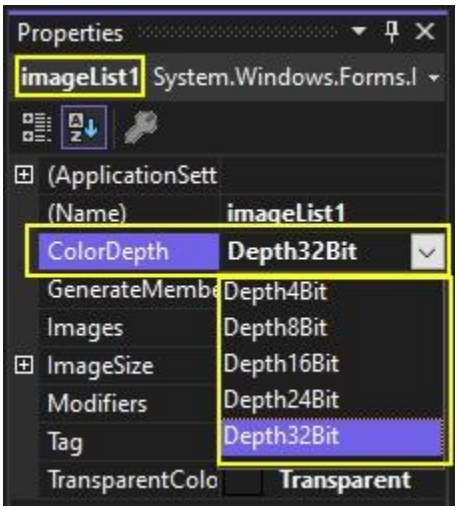


Vedem cum "imageList1" care nu are un contur anume şi nu este vizibilă pe *Form*-ă – la fel cu Controalele *OpenFileDialog*, *SaveFileDialog* şi *Timer* este plasată la baza interfeţei Visual Studio.
Să plasăm în această listă cele 5 imagini cu care vom lucra.

Click pe "imageList1" şi apoi tasta **F4** pentru a stabili câteva Proprietăţi ale acesteia, după cum este expus în continuare:

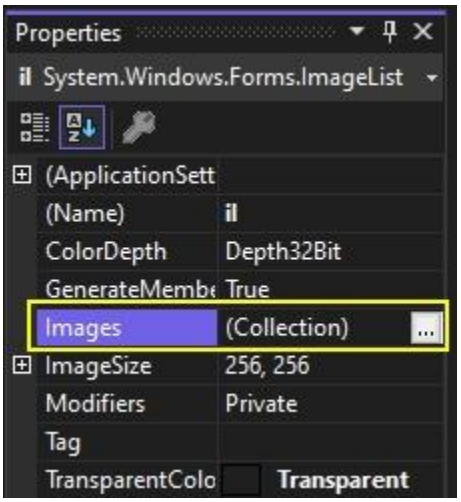
Întâi, adâncimea (ca nuanţe) a culorilor care vor fi susţinute de "imageList1", indiferent de adâncimea imaginilor care se află în acest Control.

Apoi, redenumim (Proprietatea *Name*) "imageList1" la "il":

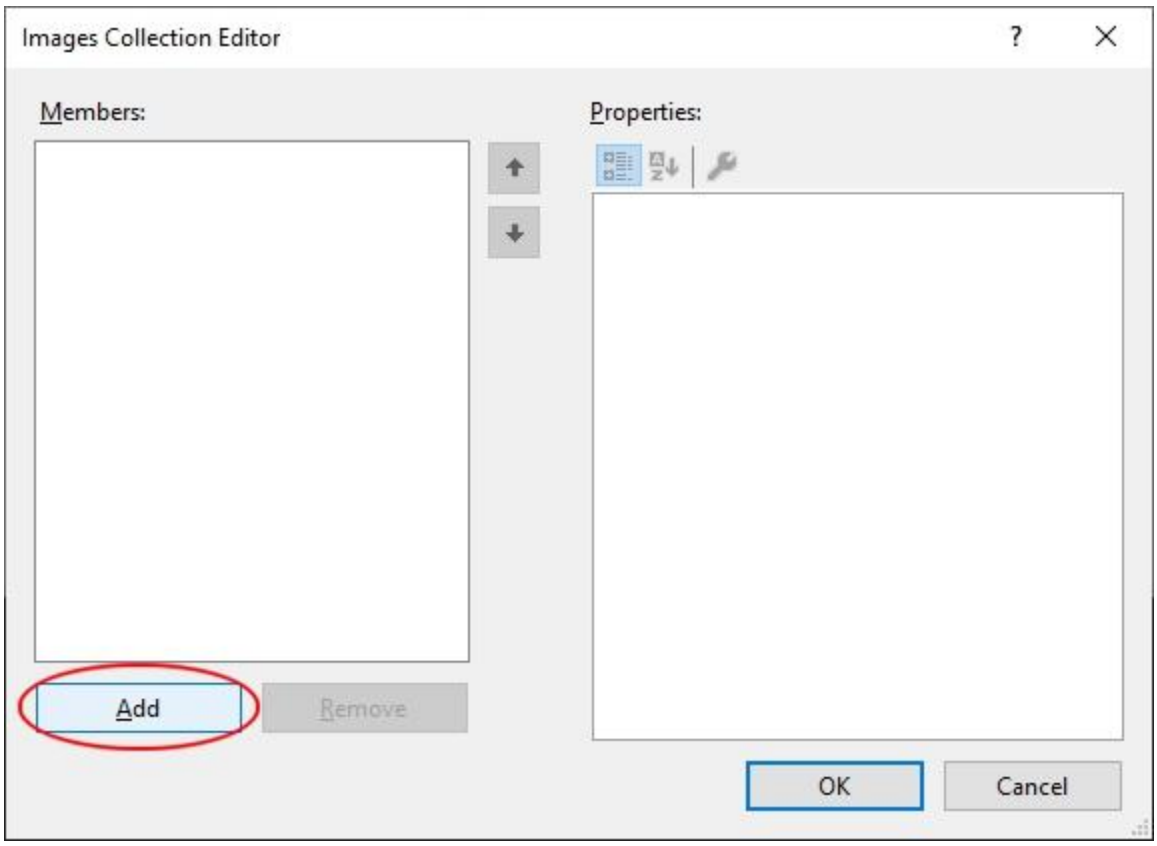


Acum stabilim valoarea Proprietăţii *ImageSize* la "256, 256":
Aceasta este dimensiunea maximă, în pixeli pe orizontală şi pe verticală,
a imaginilor *acceptabile ca reprezentare într-un Control de tipul ImageList*.
Imaginile vor fi accesibile la dimensiunea originală.
Imaginile pe care le folosim aici au dimensiunea orizontală 1152 pixeli şi pe cea verticală 958 pixeli.

Acum populăm "il" cu imaginile pe care le vom folosi:
Click pe cele trei puncte din dreptul Proprietăţii "*Images*" şi marcate "Collection":



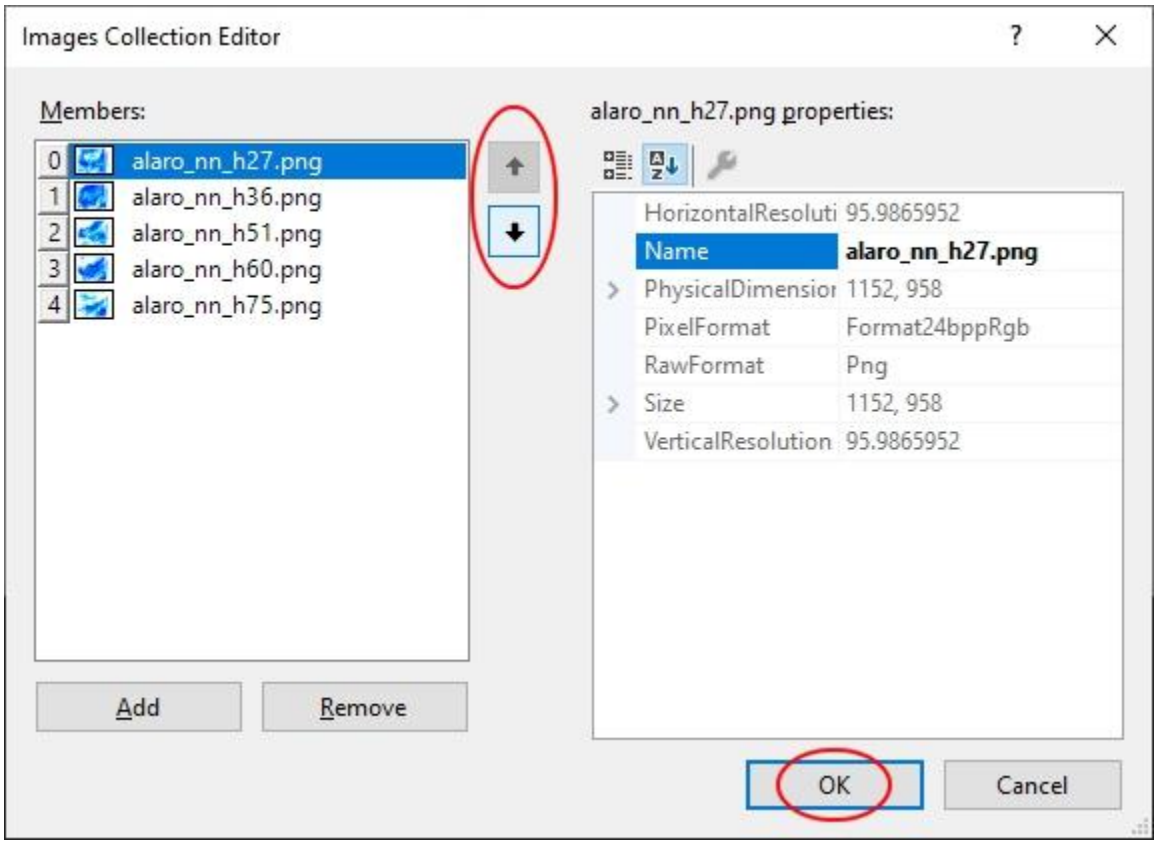
În dialogul care apare apăsăm butonul marcat "Add" şi selectăm imaginile:



Este recomandat să adăugăm imaginile într-o ordine pe care să o reţinem, pentru a şti cum să ne refeirm la fiecare. În programul nostru, adăugăm imaginile cu nebulozitate urmând linia temporală, începând cu imaginea corespunzătoare orei/anticipaţiei 27, continuând cu cea a orei 36 şi încheind cu cea a orei 75:

Cele două butoane marcate cu săgeată mută câte o poziţie imaginea selectată:

După ce adăugăm la "il" toate imaginile, apăsăm butonul OK:

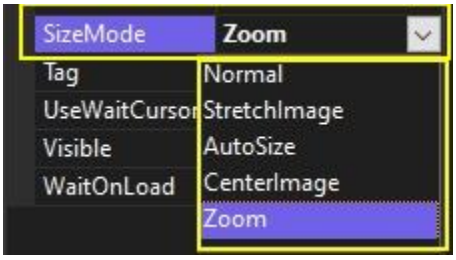


Aşadar, vrem ca imaginile să fie afişate în Controlul *PictureBox* "pbharta".

Însă este foarte probabil ca dimensiunea acestui Control să nu fie adecvată pentru imagini – acestea pot fi
- ori mai mari decât acest Control, încât se va vedea numai un fragment din ele,
- ori atât de mici încât Controlul va rămâne mai mult gol.

Un Control de tipul *PictureBox* are Proprietatea "*SizeMode*" ale cărei valori stabilesc modul în care va fi afişată o imagine în acel Control.

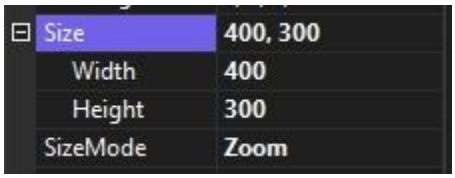
Stabilim valoarea acestei Proprietăţi la "*Zoom*":



Această valoare a *SizeMode* face încât imaginea din *PictureBox* să fie afişată integral într-un astfel de Control, respectându-se proporţiile. De asta şi avem nevoie, fiindcă:

- imaginile originale au aceeaşi dimensiune
- vrem să ne referim la acelaşi punct, din acelaşi loc, de pe fiecare imagine

Mai departe, presupunând că avem cunoştinţă despre aspectul imaginilor (de exemplu, mai late decât mai înalte – deşi aceasta nu contează fiindcă am stabilit *SizeMode* la "Zoom") stabilim Proprietatea *Size* (dimensiunea pe orizontală şi pe verticală) a acestei *PictureBox* la 400 pixeli pe orizontală (*Width, lăţime*) şi la 300 pixeli pe verticală (*Height, înălţime*):



Din Toolbox, pe *Form1* tragem un *Button*:

- pe care îl vom folosi pentru afişarea imaginii următoare din *ImageList* ("il")
- pe care îl redenumim la "bimgnext"
- al cărui *Text* îl stabilim la: "-> Imaginea urmatoare ->"

Încă un *Button*:

- pe care îl vom folosi pentru generarea unui text descriptiv al evoluţiei "valorilor punctului"
- pe care îl redenumim la "btextdinvalori"
- al cărui *Text* îl stabilim la "Un text descriind evolutia"

O casetă *TextBox*:

- unde vor fi afişate "valorile punctului" din fiecare imagine
- pe care o redenumim la "tbvaloripimg"

Încă o *TextBox*:

- unde va fi afişat textul descriptiv
- pe care o redenumim "tbtextpimg"

Acum, *Form1* ar putea arăta astfel:
(se vede cum am stabilit şi *Text*-ul/titlul *Form1* la "Evaluarea unor puncte de pe mai multe imagini"):



Vrem ca la afişarea *Form1*/la pornirea programului, în "pbharta" să fie afişată prima imagine din "il".

Pentru aceasta:
Dublu-click pe *Form1*, pentru a solicita de la Visual Studio generarea metodei asociate Evenimentului *Load* al acestui Dialog.

Odată astfel ajunşi în modul de scriere a codului:
Acea metodă o înlocuim cu aceasta:

```
private void Form1_Load(object sender, EventArgs e)
{
        pbharta.Image = il.Images[0];
}
```

Acum dăm un rost butonului cu care solicităm afişarea următoarei imagini:

În timpul rulării programului trebuie să ştim:
- **de câte ori am apăsat pe *PictureBox*, *presupunând că fiecare click defineşte un punct***
- care anume imagine din "il" este afişată în "pbharta" – anume, cea de la indexul curent în "il". Prima imagine din "il", cea de altfel şi afişată în *Form1_Load*, este imaginea de la indexul 0. Aşadar, definim o variabilă în care să fie reţinut indexul imaginii curente din "pbharta":
  Pe acelaşi nivel cu metoda *Form1_Load* scriem:

```
int i_img_curenta = 0;
```

De asemenea, vrem ca într-o variabilă să reţinem numărul imaginilor din ImageList:
Aşa că pe acelaşi nivel cu instrucţiunea precedentă declarăm o astfel de variabilă
pe care o iniţializăm cu valoarea 0; valoarea sa reală o vom stabili în metoda *Form1_Load* de mai sus

```
int inrimagini = 0;
```

Ne întoarcem în metoda *Form_Load* unde scriem o instrucţiune pentru stabilirea valorii acestei variabile:

```
private void Form1_Load(object sender, EventArgs e)
{
        inrimagini = il.Images.Count;
        pbharta.Image = il.Images[0];
}
```

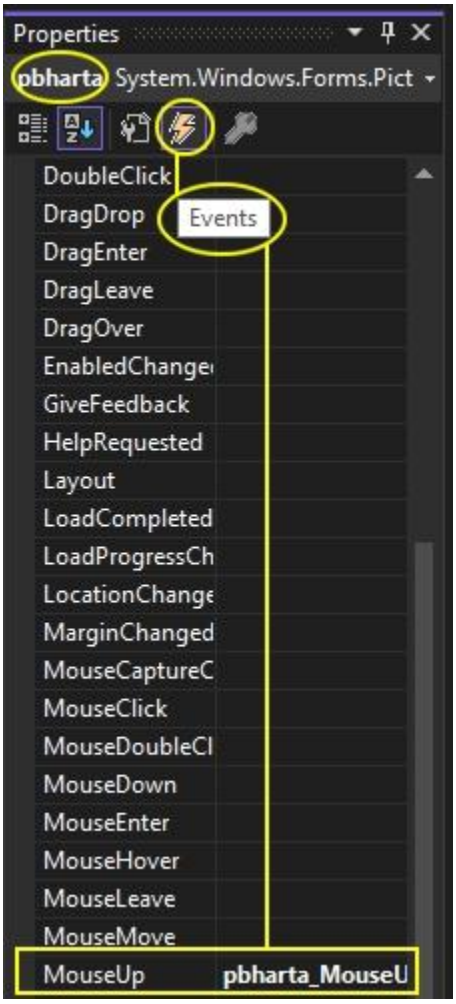
Acum putem scrie codul pentru buton:
Înapoi în modul de proiectare/Design al *Form1*: tastele Shift şi **F7**.
Dublu-click pe butonul "bimgnext" cu care solicităm următoarea imagine:
Ceea ce ne trimite înapoi în modul de scriere a codului, unde metoda tocmai generată pentru Evenimentul *Click* o înlocuim cu:

```
private void bimgnext_Click(object sender, EventArgs e)
{
	try
	{
		//inaintam o pozitie in ImageList pentru a afisa imaginea de la acel index:
		i_img_curenta++;
		//
		pbharta.Image = il.Images[i_img_curenta];
		//
		//--- acum --- fiind afisata ultima imagine din "il"
		//deci a 5-a imagine aflata la indexul 4
		//reinitializam numaratoarea:
		//
		if (i_img_curenta == 4)
		{
			//restabilim indexul la -1 fiindca prima instructiune din aceasta metoda
			//deja incrementeaza cu 1 acest index:
			//
			i_img_curenta = -1;
		}
	}
	catch (Exception ex)
	{
	}
}
```

Acum să scriem codul pentru aflarea "valorii punctului" unde am apăsat pe *PictureBox*:
Trebuie să aflăm coordonatele acelui punct.
Pentru aceasta:

Înapoi în modul de proiectare/Design al *Form1*:
Click pe "pbharta"
Apoi tasta **F4**, pentru a accesa Proprietăţile şi Evenimentele acestei *PictureBox*:

Click pe pictograma-fulger care denotă Evenimentele oricărui Control selectat (inclusiv ale unei *Form*-e):
Dublu-click pe Evenimentul *MouseUp*:



Ceea ce ne trimite înapoi în modul de scriere a codului,
unde metoda tocmai generată automat, asociată Evenimentului *MouseUp* al "pbharta" o înlocuim cu:

Urmează instrucţiuni cu referiri la noţiuni/membri nou întâlniţi.
Semnificaţia lor este:

- după ce butonul stâng al mouse-ului a fost eliberat (Evenimentul *MouseUp*)...
- în memorie este creată/pregătită o suprafaţă grafică de tipul *Bitmap* pentru o imagine...
- pe care suprafaţă este desenată imaginea afişată în *PictureBox*-ul "pbharta"...
- şi de pe care se obţine valoarea culorii din punctul unde butonul mouse-ului a fost eliberat...
- care valoare este adăugată la textul din TextBox-ul unde vor fi scrise valorile culorilor...
- iar după toate acestea distrugem obiectul creat anterior, de tip *Bitmap*
  folosit pentru a obţine valoarea culorii

```
private void pbharta_MouseUp(object sender, MouseEventArgs e)
{
        //in memorie este creata/pregatita o suprafata grafica de tipul Bitmap pentru o imagine...
        //
        Bitmap b = new Bitmap(pbharta.ClientSize.Width, pbharta.Height);
        //
        //pe care suprafata este desenata imaginea afisata în PictureBox-ul "pbharta"...
        //
        pbharta.DrawToBitmap(b, pbharta.ClientRectangle);
        //
        //si de pe care se obtine valoarea culorii din punctul
        //unde butonul mouse-ului a fost eliberat...
        //
        Color colour = b.GetPixel(e.X, e.Y);
        //
        //care valoare este adaugata la textul din TextBox-ul unde vor fi scrise valorile culorilor...
        //
        tbvaloripimg.Text += colour.ToArgb().ToString() + "\r\n";
        //
        //iar după toate acestea distrugem obiectul creat anterior, de tip Bitmap
        //folosit pentru a obţine valoarea culorii:
        //
        b.Dispose();
        //
        //plasam in tabloul valorilor de culoare valoare curenta:
        //verificand intai daca s-a revenit la -1, caz in care valoare e plasata la indexul 4:
        //
        iculori[i_img_curenta > -1 ? i_img_curenta : 4] = colour.ToArgb();
        //
}
```

Pentru descrierea evoluţiei valorilor trebuie să ştim domeniul pe care se întind acestea –
de exemplu, de la 0 la 100.
Astfel, vom putea asocia textul "cer senin" valorii 0, textul "cer acoperit" valorii 100 şamd.

Pentru aceasta trebuie să aflăm între ce valori (extreme) se află cele de pe imaginile noastre.

Presupunând că toate imaginile din programul nostru au fost desenate:
- ***folosind aceeaşi paletă de culori***
- ***ordonate în acelaşi fel***

putem afla domeniul de valori folosind chiar imaginea afişată la pornirea programului:

Astfel:

Pornim programul: tasta **F5**.

În "pbharta" deja este afişată imaginea de la indexul 0 în *ImageList* "il".

Click pe o zonă cu "cer senin":
Valoarea culorii din acel punct va fi scrisă în *TextBox*-ul " tbvaloripimg"
Click pe o zonă cu "cer acoperit":
Valoarea culorii din acel punct va fi scrisă în *TextBox*-ul " tbvaloripimg"

Vom obţine valori similare celor din această imagine, "cerul senin" fiind reprezentat de valoarea -1:



Astfel am aflat domeniul de valori de pe imaginile noastre.

Notăm în cod aceste valori (pe linii de comentarii, marcate cu //
fiindcă ne-ar putea folosi chiar şi în alte programe
şi oricum,
în raport cu ele aici vom descrie evoluţia valorilor intermediare

Pentru aflarea/analizarea/descrierea evoluţiei valorilor
trebuie să reţinem valorile culorii din acelaşi punct de pe fiecăre imagine.

Pentru aceasta:

Sub declaraţiile celor două variabile numerice scrise mai devreme şi pe acelaşi nivel cu ele:
Declarăm un tablou de valori numerice şi având 5 elemente:

int[ ] iculori = new int[5];

Tot aici declarăm şi un tablou de String-uri adecvate situaţiei din imagini şi din care să fie extras textul pentru descriere.

De exemplu:

string[ ] stexte = { "acoperit #1", "acoperit #2", "acoperit #3", "acoperit #4", "acoperit #5" };

Să generăm o scurtă descriere a situaţiei ***medii*** din imagini (deci, încă nu *evoluţia*):

Înapoi în modul de proiectare/Design al *Form1*, unde
Dublu-click pe butonul "btextdinvalori"
A cărui metodă asociată Evenimentului său *Click* o înlocuim cu:

(urmează cod pe trei pagini; comentariile explică unele instucţiuni)

```
private void btextdinvalori_Click(object sender, EventArgs e)
{
	//aflam media aritmetica a --- tuturor valorilor --- de culoare obtinute prin click pe imagini:
	int imedie_total = 0;
	int isuma_total = 0;
	//
	for (int i = 0; i < iculori.Length; i++)
	{
		isuma_total += iculori[i];
	}
	imedie_total = (int)isuma_total / 5;
	//
	MessageBox.Show("Media TUTUROR valorilor de culoare:\r\n\r\n" + imedie_total.ToString());
	//
	//
	//pentru a putea face evaluarea finala avem nevoie de media --- extremelor ---
	//ca media aritmetica a valorilor obtinute prin click pe (aproximativ) acelasi punct
	//vrem sa aflam cele doua extreme sortand toate valorile:
	//astfel ca pentru aceasta declaram un tablou cu aceeasi dimensiune (5)
	//ca a celui unde sunt acele valori si pe care il putem sorta FARA a il afecta pe cel original:
	//
	int[ ] iculori_ptmedie = new int[5];
	//
	//copiem elementele din tabloul original in acesta pe care il vom Sort-a:
	//deci, cream un duplicat al tabloului original
	//numai pentru a sorta elementele in scopul de a face o medie aritmetica
	//
	Array.Copy(iculori, iculori_ptmedie, 5);
	//
	//sortam tabloul acesta-duplicat:
	//
	Array.Sort(iculori_ptmedie);
	//
	//aflam media aritmetica a celor doua valori extreme –
	//primul si ultimul element din tabloul Sort-at:
	//
	int imedie_valoriminmax =
			(iculori_ptmedie[iculori_ptmedie.Length - 1] + iculori_ptmedie[0]) / 2;
	//
```

```
MessageBox.Show("Media celor doua valori -- extreme -- de pe imagini:\r\n\r\n" +
                imedie_valoriminmax.ToString());
//
//acum afisam o descriere scurta a evolutiei, sau a starii cerului
//in functie de subdomeniul unde se situeaza media celor 5 valori:
//
string sCerul = "Starea de ansamblu a cerului: ";
//
//acum comparam media celor 5 valori cu valorile care definesc
//cele 4 intervale alcatuite din 5 valori care includ cele 2 extreme:
//
if
        (
        (imedie_total >= iculori_ptmedie[0])
        &
        (imedie_total < iculori_ptmedie[1])
        )
{
        MessageBox.Show(sCerul + stexte[4]);
}
else if
        (
        (imedie_total >= iculori_ptmedie[1])
        &
        (imedie_total < iculori_ptmedie[2])
        )
{
        MessageBox.Show(sCerul + stexte[3]);
}
else if
        (
        (imedie_total >= iculori_ptmedie[2])
        &
        (imedie_total < iculori_ptmedie[3])
        )
{
        MessageBox.Show(sCerul + stexte[2]);
}
else if
        (
        (imedie_total >= iculori_ptmedie[3])
        &
        (imedie_total < iculori_ptmedie[4])
        )
{
        MessageBox.Show(sCerul + stexte[1]);
}
```

```
        else if
                (
                (imedie_total >= iculori_ptmedie[4])
                &
                (imedie_total < iculori_ptmedie[3])
                )
        {
                MessageBox.Show(sCerul + stexte[0]);
        }

        else if (imedie_total == iculori_ptmedie[0])
        {
                //daca media tuturor valorilor este chiar cea mai mica valoare:
                MessageBox.Show(sCerul + stexte[4]);
        }
        else if (imedie_total == iculori_ptmedie[4])
        {
                //daca media tuturor valorilor este chiar cea mai mare valoare:
                MessageBox.Show(sCerul + stexte[0]);
        }
        else
        {
                MessageBox.Show(" ? " + sCerul + " ? " + stexte[0]);
        }
}
```

Vom mai lucra la programul acesta.
Dar logica creării sale şi procedura de bază sunt cele prezentate.

Recapitulare (Reamintire)

În mediul de programare Visual Studio, unde se folosesc entităţi din platforma. NET,
punctul
.
poate fi considerat cel mai important caracter folosit în cod
fiindcă

punctul . este delimitatorul nivelelor
pe care se află entităţi cu caracteristici specifice, distincte
entităţi care derivă din cele (ori, sunt asociate celor) de pe nivelul superior

Aici, "1" denotă nivelul de bază din care decurg următoarele nivele pe care se află entităţi:

1. Spaţiul de nume în care se află toate entităţile
2. Membri ai spaţiului de nume
3. Fiecare membru are Proprietăţi, Metode, Evenimente ş.a.

Ca atare
orice
obiect,
variabilă,
constantă,
colecţie,
Control (de exemplu, un *TextBox*)
Tip (de exemplu, *NameValueCollection*),
Clase

orice astfel de entitate
având Proprietăţi şi/sau Metode şi/sau Evenimente asociate
poate fi accesată şi folosită prin intermediul acestora.

Exemplu (1):

Un *TextBox* denumit "tb"

În cod, acesta va fi apelat după denumirea sa:
tb

Iar Proprietăţile, Metodele, Evenimentele ş.a. asociate acestui tip de Control vor putea fi accesate după ce imediat după denumirea sa este scris caracterul **punct .** astfel:

tb.ForeColor = Color.Blue;

Exemplu (2):

O variabilă de tip numeric *float* denumită "fvar"

În cod, aceasta va fi apelată după denumirea sa:
fvar

Iar Proprietăţile, Metodele, Evenimentele ş.a. asociate acestui Tip numeric vor putea fi accesate după ce imediat după denumirea sa este scris caracterul **punct .** astfel:

fvar.ToString();

Exemplu (3):

Un Tip, de exemplu *String*

În cod, acest Tip va fi apelat după denumirea sa:
String

Iar Proprietăţile, Metodele, Evenimentele ş.a. asociate acestui Tip numeric vor putea fi accesate după ce imediat după denumirea sa este scris caracterul **punct .** astfel:

String.Parse(...

Exemplu (4):

O Metodă declarată *public static*

```
public static string MetodaXYZ(int inumar, string stext)
{
        stext = inumar.ToString();
        return stext;
}
```

În cod, această metodă va fi apelată după denumirea sa precedată de cea a clasei:

clasa.MetodaXYZ(...

Astfel, de la nivelul de bază, generic, pe care se află spaţiul de nume
folosind **punctul .**
se coboară până la Proprietăţi ale unei entităţi din acel spaţiu de nume.
de exemplu:

- în cod, având o variabilă "str" de tipul *String*
  - lungimea sa în caractere poate fi ***aflată*** astfel:
    - **str.Length**
- în cod, având o variabilă "nvc" de tipul colecţiei *NameValueCollection*
  - proprietatea sa "Count" care întoarce numărul elementelor poate fi ***aflată*** astfel:
    - **int inumelemente = nvc.Count;**
- în cod, având o variabilă "tb" de tipul *TextBox* (poate fi un *TextBox* plasat pe *Form*-a respectivă)
  - proprietatea sa "MultiLine" poate fi ***aflată şi stabilită*** astfel:
    - **bool btbml = tb.MultiLine;**
    - **tb.MultiLine = true;**
  - proprietatea sa "Text" poate fi ***aflată şi stabilită*** astfel:
    - **string stext = tb.Text;**
    - **tb.Text = "abcd efgh ijkl";**

În cod,
toate Proprietăţile unui obiect pot fi accesate
dar
nu toate pot fi schimbate.

Printre Proprietăţile care
- nu pot fi stabilite în timpul scrierii codului, adică *nu se pot constitui într-o declaraţie*
- nu pot fi schimbate în timpul rulării programului

sunt:
- *Length* a unui String (text)
- *Count* a unei colecţii
- *Created* a unui Control de interfaţă (de exemplu, *TextBox* şi *Button*)

Atunci când se proiectează un program cu interfaţă - adică, unul care va conţine sau care va fi alcătuit din cel puţin o *Form*-ă, este recomandat ca proiectul să fie alcătuit astfel:

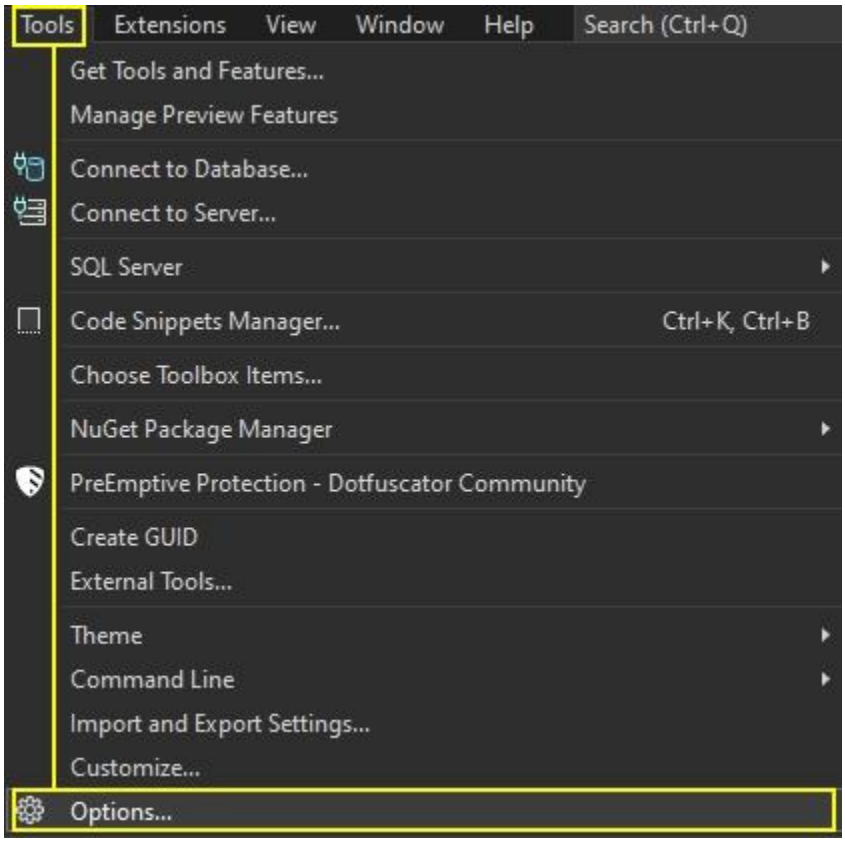
- pentru ca programatorul să poată crea mai departe fluxul operaţiunilor, adică efectiv să stabilească mersul lucrurilor în program, întâi se creează interfaţa
- apoi se scriu instrucţiuni pentru Evenimentele la care trebuie să răspundă unele Controale
- apoi, dacă este cazul:
    - secvenţe de mai mult de 2 instrucţiuni apelate repetat se plasează în Metode care vor fi apelate în locul acelor secvenţe. Astfel, codul este:
        - scurtat
        - organizat mai bine
        - făcut mai lizibil
- apoi, dacă este cazul:
    - secvenţe lungi de instrucţiuni apelate repetat (de exemplu, de la 10 instrucţiuni încolo) se plasează în Metode care la rându-le se plasează în Clase
        - dacă este posibil, acele Metode se declară ***public static*** pentru a nu fi nevoie să fie apelate după instanţierea Clasei unde se află, ci prin ***apelarea lor direct după numele clasei folosind de asemenea punctul .*** cum se vede în Exemplul 4.

În unele situaţii, codul scris în Visual Studio poate fi – în mod automat:
- completat
- documentat parţial

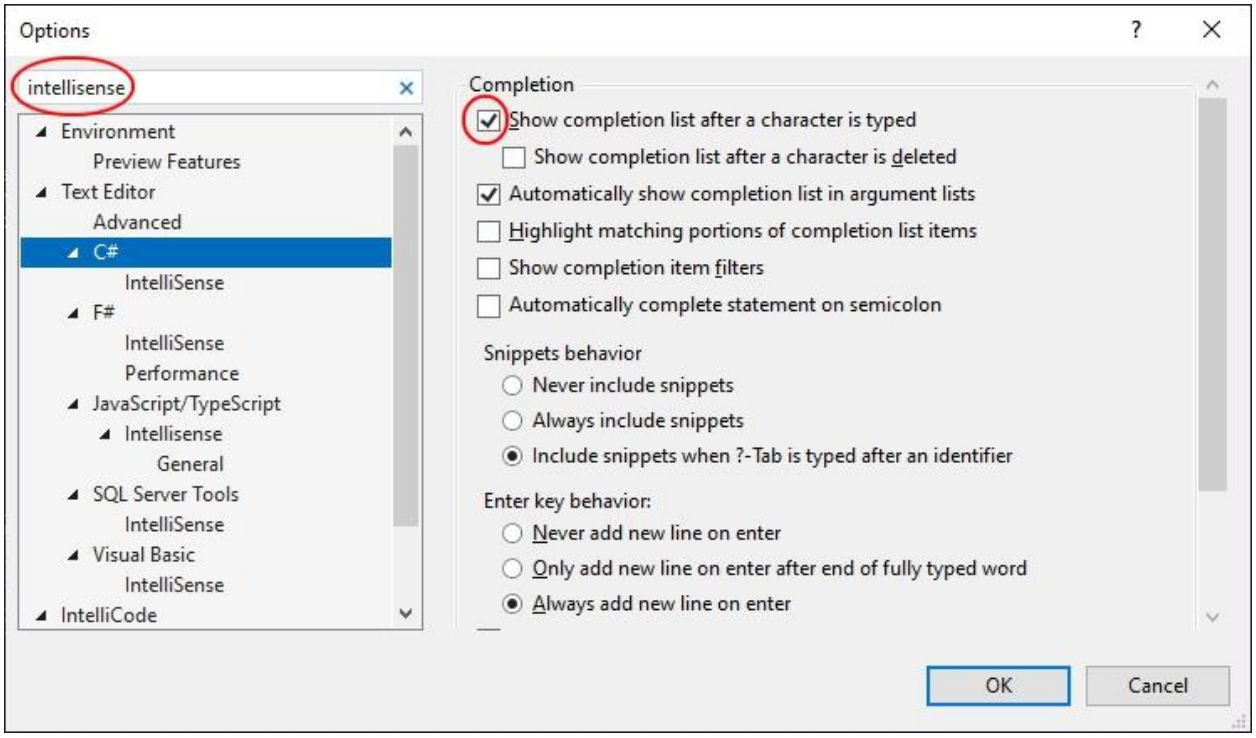
De exemplu, după ce se scriu două litere, programatorului îi pot fi propuse denumirile entităţilor a căror denumire începe cu acele litere – încât să nu scrie toate literele din acea denumire, ci să aleagă:
- fie din acele entităţi definite numai în *scopul* (pe nivelul) curent de cod: variabile, metode ş.a.
- fie dintr-unele cunoscute ca membri ai spaţiilor de nume a căror utilizare s- a anunţat la începutul codului scris în acel moment, prin secvenţa *using* dinaintea oricăror instrucţiuni
- fie dintr-unele cunoscute ca denumiri ale unor clase definite drept componente ale programului

Această funcţionalitate se numeşte (sau se bazează pe noţiunea de) ***Intellisense***.

În Visual Studio, opţiunile corespunzătoare acesteia se accesează prin meniul *Tools*:



În caseta de dialog unde se stabilesc opţiunile pentru mediul de programare,
pentru accesarea rapidă a celor pentru *Intellisense* se scrie acest termen în caseta de editare/căutare din stânga sus:



Cea mai importantă opţiune este cea de a lăsa Visual Studio să propună entităţile a căror denumire începe cu literele scrise în acel timp – anume, opţiunea marcată "*completion list*" sau similar:

"Show completion list after a character is typed"
("Arată lista pentru completare după fiecare caracter/literă scris/ă")

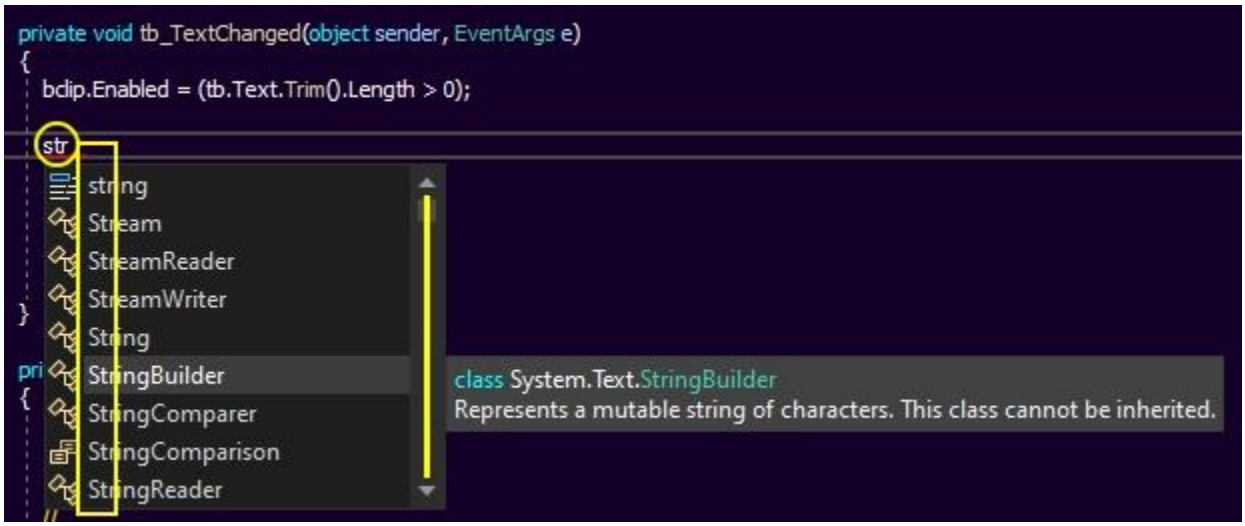
Dacă este bifată această opţiune, atunci în cod va fi reflectată de apariţia acelei liste din care pot fi alese entităţi de interes în acel punct al programului, de exemplu:

De exemplu:

În codul unei clase în a cărei secvenţă ***using*** există

using System.Text;

într-o metodă în care tocmai scriem "**str**"
după aceste litere apare lista din care putem alege ceva:



Din lista astfel afişată poate fi selectată o entitate a cărei denumire să fie AutoComplet-ată, astfel:

- navigând până la acea entitate apăsând tastele marcate cu săgeţi în sus şi în jos şi...
    - apăsând...
        - tasta de spaţiu/blank (recomandat)
        - tasta Enter
        - tasta Tab

Bineînţeles, în acest mod...

- se scurtează timpul de scriere a codului
- se pot revizui entităţile folosite în program...
    - printre care unele pot fi inutile
    - dintre care unele pot lipsi (de exemplu, dacă vrem a folosi tipul *StringBuilder* aflat în spaţiul de nume *System.Text*, dar denumirea aceasta nu apare în listă, înseamnă că nu există declaraţia **using System.Text;** şi ca urmare trebuie să scriem aceasta în secvenţa ***using*** dinaintea codului.

Pentru a evita listinguri lungi de cod, de exemplu litanii de zeci şi sute de instrucţiuni, indiferent dacă sunt
- declaraţii/atribuiri de variabile/obiecte
- comentarii la cod
- şi diverse alte linii prezente în cod

lungimea ***percepută*** a codului (cea reală nu este afectată)
poate fi scurtat folosind directivele

#region
#endregion

Acestea definesc porţiuni de cod care pot fi restrânse vizual la o singură linie pe care să scrie "#region...".
#region marchează începutul secvenţei care trebuie restrânsă
#endregion marchează sfârşitul secvenţei care trebuie restrânsă - inclusiv linia aceasta

Astfel, cuprinzând între aceste directive o secvenţă de cod lungă de 100 linii micşorăm lungimea percepută a codului cu 100 linii în editorul de cod (numai vizual, pe ecran).

O regiune deschisă cu **#region** trebuie închisă cu **#endregion**.
Aceste directive trebuie să se afle pe acelaşi nivel

Adică:

Dacă vrem să "scurtăm" o porţiune de cod dintr-o metodă care arată astfel:

```
private double dmedia_aritmetica(double[ ] dValoriPentruMediereAritmetica)
{
	//aceasta metoda intoarce un numar real
	//care denota media aritmetica a valorilor numerice reale
	//primite in argumentul dValoriPentruMediereAritmetica
	//si face si alte lucruri
	//
	//algoritmul este cel clasic: insumare, apoi impartire
	double dsuma = 0;
	double dmedia_aritmetica = 0;
	for (int i = 0; i < dValoriPentruMediereAritmetica.Length; i++)
	{
		dsuma += dValoriPentruMediereAritmetica[i];
	}
	//calculam media, impartind suma la numarul valorilor:
	dmedia_aritmetica = dsuma / dValoriPentruMediereAritmetica.Length;
	//
	return dmedia_aritmetica;
	//
	//
	//... urmeaza o mie de linii de cod, comentarii s.a.
	//in legatura, bineinteles, cu functonalitatea acestei metode
	//anume, calcularea mediei aritmetice a unui sir de valori numerice reale
	//
}
```

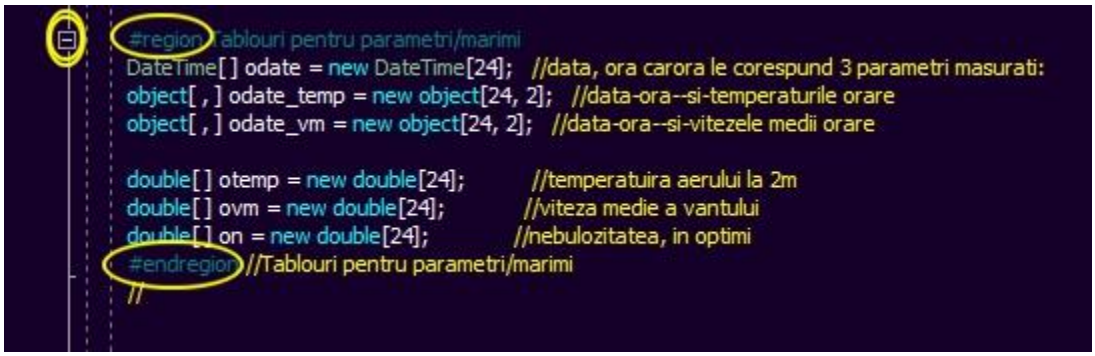
Vom folosi directivele **#region** şi **#endregion** astfel:
atenţie la nivelul pe care se află fiecare:

```
private double dmedia_aritmetica(double[ ] dValoriPentruMediereAritmetica)
{
        //aceasta metoda intoarce un numar real
        //care denota media aritmetica a valorilor numerice reale
        //primite in argumentul dValoriPentruMediereAritmetica
        //si face si alte lucruri
        //
        //
        #region Calcularea mediei aritmetice in stilul clasic
        //
        //algoritmul este cel clasic: insumare, apoi impartire
        double dsuma = 0;
        double dmedia_aritmetica = 0;
        for (int i = 0; i < dValoriPentruMediereAritmetica.Length; i++)
        {
                dsuma += dValoriPentruMediereAritmetica[i];
        }
        //calculam media, impartind suma la numarul valorilor:
        dmedia_aritmetica = dsuma / dValoriPentruMediereAritmetica.Length;
        //
        #endregion //Calcularea mediei aritmetice in stilul clasic
        //
        //
        //... urmeaza o mie de linii de cod, comentarii s.a.
        //in legatura, bineinteles, cu functonalitatea acestei metode
        //anume, calcularea mediei aritmetice a unui sir de valori numerice reale
        //
        //
        //rezultatul intors de aceasta metoda:
        return dmedia_aritmetica;
}
```

În acest exemplu
am plasat corect
directiva **#endregion**
pe acelaşi nivel cu directiva **#region**

Vedem cum:

- înaintea liniei pe care se află directiva **#region** apare **semnul minus** -
  - Apăsarea pe semnul **minus** determină restrângerea/colaps-area codului
    - de la linia următoare celei pe care se află directiva **#region** până la linia pe care se află directiva **#endregion**, inclusiv





Mai mult – dacă după **#region** scriem un text, ca mai sus (descriptiv al secvenţei de cod astfel restrânse) odată ce restrângem acea secţiune

acel text va apărea în cod, precedat de **semnul plus**

care indică o regiune ce poate fi extinsă

- Apăsarea pe semnul **plus** determină extinderea/desfăşurarea codului

de la linia pe care se află directiva **#region**, inclusiv până la linia pe care se află directiva **#endregion**, inclusiv

F5

➔ lansează în execuţie (porneşte) programul deschis în acest moment ca Proiect singular/Soluţie compusă dintr-unul sau mai multe Proiecte

Ctrl Alt L

➔ afişarea secţiunii *Solution Explorer*
- unde putem accesa toate componentele programului/proiectului/soluţiei deschise

În modul de scriere a codului:

Shift F7
➔ afişarea interfeţei (dacă există) Form-ei sau Controlului asociat/ă codului respectiv

Home
➔ la începutul liniei curente, pe care se află cursorul de text

End
➔ la sfârşitul liniei curente, pe care se află cursorul de text

Ctrl Home
➔ la începutul codului clasei ş.a. afişate

Ctrl End
➔ la sfârşitul codului clasei ş.a. afişate

Page Up
➔ o pagină în sus

Page Down
➔ o pagină în jos

Ctrl Săgeata-Sus
➔ derulare o linie în sus

Ctrl Săgeata-Jos
➔ derulare o linie în jos

F9
după click sau plasarea cursorului pe o linie de cod
➔ inserarea sau ştergerea dacă există a unui punct de pauză ("Breakpoint") pe acea linie
- programul se va opri la acea instrucţiune/la acel bloc de instrucţiuni, încât programatorul să inspecteze ce s-a întâmplat până acolo –
  - de exemplu:
    - plasarea pointer-ului de mouse pe denumirea unei variabile va arăta valoarea din acel moment a acelei variabile
- programul va trece de acel punct de pauză după apăsaerea pe tasta **F5**

F12

după click pe o entitate declarată în cod

➔ la linia pe care se află declararea/definirea acelei entităţi

Click-dreapta

➔ afişarea unui meniu cu comenzi pentru modul de scriere a codului

- în care se află comenzi ca "Remove and Sort Usings"

În modul de Proiectare/Design a unei Form-e sau a unui Control:

F7

➔ afişarea codului Form-ei

F4

după click pe un Control sau pe Form-ă

➔ afişarea secţiunii *Properties*

- unde putem stabili Proprietăţi ale
  - tuturor Controalelor de pe o Form-ă,
    fie acelea şi invizibile în timpul rulării programului (*Timer*, *OpenFileDialog* ş.a.)
  - Form-ei înseşi,
  - chiar ale unor clase

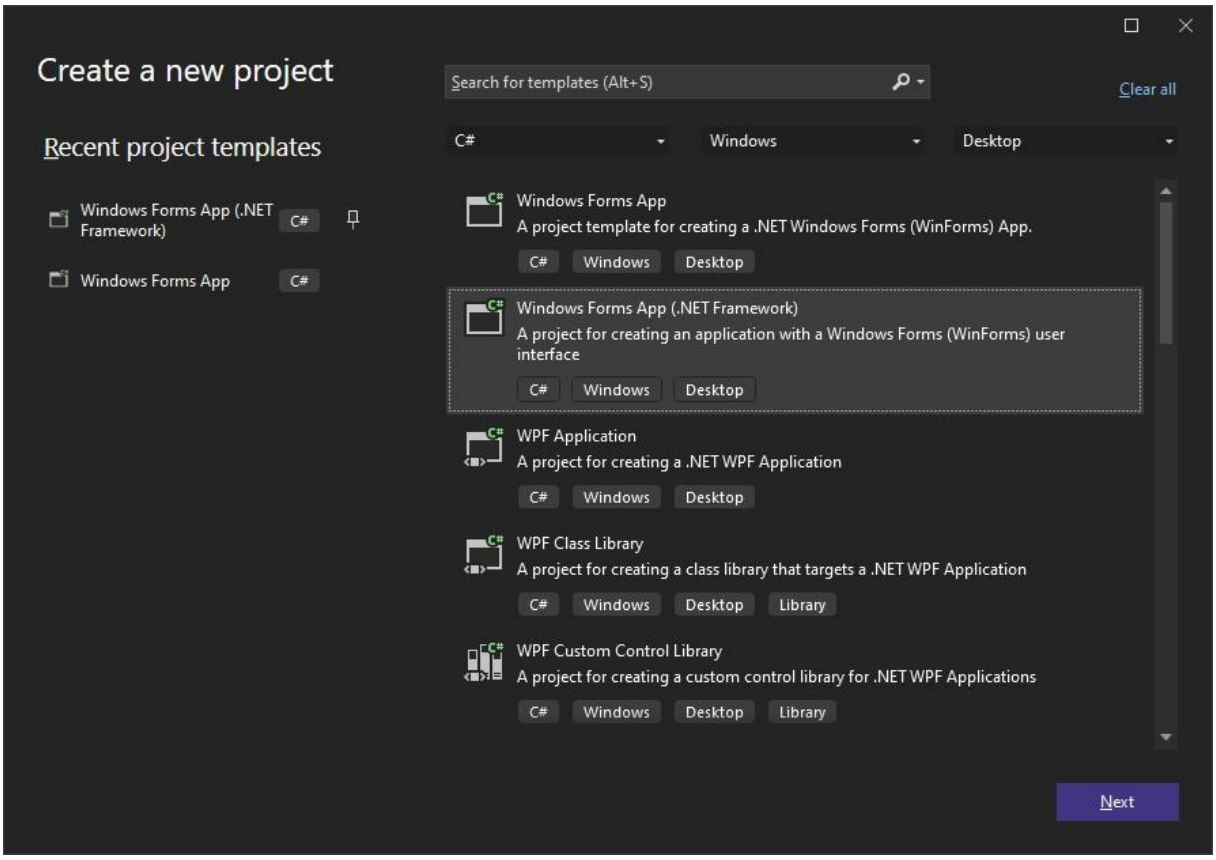
Folosirea claselor: porţiuni de cod cu funcţionalitate distinctă de a altora

Uneori, codul unei Form-e poate fi /foarte/lung şi/sau dificil de citit/parcurs şi de înţeles, cu sute de instrucţiuni şi comentarii înşirate pe mii de linii, încât într-adevăr codul să pună probleme de înţelegere chiar celui/celor care l-au scris, darmite celor din altă echipă sau din aceeaşi echipă cu cea a acelora.

Pentru prezentarea celor ce se pot face încât să se ajungă la o organizare mai bună a codului unui program inclusiv în vederea parcurgerii mai facile, vom folosi tot mediul de programare Visual Studio 2022.

Creăm un Proiect de tipul celor prezentate aici – unul pentru un program care conţine Form-e (deci care oferă posibilitatea de interacţiune vizuală), cum deja se ştie:

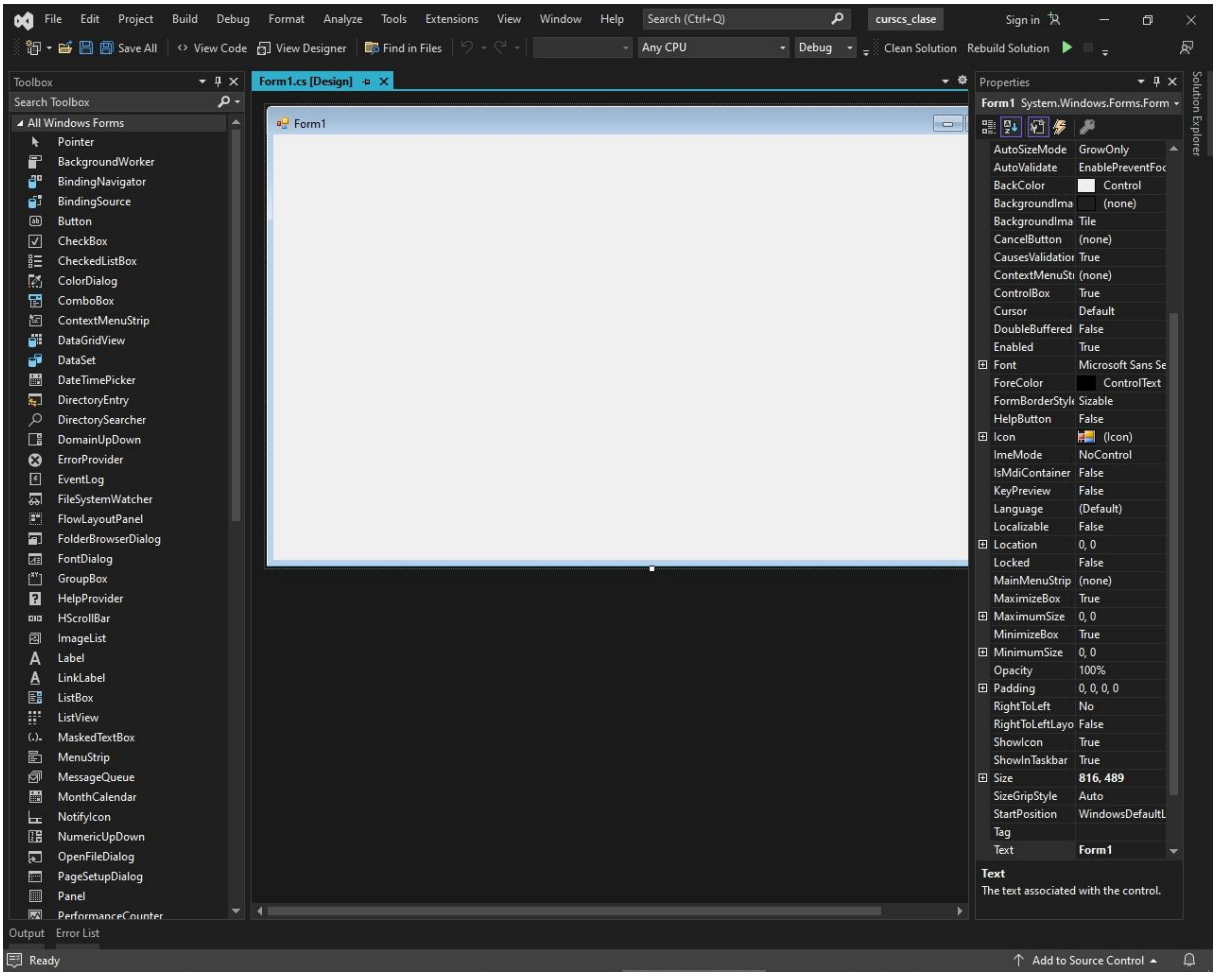
- pornim Visual Studio
- meniul File -> New -> Project...



Nu contează ce denumire specificăm, va fi un program simplu care calculează medii aritmetice.

Aici, programul se numeşte "curscs_clase".

Ca de obicei, Visual Studio ne prezintă o Form-ă creată implicit şi denumită "Form1":



Planul programului nostru este simplu: vrem să...
- calculăm media aritmetică a unui tablou de numere reale. Aceasta se va face la apăsarea unui buton
- afişăm acea valoare într-un TextBox.

Avantajul de a folosi TextBox în loc de o etichetă Label care de asemenea permite afişarea unui text este: spre deosebire de textul dintr-o Label cu care nu se poate interacţiona prin mouse şi tastatură, textul afişat într-un TextBox poate fi:

- *Selectat*
- *Copy-at*
- ***şi dacă Proprietatea ReadOnly este False** –*
    - *editat (modificat) prin scriere direct în TextBox,*
    - *înlocuit parţial sau total cu text adus din Clipboard, folosind comanda "Paste"*
        - care la fel cu "Copy", "Cut", "Select All" ş.a. apare în meniul contextual al TextBox-ului afişat după click-dreapta în interiorul acestui tip de Control

Având planul programului, proiectăm interfaţa:

Micşorăm Form-a care acum este prea mare pentru rostul său – anume, cel de a prezenta un număr.

O Form-ă şi de fapt majoritatea Controalelor cu contur vizibil pe o Form-ă pot fi redimensionate astfel:
- trăgând de colţul din dreapta-jos
- stabilind Proprietatea "*Size*"

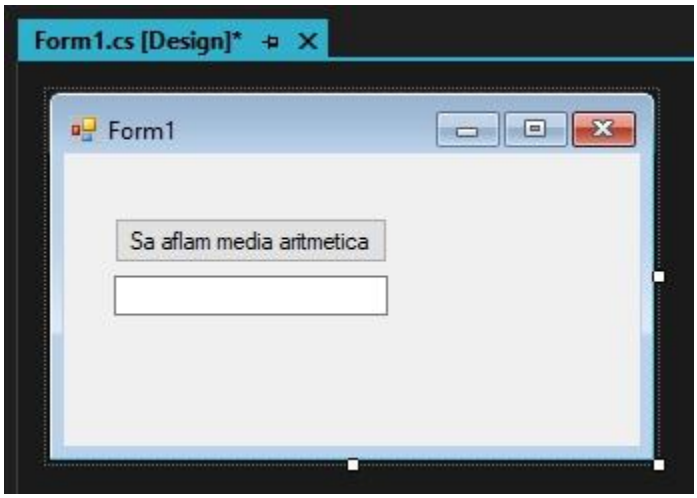
Va fi nevoie de un Button a cărui apăsare
(Evenimentul *Click* sau un Eveniment similar – de obicei, *MouseUp*)
va invoca procedura de calculare a mediei aritmetice.

Va fi nevoie şi de un TextBox
unde va fi afişată media aritmetică.

Aşadar, din Toolbox, pe Form-ă tragem
un Button şi un TextBox
pe care le redenumim (Proprietatea "*Name*"), respectiv, la "b" şi "tb".

Pentru "b", stabilim Proprietatea "*Text*" la " Sa aflam media aritmetica".

Form1 ar avea acest aspect:



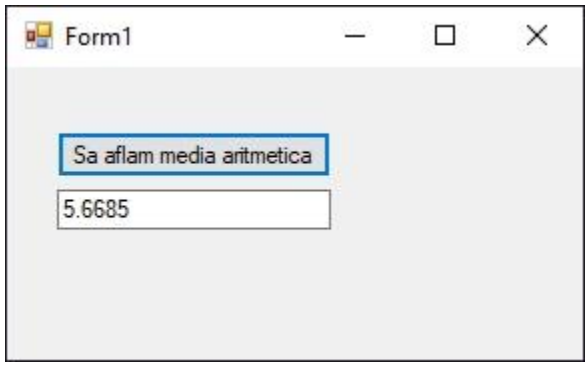
Să scriem cod.

Dublu-click pe buton, ceea ce ne trimite în modul de scriere a codului, în metoda Evenimentului *Click* al acestui buton, metodă în care scriem instrucţiunile de calculare a mediei aritmetice a unor numere reale:

```
private void b_Click(object sender, EventArgs e)
{
	//numerele a caror medie aritmetica vrem sa o calculam
	//sunt presupuse cunoscute
	//si aflate intr-un tablou unidimensional (o lista simpla, matrice cu 1 linie) de tipul double
	//adica, un tablou care contine numai valori de tipul double:
	//
	double[ ] dnumerepentrumedie = new double[ ] { 1, 2, 3, 4.25, 5.5, 6.75, 7, 8.125, 9.06, 10 };
	//
	//media aritmetica va fi un numar real
	//al carui tip il propunem a fi, de asemenea, double
	//acest tip este suficient pentru operatiuni chiar si cu numere foarte mari
	//
	//algoritmul generic pentru calcularea mediei aritmetice este:
	//
	//initializam cu valoarea 0 zero media aritmetica:
	//
	double dmedar = 0;
	//
	//initializam cu 0 zero suma numerelor:
	//
	double dsuma = 0;
	//
	//la fiecare iteratie cu variabila de ciclare i
	//dsuma va fi incrementata cu numarul de la indexul curent (i) in tablou:
	//
	for (int i = 0; i < dnumerepentrumedie.Length; i++)
	{
		dsuma += dnumerepentrumedie[i];
	}
	//acolada de deasupra a inchis ciclul for
	//la sfarsitul caruia, adica in acest punct,
	//avem suma numerelor din tablou
	//
	//pe care o impartim la numarul elementelor din tablou
	//pentru a afla media artmetica:
	//
	dmedar = dsuma / dnumerepentrumedie.Length;
	//
	//afisam valoarea mediei aritmetice in TextBox:
	//
	//intai stergem textul care poate exista in TextBox:
	//
	tb.Clear();
	//
	//aici, operatiunea aceasta e inutila fiindca urmeaza tb.Text = ...
	//adica --- inlocuirea --- intregului Text din TextBox
	//asadar, putem comenta linia cu tb.Clear();
	//
	tb.Text = dmedar.ToString();
}
```

Pornim programul: apăsăm tasta **F5**:
Rezultatul:

După ce apăsăm pe buton este afişată media atitmetică a numerelor din codul de mai sus:



Uitându-ne la cod vedem cum numai această funcţionlaitate relativ neînsemnată ocupă, cu tot cu comentariile, vreo 60 linii.

Probabil că nu am creat această Form-ă numai pentru a afişa un număr, ci vom mai adăuga Controale şi cod asociat lor.
Astfel, codul pentru *Form1* se va lungi la sute şi chiar mii de linii.

Totodată, acum avem gata pregătit şirul de numere a căror medie aritmetică va fi calculată,
în ideea că vom repeta această operaţiune pentru acel şir.

În aceste cazuri – când una sau mai multe operaţiuni sunt efectuate în mod repetat,
pentru a organiza codul
şi pentru a separa o anumită funcionalitate a programului de Dialogul (Form-a) sau de contextul curent al unei anumite secvenţe de cod
se pot plasa acele operaţiuni în Clase dedicate lor.

Vom face aşa ceva în programul nostru, fiindcă pur şi simplu vrem să scurtăm codul *Form1*.

Adică, vrem să mităm calcularea mediei aritmetice într-o clasă distinctă, separată de *Form1*.

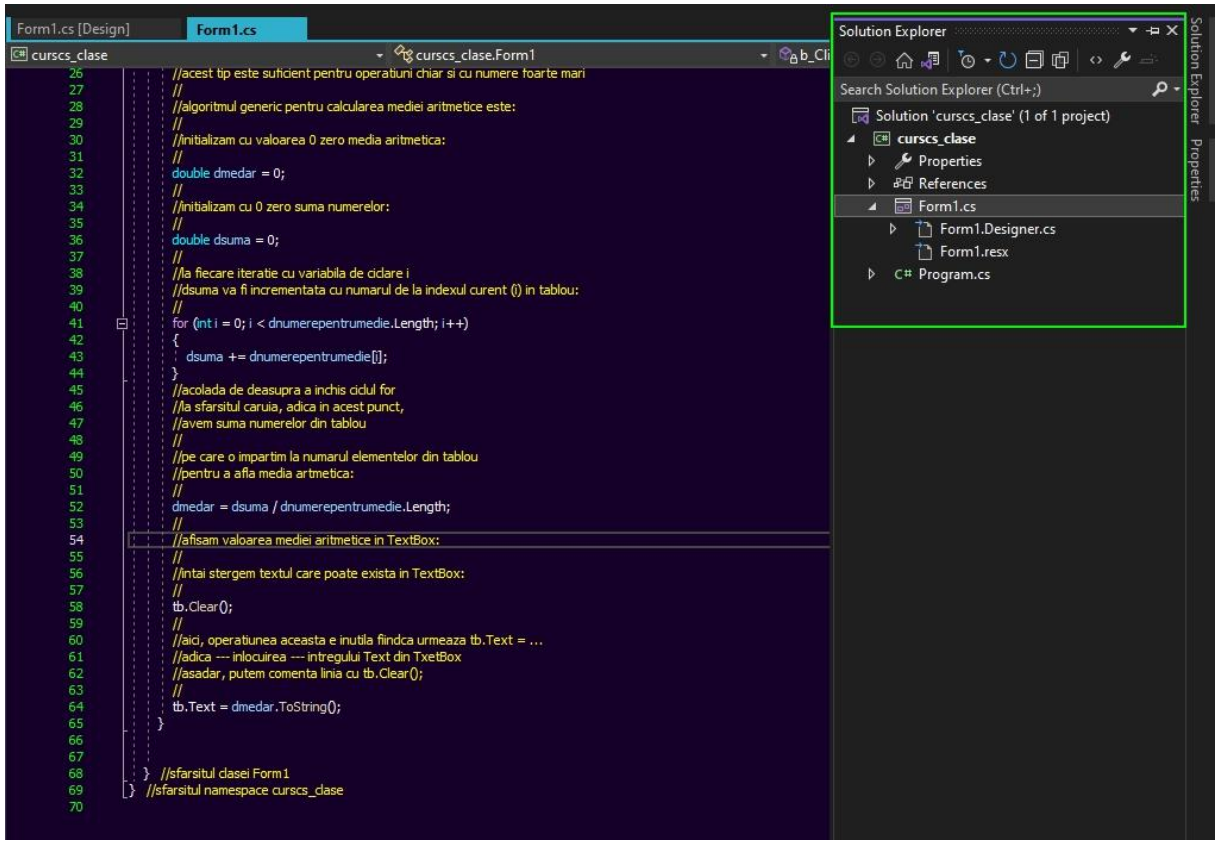
De asemenea, poate vom intenţiona să scriem cod pentru alte operaţii matematice care vor sta foarte bine lângă calcularea mediei aritmetice.

Deci putem crea o clasă dedicată operaţiilor matematice utile în programul nostru.

Aceasta se face astfel:

Solicităm Visual Studio afişarea secţiunii "Solution Explorer"
unde sunt prezentate toate componentele programului/proiectului/soluţiei curente:
apăsăm tastele

Ctrl Alt L

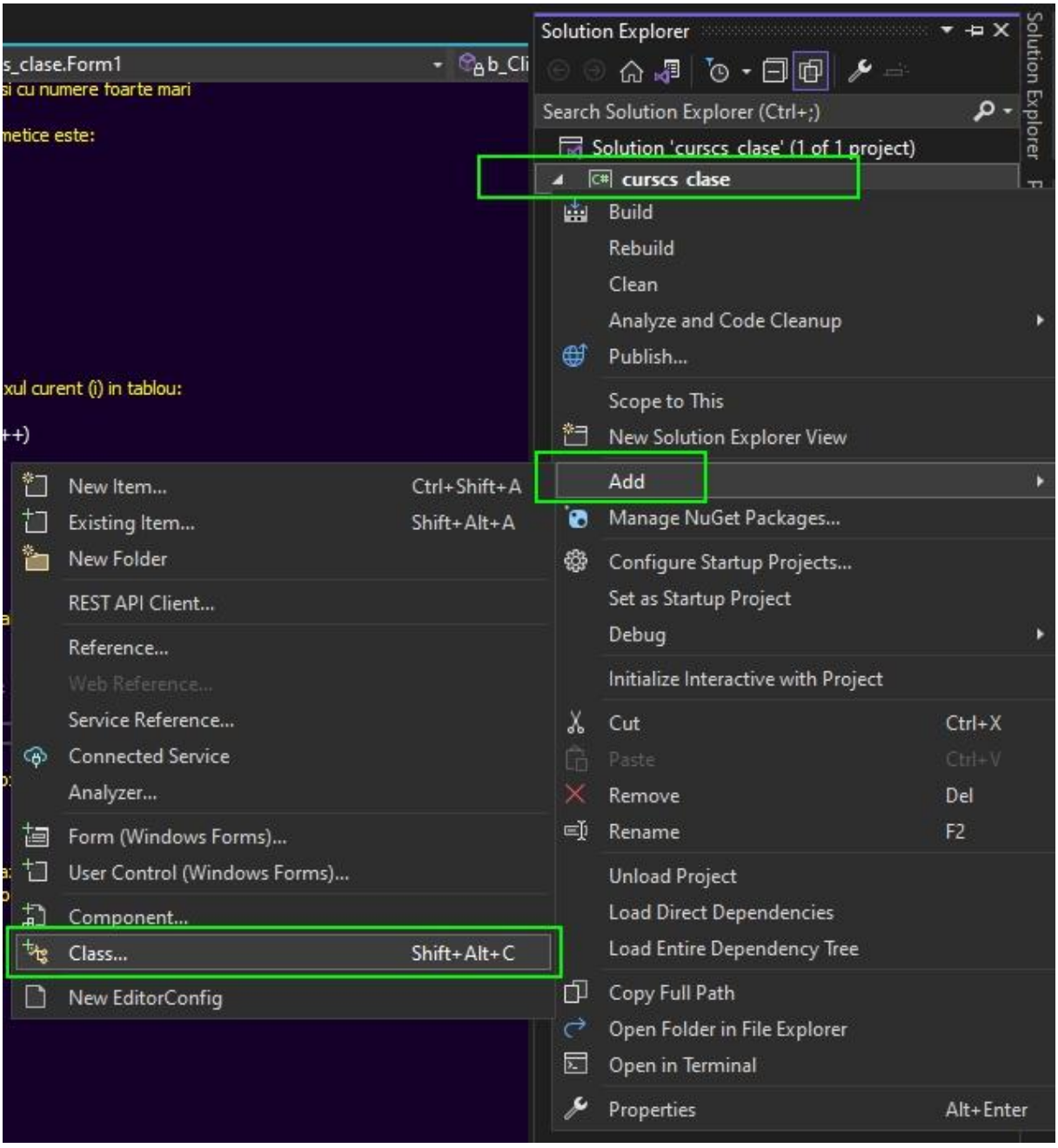


În structura arborescentă a Soluţiei curente sunt afişate:

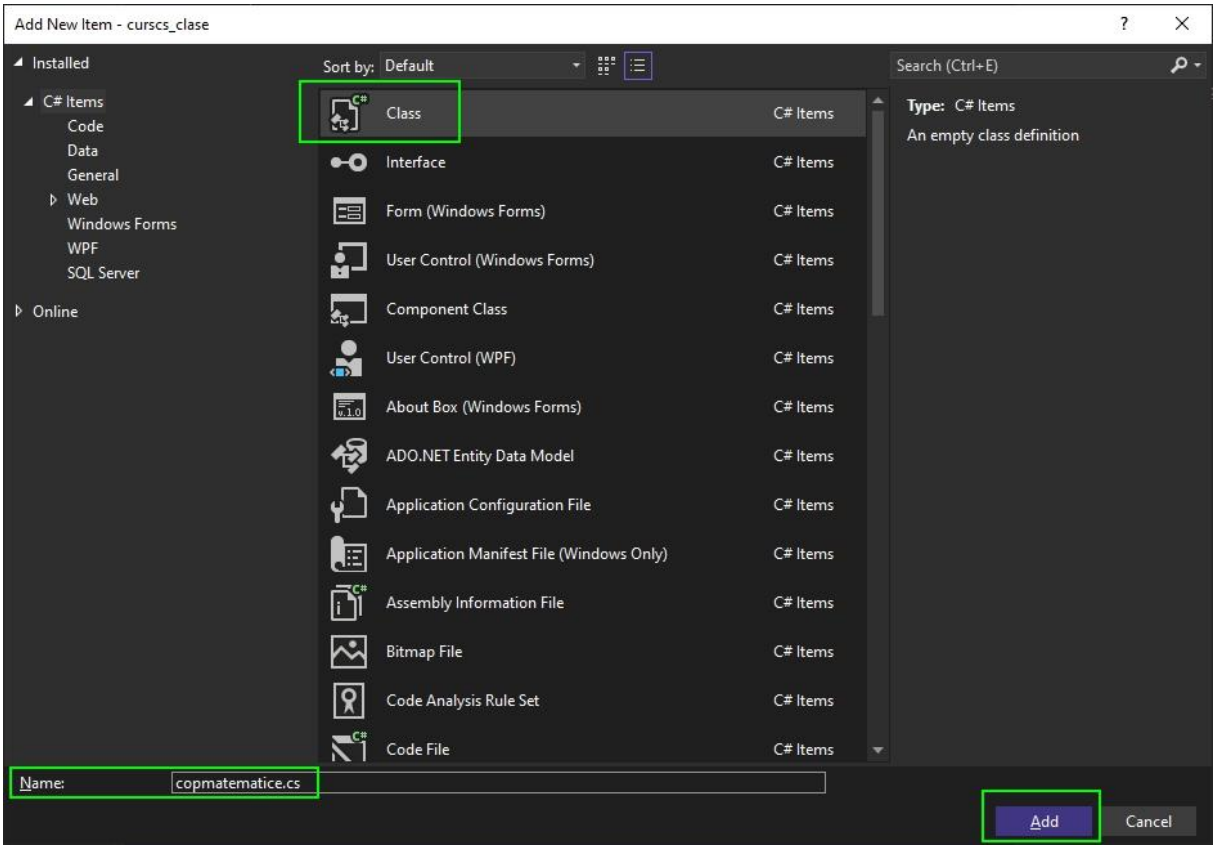
- numele Proiectului curent: "curscs_clase".
- numele Form-ei care se află în Proiect: "Form1"

La acest Proiect adăugăm o clasă pe care o dedicăm operaţiilor matematice:

Click-dreapta pe numele Proiectului:
Apoi pe intrarea de meniu "Add"
Apoi pe inrarea de meniu "Class..."



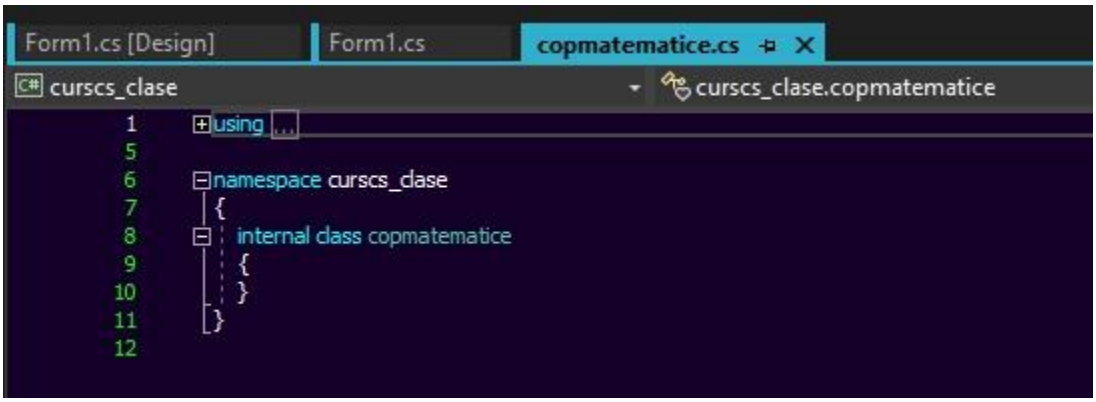
Denumim clasa "copmatematice.cs":
Apoi apăsăm butonul "Add":



Visual Studio ne prezintă codul noii clase.

Ca şi codul *Form1*, codul acestei clase este plasat în spaţiul de nume având denumirea Proiectului/Soluţiei:

namespace curscs_clase



Adică, tot ce se întâmplă în codul acestei clase va fi cunoscut în interiorul Proiectului/Soluţiei cu această denumire (acest nume) din acest program.
Un program numit la fel, compilat în fişier cu extensia .exe sau alta, va fi ceva distinct.

Orice altă denumire decât cea a acelui spaţiu de nume va face inaccesibilă această clasă.

Acum putem aduce în această clasă
operaţiunile matematice din *Form1*
pentru calcularea mediei aritnetice.

Aşadar, înapoi la codul *Form1*.

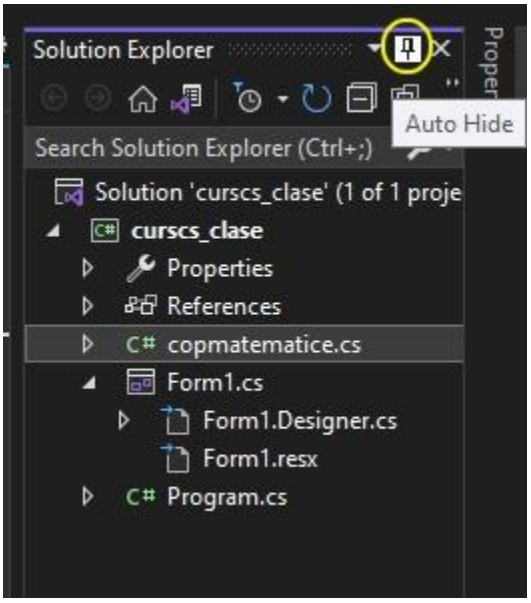
Se pare că am pierdut pe undeva acel cod, însă e numai o părere.

Form1 este parte a aceluiaşi proiect cu clasa al cărei cod îl vedem acum,
astfel că poate fi accesată - ca interfaţă şi cod – din "Solution Explorer".

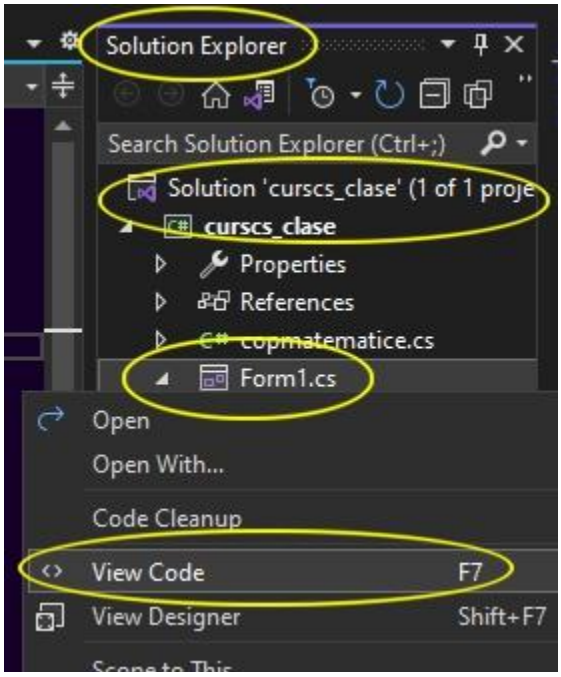
Aşadar, solicităm iarăşi afişarea "Solution Explorer" apăsând tastele

Ctrl Alt L

Ca aproape oricare altă secţiune a mediului de programare Visual Studio,
"Solution Explorer" poate fi făcută să rămână afşată permanent –
apăsând pe "acul" din colţul său din dreapta-sus:



Click-dreapta pe "Form1.cs"
şi apoi pe intrarea de meniu "View Code":



Aşa ajungem înapoi în codul *Form1*,
de unde copiem instrucţiunile din metoda Evenimentului *Click* pe butonul "b1"
pe care le vom muta în clasa tocmai creată, "copmatematice.cs"

copiem instrucţiunile de până la cele care se referă la TextBox-ul "tb"

fiindcă numele Controalelor de pe orice Form-ă
nu sunt cunoscute implicit într-o clasă separată de Form-a care le conţine

Selectăm acea porţiune de cod ca pe oricare alt text
şi îl plasăm în Clipboard apăsând tastele

Ctrl C

Odată ce am plasat astfel codul în Clipboard,
ne întoarcem la clasa unde vrem să-l ducem:
în "Solution Explorer"
click-dreapta pe "copmatematice.cs"
apoi "View Code"

Sub acolada de sub instrucţiunea care iniţializează clasa,
adică în corpul clasei,
definim o metodă dedicată instrucţiunilor de adus din codul *Form1*:

Se observă cum în corpul metodei se află numerele de interes.
Totodată, cum metoda nu are parametri/argumente.
Deci este o metodă al cărei rezultat va fi întotdeauna acelaşi.

Pentru a da o utilitate extinsă acestei metode, mai târziu vom crea o variantă a sa care calculează media aritmetică a numerelor primite ca parametru/argument.

```
public static double dMediaAritmetica()
{
	//numerele a caror medie aritmetica vrem sa o calculam
	//sunt presupuse cunoscute
	//si aflate intr-un tablou unidimensional (o lista simpla, matrice cu 1 linie) de tipul double
	//adica, un tablou care contine numai valori de tipul double:
	//
	double[ ] dnumerepentrumedie = new double[ ] { 1, 2, 3, 4.25, 5.5, 6.75, 7, 8.125, 9.06, 10 };
	//
	//media aritmetica va fi un numar real
	//al carui tip il propunem a fi, de asemenea, double
	//acest tip este suficient pentru operatiuni chiar si cu numere foarte mari
	//
	//algoritmul generic pentru calcularea mediei aritmetice este:
	//
	//initializam cu valoarea 0 zero media aritmetica:
	//
	double dmedar = 0;
	//
	//initializam cu 0 zero suma numerelor:
	//
	double dsuma = 0;
	//
	//la fiecare iteratie cu variabila de ciclare i
	//dsuma va fi incrementata cu numarul de la indexul curent (i) in tablou:
	//
	for (int i = 0; i < dnumerepentrumedie.Length; i++)
	{
		dsuma += dnumerepentrumedie[i];
	}
	//acolada de deasupra a inchis ciclul for
	//la sfarsitul caruia, adica in acest punct,
	//avem suma numerelor din tablou
	//
	//pe care o impartim la numarul elementelor din tablou
	//pentru a afla media artmetica:
	//
	dmedar = dsuma / dnumerepentrumedie.Length;
	//
	//
	//rezultatul intors de aceasta metoda este dmedar:
	//
	return dmedar;
}
```

Acum, înapoi în codul *Form1*, unde ştergem instrucţiunile plasate în clasă.

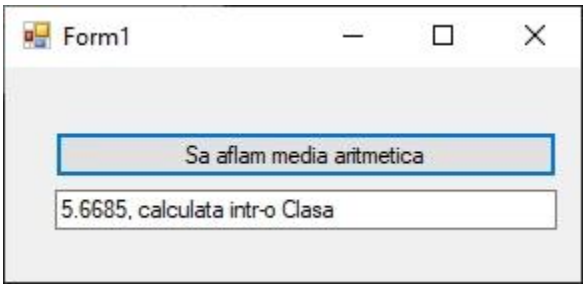
Le înlocuim cu instrucţiunea care apelează metoda de calculare a mediei aritmetice
a unor numere cunoscute în acea metodă
aflată în clasa dedicată unor operaţii matematice:

```
double dma = copmatematice.dMediaAritmetica();
```

Metoda b_Click o înlocuim cu:

```
private void b_Click(object sender, EventArgs e)
{
        //apelam metoda dMediaAritmetica()
        //din clasa copmatematice:
        //
        double dma = copmatematice.dMediaAritmetica();
        //
        //afisam valoarea mediei aritmetice in TextBox:
        //
        //intai stergem textul care poate exista in TextBox:
        //
        tb.Clear();
        //
        //aici, operatiunea aceasta e inutila fiindca urmeaza tb.Text = ...
        //adica --- inlocuirea --- intregului Text din TextBox
        //asadar, putem comenta linia cu tb.Clear();
        //
        tb.Text = dma.ToString()+ ", calculata intr-o Clasa";
}
```

Pornim programul: tasta **F5**:
Rezultatul:



Să creăm o variantă a acelei metode, încât să fie utilă pentru orice şir de numere:

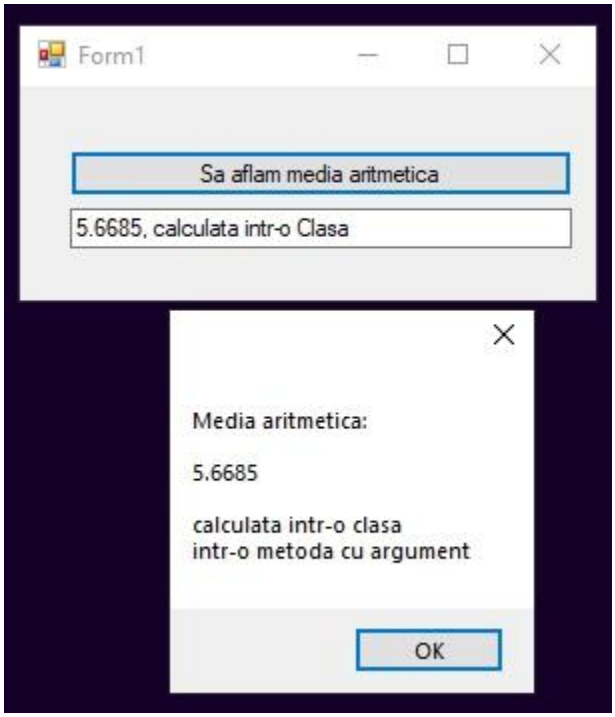
Pentru aceasta, copiem instrucţiunile din metoda existentă
dar *schimbăm semnătura* noii metode –
anume, o proiectăm ca primind un argument de tipul double[]:

```
public static double dMediaAritmetica(double[ ] dnumerepentrumedie)
{
        double dmedar = 0;
        double dsuma = 0;
        for (int i = 0; i < dnumerepentrumedie.Length; i++)
        {
                dsuma += dnumerepentrumedie[i];
        }
        dmedar = dsuma / dnumerepentrumedie.Length;
        return dmedar;
}
```

Înapoi în codul Form1, unde modificăm metoda lui b_Click:

```
private void b_Click(object sender, EventArgs e)
{
        //apelam metoda dMediaAritmetica()
        //din clasa copmatematice:
        //
        double dma = copmatematice.dMediaAritmetica();
        //
        //afisam valoarea mediei aritmetice in TextBox:
        //
        //intai stergem textul care poate exista in TextBox:
        //
        tb.Clear();
        //
        //aici, operatiunea aceasta e inutila fiindca urmeaza tb.Text = ...
        //adica --- inlocuirea --- intregului Text din TextBox
        //asadar, putem comenta linia cu tb.Clear();
        //
        tb.Text = dma.ToString() + ", calculata intr-o Clasa";
        //
        //acum apelam varianta cealalta a metodei din clasa
        //trimitandu-i ca argument acelasi (pentru compararea rezultatelor) sir de numere:
        //
        double[ ] dnumere = new double[ ] { 1, 2, 3, 4.25, 5.5, 6.75, 7, 8.125, 9.06, 10 };
        double dma_2 = copmatematice.dMediaAritmetica(dnumere);
        MessageBox.Show("Media aritmetica:\r\n\r\n" +
                                dma_2.ToString() +
                                "\r\n\r\ncalculata intr-o clasa\r\nintr-o metoda cu argument");
}
```

Poruim programul: tasta **F5**.
Rezultatul:



De foarte multe ori, pe timpul execuţiei

sub Visual Studio – adică, după apăsarea tastei F5 –
deci, nu ca program independent, ca fişier cu extensia .exe sau alta

a unei operaţiuni sau proceduri complexe sau chiar foarte simple
vrem să ştim ce anume se întâmplă în anumite instrucţiuni.

Pentru aceasta, trebuie să putem urmări execuţia acelor instrucţiuni –
de exemplu, ce valori sunt vehiculate în acel moment.

Aceasta înseamnă că programul trebuie oprit cumva până aflăm ce ne interesează şi apoi lăsat să ruleze.

Avem la dispoziţie trei metode (pot fi descoperite şi altele, dar acestea sunt suficiente):

1. inserarea în cod a unui **Breakpoint** (punct de pauză) pe instrucţiunea/linia unde se află ceea ce ne interesează
    - programul oprindu-se în acel punct
        - ţinând pointerul mouse-ului deasupra variabilei/enităţii de interes solicităm de la Visual Studio afişarea unor informaţii despre aceea
    - programul va trece de acel Breakpoint (va rula în continuare) după apăsarea tastei **F5** sau a celei echivalente pentru comanda "Continue" (continuarea execuţiei controlate de Visual Studio)

2. inserarea în cod a unui **Breakpoint** (punct de pauză) pe instrucţiunea/linia unde se află ceea ce ne interesează
    - programul oprindu-se în acel punct
        - scrierea în secţiunea "Immediate Window" a Visual Studio a unei instrucţiuni cu care solicităm informaţii
    - programul va trece de acel Breakpoint (va rula în continuare) după apăsarea tastei **F5** sau a celei echivalente pentru comanda "Continue" (continuarea execuţiei controlate de Visual Studio)

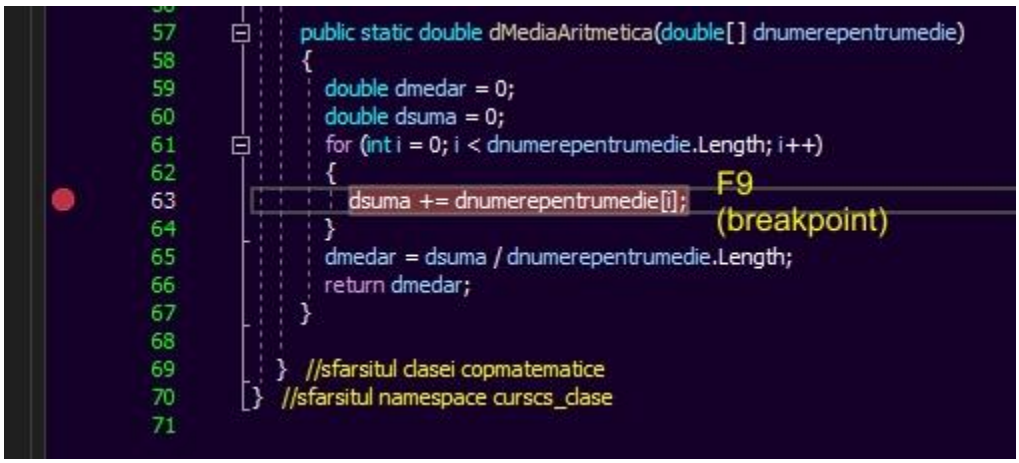
3. plasarea unei instrucţiuni **MessageBox.Show(...** înainte sau după ceea ce ne interesează şi care să afişeze informaţii despre acel lucru
    - programul ***eventual*** se va opri în acel punct
      până când se răspunde la/până când se închide acea MessageBox

Metodele 1 şi 2 nu sunt disponibile în afara Visual Studio
Adică, nu pot fi folosite când programul rulează independent, ca fişier cu extensia .exe sau alta

De un Breakpoint se poate trece...
- ***fie apăsând tasta F5***
    - ***pentru ca programul să continue...***
        - ***până la următorul Breakpoint sau***
        - ***până la "sfârşitul" său***
- ***fie apăsând tasta F10***
    - ***pentru ca programul să treacă de toată linia/instrucţiunea cu Breakpoint***
- ***fie apăsând tasta F11***
    - ***pentru ca programul să treacă la instrucţiunea următoare celei la care Breakpoint-ul a întrerupt rularea programului -*** *instrucţiune aflată de pe aceeaşi linie (din acelaşi grup) cu cea pe care este Breakpoint-ul*

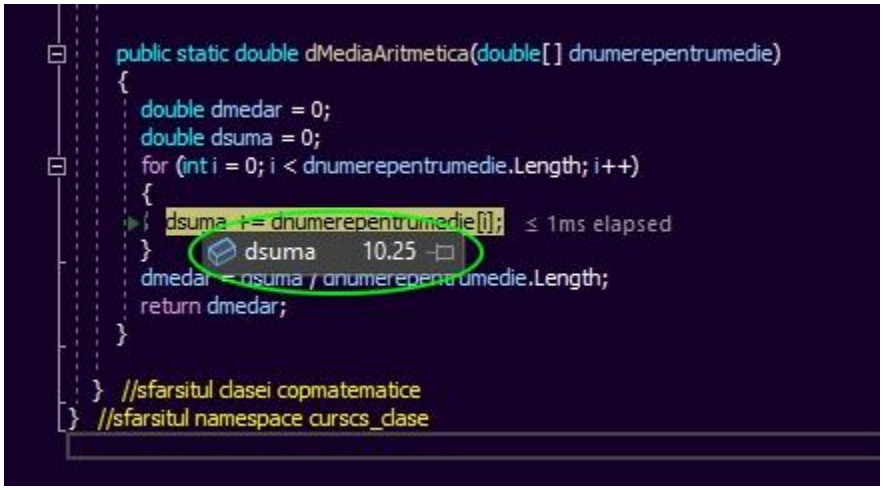
În programul nostru:
vrem să ştim care anume e valoarea sumei numerelor a căror medie aritmetică o solicităm
la fiecare iteraţie prin tabloul care le conţine

1. Inserăm un Breakpoint pe linia cu instrucţiunea unde la suma curentă se adună numărul de la indexul respectiv în tablou: 1) ***plasăm cursorul de text pe acea linie, apoi*** 2) ***apăsăm tasta F9***

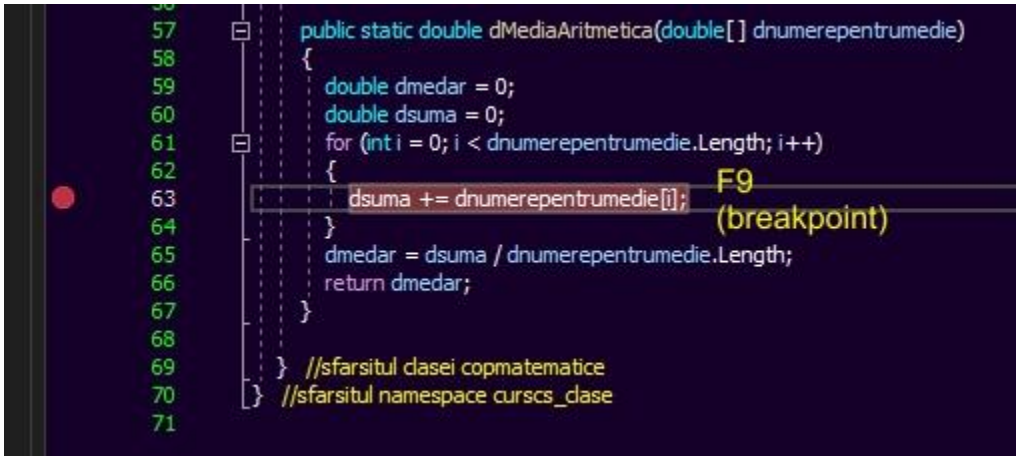


Pornim programul: tasta **F5**:

Visual Studio va opri programul la fiecare iteraţie, va deschide/afişa codul clasei unde se află acel Breakpoint şi aşa vom putea plasa pointerul pe variabile/entităţi de interes – aici, vrem să ştim la cât anume a ajuns suma, astfel că plasăm pointerul pe variabila "dsuma":

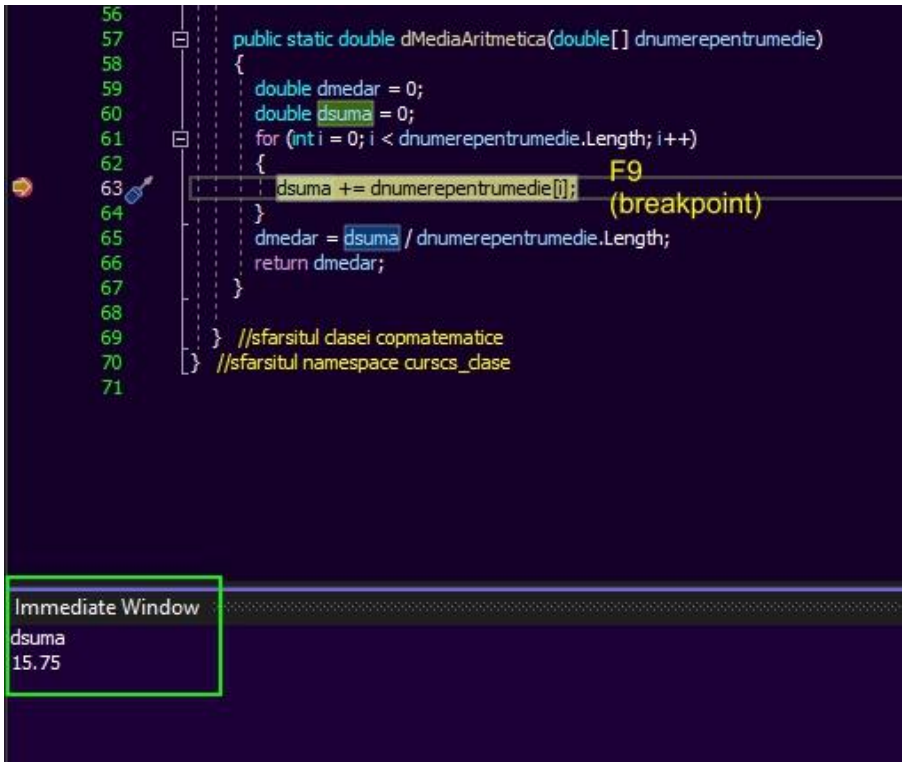


2. Inserăm un Breakpoint pe linia cu instrucţiunea unde la suma curentă se adună numărul de la indexul respectiv în tablou: 1) ***plasăm cursorul de text pe acea linie, apoi*** 2) ***apăsăm tasta F9***



Pornim programul: tasta **F5**:

Visual Studio va opri programul la fiecare iteraţie, va deschide/ne va arăta codul clasei unde se află acel Breakpoint – *chiar dacă în mediul de programare acel cod nu era vizbil/deschis în momentul lansării programului* – şi aşa vom putea scrie în secţiunea "Immediate Window" instrucţiuni cu care să solicităm informaţii – aici, vrem să ştim la cât anume a ajuns suma, astfel că scriem **dsuma** şi apăsăm **Enter**:



3. O instrucţiune **MessageBox.Show(...** poate fi inserată înainte sau după entitatea de interes.

   Această metodă

- este accesibilă în timpul rulării programului atât sub Visual Studio, cât şi independent

- *nu este recomandată pentru verificări frecvente*,
  fiindcă pur şi simplu deranjează utilizatorul forţându-l să răspundă la un MessageBox
  (să o închidă manual, apăsând butonul OK sau altul)
  care prezintă ceva probabil de mai mare interes pentru programator decât pentru utilizator
  şi întrerupând în mod repetat fluxul cursiv al programului

Controlul CheckedListBox

Acest Control este asemănător în funcţionalitate cu *ListBox* –
anume, prezintă o listă de texte care poate fi derulată încât să fie selectate unele din acele elemente .

Deosebirea de *ListBox* şi de fapt funcţionalitatea sa principală este
prezenţa, înaintea fiecărui element, a unei casete care poate fi bifată la fel ca un Control *CheckBox*.

Ca urmare, *CheckedListBox* este util în principal atunci când se intenţionează:

- selectarea anumitor elemente cu indicarea explicită, vizual, a acelor elemente – fiind vizibile casetele bifate
    - încât în program să fie procesate numai acele elemente

Scurtă istorisire de context – Controlul *CheckedListBox* nu a existat în unele versiuni anterioare ale Visual Studio, deşi neîndoios este util.
Unii programatori (independenţi de Microsoft) care aveau nevoie de astfel de Controale relativ complexe au recurs la "tweak-uri" şi "workaround-uri" – subtilităţi ale sistemului de operare Windows care permit accesul la niveluri joase în crearea de către sistemul de operare a Controalelor simple care oferă posibilitatea de interacţiune cu utilizatorul.
Aceasta s-a făcut cu dificultate – apelând, în general, proprietatea "handle-to-Window" ("mânerul" de care poate fi "apucat" un Control, un identificator unic al său în sistemul de operare) – menţionată în cursul de programare în C# – şi unele derivate din aceasta şi asociate ei. Astfel, *înainte* ca o parte dintr-un Control "simplu", deja existent/disponibil, sau acel Control în totalitatea sa (de exemplu, o intrare de meniu colorată la fel ca restul) să fie "desenat/ă" (creată de către sistemul de operare), se accesa o astfel de proprietate de nivel jos a sa şi/sau Evenimente asociate iniţializării/creării acelui Control şi se modificau încât Controlul să obţină proprietăţi noi sau să i se schimbe proprietăţi care implicit erau inaccesibile din mediul de programare.
De exemplu, pentru un Control similar lui *CheckedListBox* procedura generică era aceasta: prin *hWnd* sau alt identificator se obţinea dreptunghiul-conturul unui Control existent şi mai simplu de tip *ListBox* care tocmai urma să fie "desenat", se aflau/calculau dimensiunile sale (în pixeli), înălţimea se împărţea la cea a *font*-ului textelor/elementelor care urmau să fie afişate simultan (care încăpeau integral) în acel dreptunghi, se crea un dreptunghi pentru fiecare astfel de element, înaintea sa se "desena" un dreptunghi căruia i se asociau Evenimente ale unui Control clasic *CheckBox* şi care în acest fel primea rolul de *CheckBox* asociat textului din dreptul său, apoi se puteau face diverse operaţiuni cu această pereche-element aparte (de exemplu, colorare, sau redimensionare încât să aibă altă înălţime decât alte elemente, sau font-ul era colorat sau făcut *italic*), apoi această pereche astfel "desenată" se trimitea în Controlul acela mai simplu, de tip *ListBox*.
Ca atare, în urma unor astfel de strădanii publicate pe Internet de către programatori care creau astfel de Controale, Microsoft a prins ideea şi treptat a creat Controale necesare şi complexe în funcţionalitate şi aspect, oferindu-le "din sursă oficială" programatorilor în Visual Studio şi în alte medii de programare.
Astfel, de la o versiune la alta a Visual Studio numărul Controalelor disponibile pentru crearea unei interfeţe s-a mărit – sau, cel puţin cele existente într-o versiune au căpătat Proprietăţi noi în următoarea versiune.
Platforma .NET a venit cu astfel de Controale complexe – fie inexistente în versiuni de Visual Studio de la 6.0 în jos, fie derivate din unele existente, mai simple – şi fie vizibile pe un Dialog (ca *PropertyGrid* – cunoscut ca secţiunea *Properties* din Visual Studio), fie nu (ca *SerialPort*).

Să vedem ce se poate face – ca operaţiuni de importanţă mare – cu o *CheckedListBox*.

Planul programului este foarte simplu:
Vrem să facem un program unde dintr-o *CheckedListBox* să selectăm (să bifăm) anumite elemente pe care să le prelucrăm cumva.
Vom apăsa un buton şi într-un *TextBox* vor fi scrise informaţii despre elementele selectate.

Pornim Visual Studio, *File -> New... Project...*

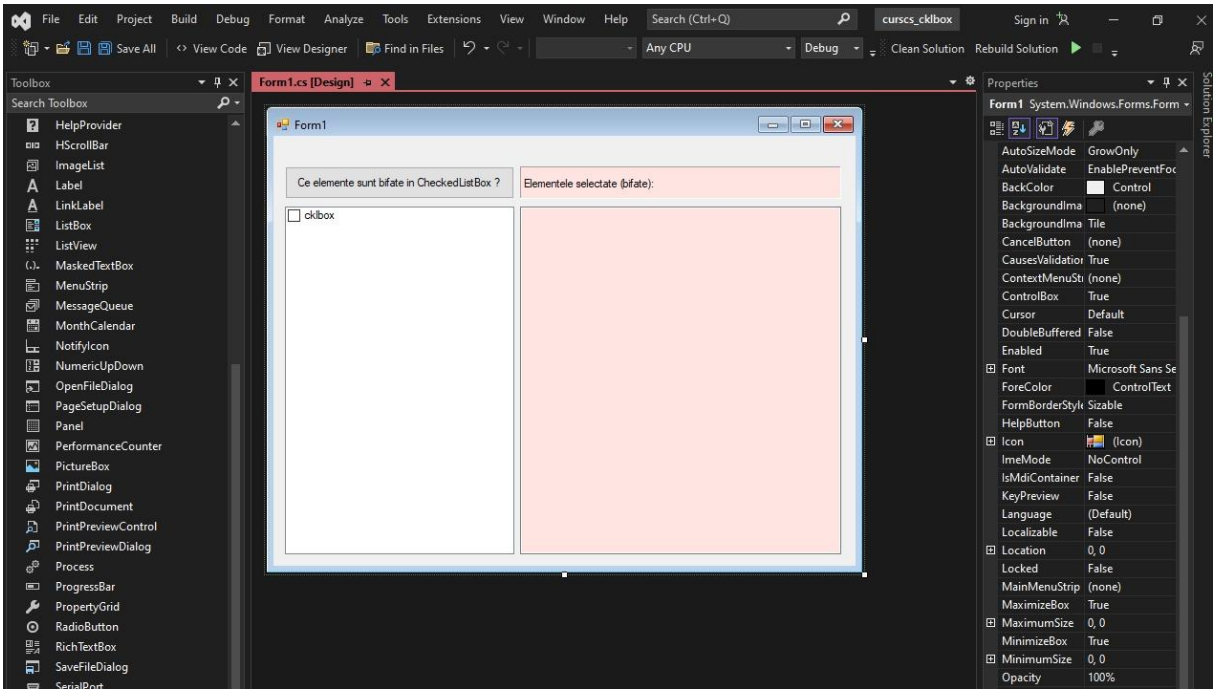
Pe *Form1*, din Toolbox tragem un Control *CheckedListBox*, un *Button*, un *TextBox* deasupra căruia plasăm o etichetă *Label*.

Redenumim
CheckedListBox-ul la "cklbox"
Button-ul la "b"
TextBox-ul la "tb"

Atunci când pe interfaţă există /numai câteva (de exemplu, 3-5/ Controale, se presupune că ştim ce anume face fiecare încât să nu le dăm denumiri (Proprietatea *Name*) descriptive în mod exhaustiv, ci unele scurte ca aici, pur şi simplu pentru scurtarea instrucţiunilor.

Redimensionăm eticheta stabilind Proprietatea sa *AutoSize* la False şi trăgând de colţul său din dreapta-jos, încât să avem o interfaţă decentă.

Interfaţa, *Form1*, ar putea arăta aşa:



Vrem ca, la afişarea *Form1*. în *CheckedListBox* să fie afişate nişte texte.

Aşadar, trebuie să scriem cod pentru Evenimentul *Load* al *Form1* – cel implicit al unei *Form*-e.

Pentru a accesa codul lui *Form1*:
- fie apăsăm tasta **F7**
- fie din secţiunea "Solution Explorer" a Visual Studio – click-dreapta pe "Form1.cs" şi apoi pe intrarea de meniu "View Code".

Ceea ce ne va trimite în modul de scriere a codului, unde:

Declarăm o variabilă de tip tablou de String-uri
în care specificăm textele-elemente de trimis în "cklbox" la Evenimentul *Load* al lui *Form1*:

```
string[ ] sa_statii = {
                        "15420 - Bucuresti-Baneasa",
                        "15421 - Bucuresti-Afumati",
                        "15422 - Bucuresti-Filaret",
                        "15450 - Craiova",
                        "15480 - Constanta",
                        "15120 - Cluj-Napoca" };
```

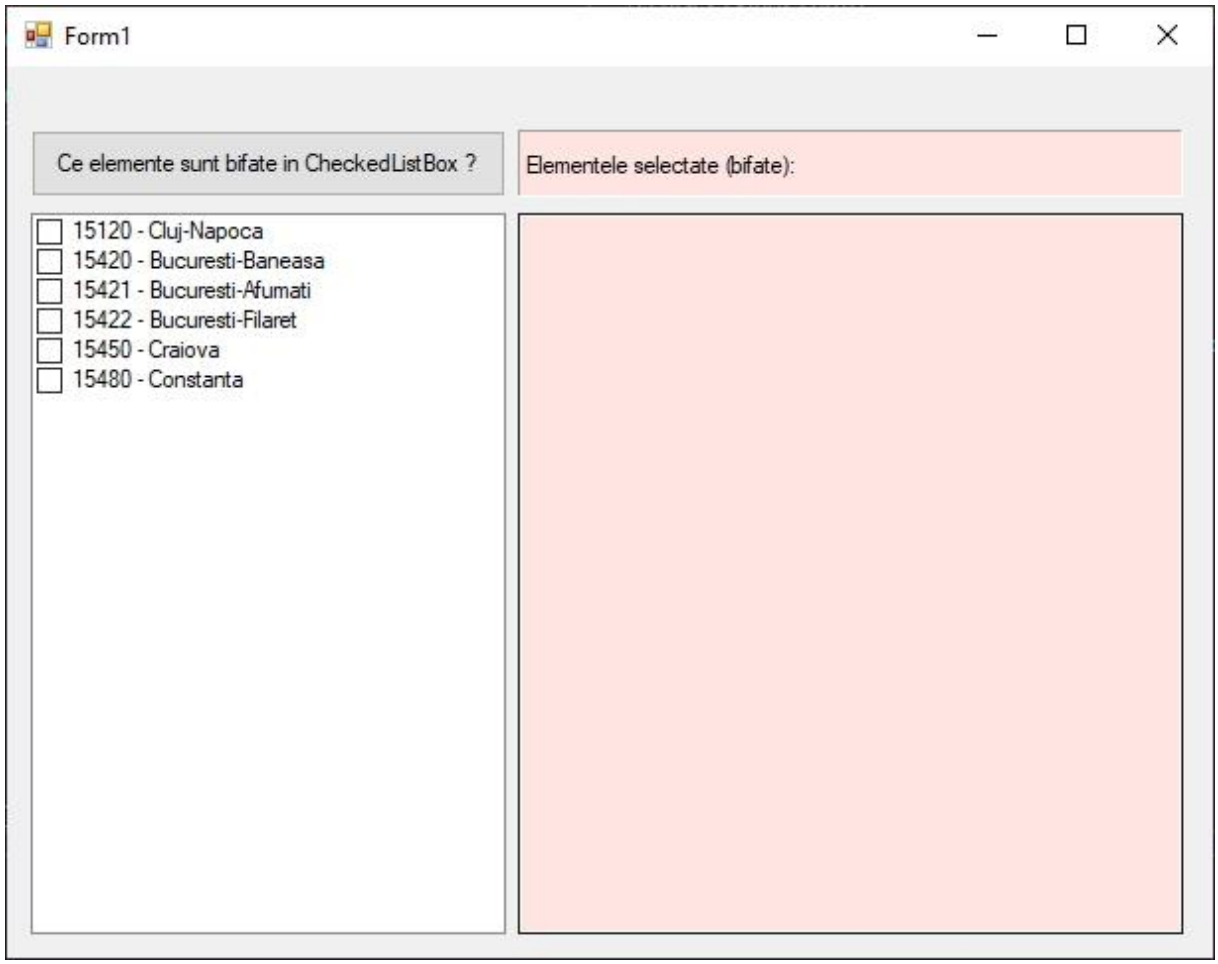
➔ elementele dintr-un tablou pot fi specificate pe oricâte linii, atât timp cât se respectă sintaxa acelei instrucţiuni;
- aici textele sunt fiecare pe o linie numai pentru lizibilitate.

Înapoi în modul de proiectare/Design al *Form1*:
- fie apăsăm simultan tastele **Shift F7**
- fie din secţiunea "Solution Explorer" a Visual Studio – click-dreapta pe "Form1.cs" şi apoi pe intrarea de meniu "View Designer".

Dublu-click pe *Form1* pentru a solicita de la Visual Studio crearea metodei asociate Evenimentului *Load*:
Înlocuim metoda astfel creată cu:

```
private void Form1_Load(object sender, EventArgs e)
{
        //la cklbox adaugam String-urile din tabloul declarat ca disponibil intregului cod al Form1:
        //iteram prin elementele tabloului sa_statii:
        for (int i = 0; i < sa_statii.Length; i++)
        {
                //in cklbox plasam elementul/textul de la indexul curent in iteratie:
                cklbox.Items.Add(sa_statii[i]);
                //
                //sortam alfabetic lista ("alfabetic" = in raport cu particula de [nceput "15xxx")
                cklbox.Sorted = true;
        }
}
```

Pornim programul: apăsăm tasta **F5**: Rezultatul:



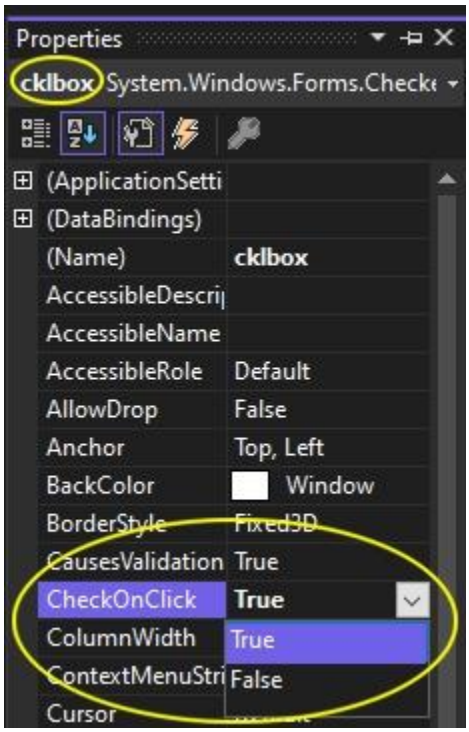
Până acum, totul e cum trebuie.

Să scriem cod pentru afişarea unor informaţii despre elementele bifate din *CheckedListBox*.

Închidem programul şi înapoi în modul de proiectare/Design al *Form1*.

Vom scrie cod pentru Evenimentul *Click* al butonului "b" – prin care vom solicita informaţii despre elementele bifate în "cklbox".

Un Control de tipul *CheckedListBox* are Proprietatea numită "*CheckOnClick*".
Aceasta scuteşte utilizatorul de la a face Click de două ori pe un element, încât elementul să fie bifat/de-bifat la un singur Click.
În secţiunea Properties stabilim această Proprietate a "cklbox" la "True":



Dublu-click pe butonul "b", ceea ce ne trimite la cod unde înlocuim metoda propusă de Visual Studio cu aceasta:

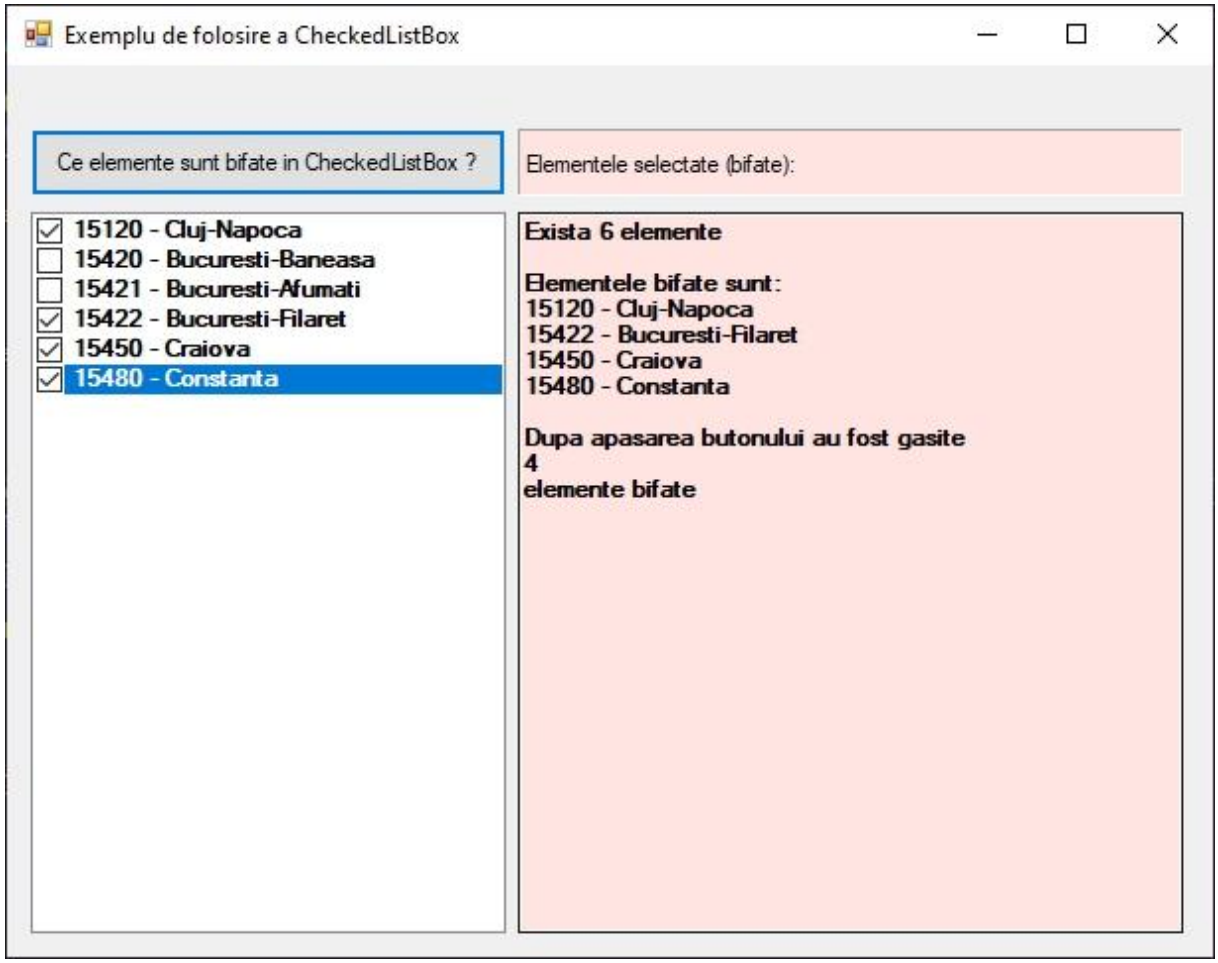
```
private void b_Click(object sender, EventArgs e)
{
        //in TextBox scriem cate elemente se afla in CheckedListBox-ul "cklbox":
        tb.Text = "Exista " + cklbox.Items.Count.ToString() + " elemente";
        tb.Text += "\r\n\r\n";  //doua randuri/linii goale
        // secventa \r\n fiind echivalentul lui Enter
        //
        //verificam daca in "cklbox" exista elemente bifate:
        //
        //daca niciun element nu e bifat:
        if (cklbox.CheckedItems.Count == 0)
        {
                tb.Text += "Nu a fost bifat niciun element.";
        }
        //exista elemente bifate:
        else
        {
                tb.Text += "Elementele bifate sunt:\r\n";
                for (int i = 0; i < cklbox.CheckedItems.Count; i++)
                {
                        tb.Text += cklbox.CheckedItems[i].ToString() + "\r\n";
                        //^ dupa fiecare element scris aici "tastam" Enter
                        //fiecare element pe o linie in TextBox)
                }
                //
                //afisam numarul elementelor bifate:
                tb.Text += "\r\n";
                tb.Text += "Dupa apasarea butonului au fost gasite\r\n" +
                                cklbox.CheckedItems.Count.ToString() + "\r\n"+
                                "elemente bifate";
        }
}
```

Se observă ***diferenţa dintre Proprietăţile "CheckedItems" şi "Items"***
ale unui Control de tipul *CheckedListBox*:

- "***CheckedItems***" reprezintă ***colecţia elementelor bifate***
- "***Items***" reprezintă ***colecţia tuturor elementelor***

Totuşi, acestea fiind colecţii, elementele lor pot fi parcurse/accesate cu iteraţii ca ***for...*** şi ***foreach***

Pornim programul: apăsăm tasta **F5**: Rezultatul:



Vrem să extindem funcţionalitatea programului, aflând suma "indicativelor" elementelor selectate, presupuse a fi staţii meteorologice descrise la fel pentru afişare în "cklbox" – anume, indicativul urmat de semnul minus şi numele staţiei.

Pentru aceasta trebuie să extragem cumva indicativul staţiei, anume primele 5 litere ale fiecărui element din "cklbox"; acesta va fi un String, extras fiind dintr-un String. Pentru separarea unei secvenţe dintr-un String vom folosi metoda **Substring(...** oprindu-ne la prima apariţie a semnului minus identificat cu metoda **IndexOf(...**
Apoi, trebuie să convertim acel String la număr.
Pentru conversie între ***Tipuri*** vom folosi metoda "clasică" **Parse(...**
anume, **int.Parse(...** – adică, încercăm conversia din ***Tipul*** String la ***Tipul*** int (Int32/Int64).
Astfel că modificăm codul de mai sus încât să implementăm această funcţionalitate:

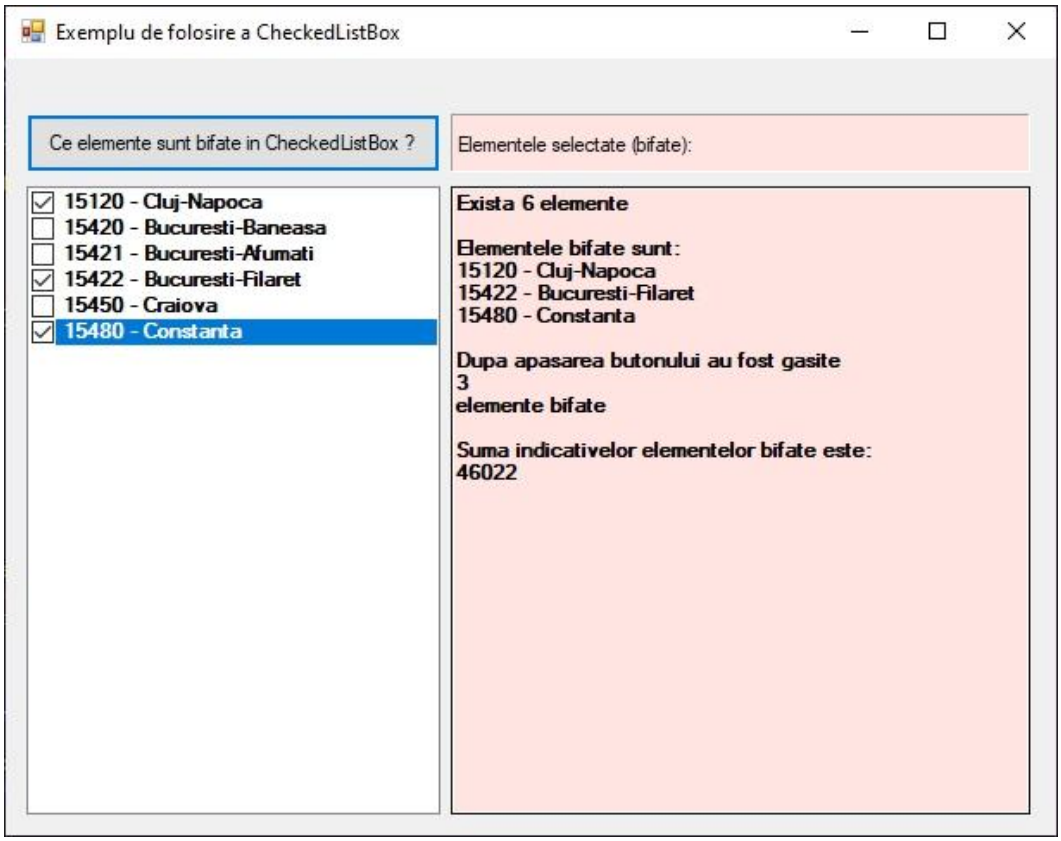
Urmează cod pe două pagini.
Ca de obicei:

- comentariile explică unele instrucţiuni
- liniile libere pot fi şterse după plasarea în cod (prin *Copy-Paste*) a celui de aici:

```
private void b_Click(object sender, EventArgs e)
{
	//in TextBox scriem cate elemente se afla in CheckedListBox-ul "cklbox":
	tb.Text = "Exista " + cklbox.Items.Count.ToString() + " elemente";
	tb.Text += "\r\n\r\n";  //doua randuri/linii goale
	// secventa \r\n fiind echivalentul lui Enter
	//
	//verificam daca in "cklbox" exista elemente bifate:
	//
	//daca niciun element nu e bifat:
	if (cklbox.CheckedItems.Count == 0)
	{
		tb.Text += "Nu a fost bifat niciun element.";
	}
	//exista elemente bifate:
	else
	{
		tb.Text += "Elementele bifate sunt:\r\n";
		for (int i = 0; i < cklbox.CheckedItems.Count; i++)
		{
			tb.Text += cklbox.CheckedItems[i].ToString() + "\r\n";
			//^ dupa fiecare element scris aici "tastam" Enter
			//fiecare element pe o linie in TextBox)
		}
		//
		//afisam numarul elementelor bifate:
		tb.Text += "\r\n";
		tb.Text += "Dupa apasarea butonului au fost gasite\r\n" +
			cklbox.CheckedItems.Count.ToString() + "\r\n"+
			"elemente bifate";
	}
	//
	//adunam indicativele statiilor:
	//vom face operatiuni cu String-uri:
	//indicativul il vom separa cu metodele:
	//Substring
	//IndexOf:
	//
	//apoi se va incerca transformarea/conversia unui astfel de indicativ – String
	//la numar folosind metoda
	//Parse
	//disponibila pentru aproape orice Tip de obiect
```

```
        int isuma = 0;
        //incercam insumarea intr-un bloc try-catch
        //fiindca e posibil ca un element sa nu fie marcat astfel
        //si extragerea indicativului-numar din textul elementului
        //sa nu reuseasca:
        try
        {
            for (int i = 0; i < cklbox.CheckedItems.Count; i++)
            {
                string selementbifat = cklbox.CheckedItems[i].ToString();
                int indicativ =
                    int.Parse(selementbifat.Substring(0, selementbifat.IndexOf("-")));
                isuma += indicativ;
            }
            tb.Text += "\r\n\r\n";
            tb.Text += "Suma indicativelor elementelor bifate este:\r\n" + isuma.ToString();
        }
        catch
        {
        }
}
```

Pornim programul: apăsăm tasta **F5**: Rezultatul:



Duplicarea unor fişiere (Fişiere de rezervă)

Uneori modificăm fişiere, de exemplu documente sau imagini importante
şi apoi descoperim că avem nevoie de versiuni mai vechi ale lor,

- fie pentru a modifica alte porţiuni şi astfel a crea versiuni diferite ale unui fişier
- fie pentru a face aceleaşi schimbări şi astfel a crea mai multe versiuni identice
- fie din cauza deteriorării sau ştergerii versiunilor de fişier modificat
- fie din alte cauze care fac necesară întoarcerea la versiunea iniţială/originală a unui fişier

Aşadar, atunci când lucrăm cu fişiere importante ori cu semnificaţie mai mare decât a altora
este foarte bine să păstrăm versiunea originală a lor.

Aceasta se poate face

- din sistemul de operare însuşi
    - de exemplu, în Windows®™
        - un fişier poate fi duplicat astfel:
            - se lansează programul "*Windows Explorer*" ("*My Computer*") – apăsând simultan tastele "**fereastră**" şi "**E**"
            - se navighează până la folderul/directorul unde se află fişierul respectiv
            - se apasă butonul drept al mouse-ului pe denumirea fişierului pentru afişarea meniului de context
            - se alege comanda "**Copy**" sau similar, cu acelaşi înţeles – de plasare a acelui fişier în memorie-Clipboard
            - se navighează până la folderul-destinaţie pentru acel fişier de rezervă
            - se apasă butonul drept al mouse-ului şi se alege comanda "**Paste**" sau similar, cu acelaşi înţeles – de aducere a fişierului din Clipboard
                - sau în loc de aceasta se apasă simultan tastele **Ctrl** şi "**V**"
- dintr-un program care are această funcţionalitate

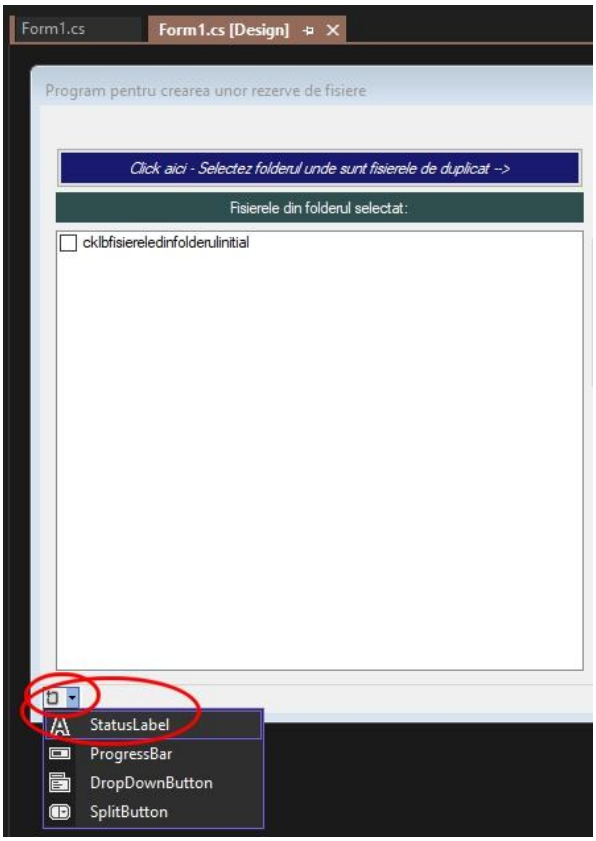
Aici vom crea un astfel de program.

Planul programului este:

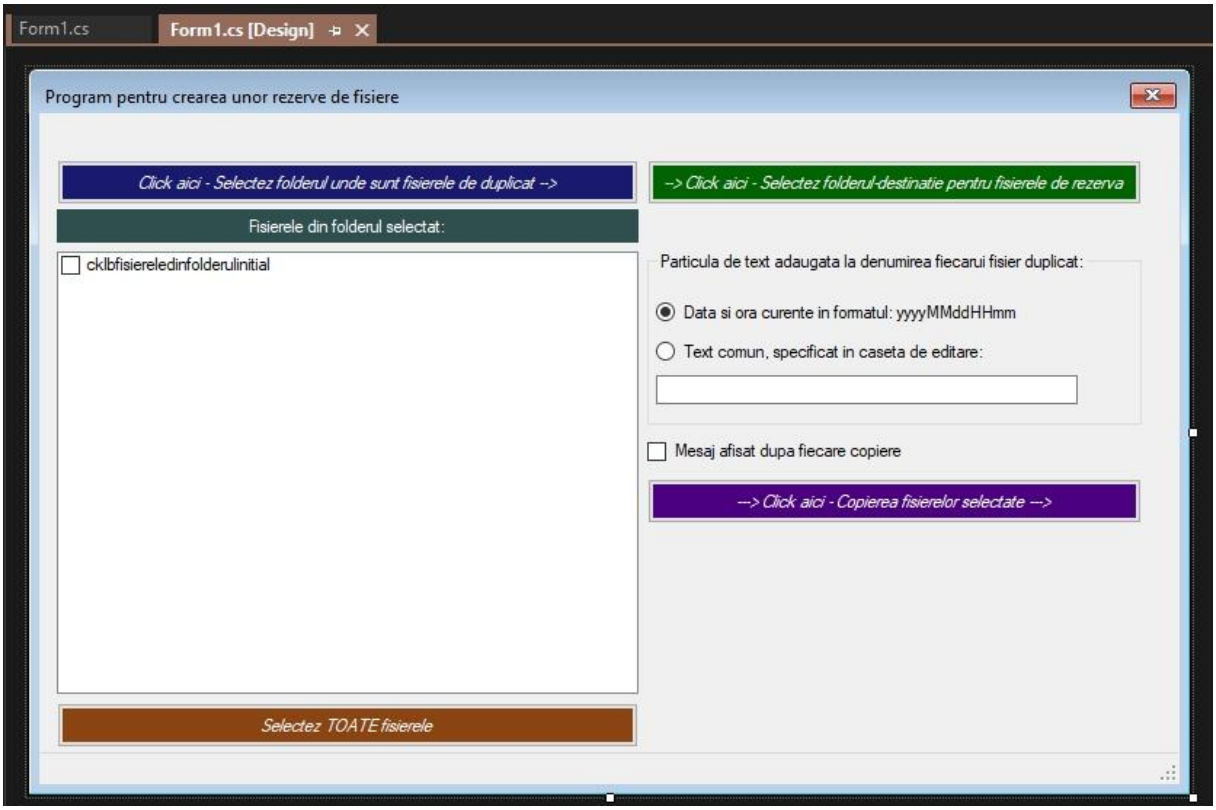
- să proiectăm o interfaţă alcătuită dintr-o singură Form-ă, presupunând că o Form-ă este suficient pentru scopul programului
- vrem să punem deoparte ca rezerve câteva fişiere (sau chiar unul singur) la care să ne întoarcem cel puţin câte o dată pentru a le modifica în alt folder şi în diverse feluri
- vom solicita lista fişierelor din folderul de unde le preluăm
- vom selecta/bifa în acea listă fişierele pe care le vom copia în alt folder
- vom putea redenumi cumva fişierele de rezervă, ajunse la destinaţie,
  pentru a şti, de exemplu, de ce/când/a câta oară le-am creat

Din planul nostru reiese că pe Form-ă trebuie să existe aceste Controale:

- un **FolderBrowserDialog**
  - pentru a specifica folderul/directorul unde se află fişierele iniţiale
- un **CheckedListBox**
  - pentru a o popula cu numele/denumirile fişierelor din acel folder
  - şi pentru a bifa fişierele de copiat în altă parte
- încă un **FolderBrowserDialog**
  - pentru a specifica folderul-destinaţie al fişierelor selectate
    (pentru ambele astfel de alegeri poate fi folosit un singur astfel de Control,
    dar pentru a nu ne încurca şi a nu complica listingul
    creăm două ***FBD***-uri)
- câteva **RadioButton**
  - ca opţiuni pentru redenumirea-marcarea fişierelor ajunse la destinaţie, de exemplu
    - cu data curentă
    - cu un text comun, de exemplu "**_**imagineoriginala**_**"
- trei **Button**
  - unul, pentru a selecta folderul unde se află fişierele iniţiale
  - unul, pentru a selecta folderul unde se vor plasa fişierele selectate
  - unul, pentru a iniţia copierea fişierelor selectate
- un **CheckBox** pentru a indica afişarea sau nu a unui mesaj după copierea fiecărui fişier
- un **StatusStrip**
  - în care plasăm o etichetă ***StatusLabel***
    - unde va fi afişat numărul fişierelor din folderul iniţial:



Interfaţa ar putea arăta astfel:



Întrucât vom lucra cu fişiere,

- în secţiunea **using** vom face o referire la spaţiul de nume **System.IO**
  using System.IO;
- vom încadra instrucţiunile în blocuri **try – catch** (**- finally**)

Programul care ilustrează aceste proceduri poate fi modificat sau completat după preferinţe/nevoi.

De asemenea, înainte de modificare pot fi puse într-o locaţie de rezervă
fişierele din acest proiect
denumite
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs

care la nevoie pot fi preluate (**Ctrl C**) din acea locaţie de rezervă şi suprascrise (**Ctrl V**)
peste cele din folderul acestui proiect
"curscs_fisierezerve"

Biblioteca – bază de date SQL Server- separarea unor înregistrări: LIKE

Partea introductivă la acest exemplu se află în
- documentul-tutorial despre lucrul cu baze de date.
- secţiunea anterioară acesteia – "***Biblioteca – bază de date SQL Server***"

Vom face un program cu care:
- se vor putea adăuga înregistrări într-o bază de date
- vom separa din toate înregistrările numai pe cele care în valoarea unui câmp au o anumită particulă de text (fragment de cuvânt sau cuvânt întreg)

Întrucât programul va avea această funcţionalitate "dublă"
vom folosi Controlul **TabControl** din Visual Studio

Acest Control este alcătuit (după proiectare de către programator) din mai multe "pagini",
fiecare "pagină"
- fiind dedicată unui scop distinct
- şi aşadar conţinând acele Controale cu care se realizează funcţionalitatea respectivă

Vom crea un TabControl pe care vom pune 2 două pagini:
- una pentru funcţionalitatea de adăugare/inserare a unor înregistrări în baza de date
- una pentru separarea/extragerea numai anumitor înregistrări, cum s-a menţionat

Codul programului este comentat, încât la final să fie înţelese noţiunile:
- funcţionalitatea **TabControl**
- conexiune la baza de date
- cuvântul-cheie **LIKE** care se foloseşte pentru a separa înregistrări asemănătoare (ca valori ale unor câmpuri)

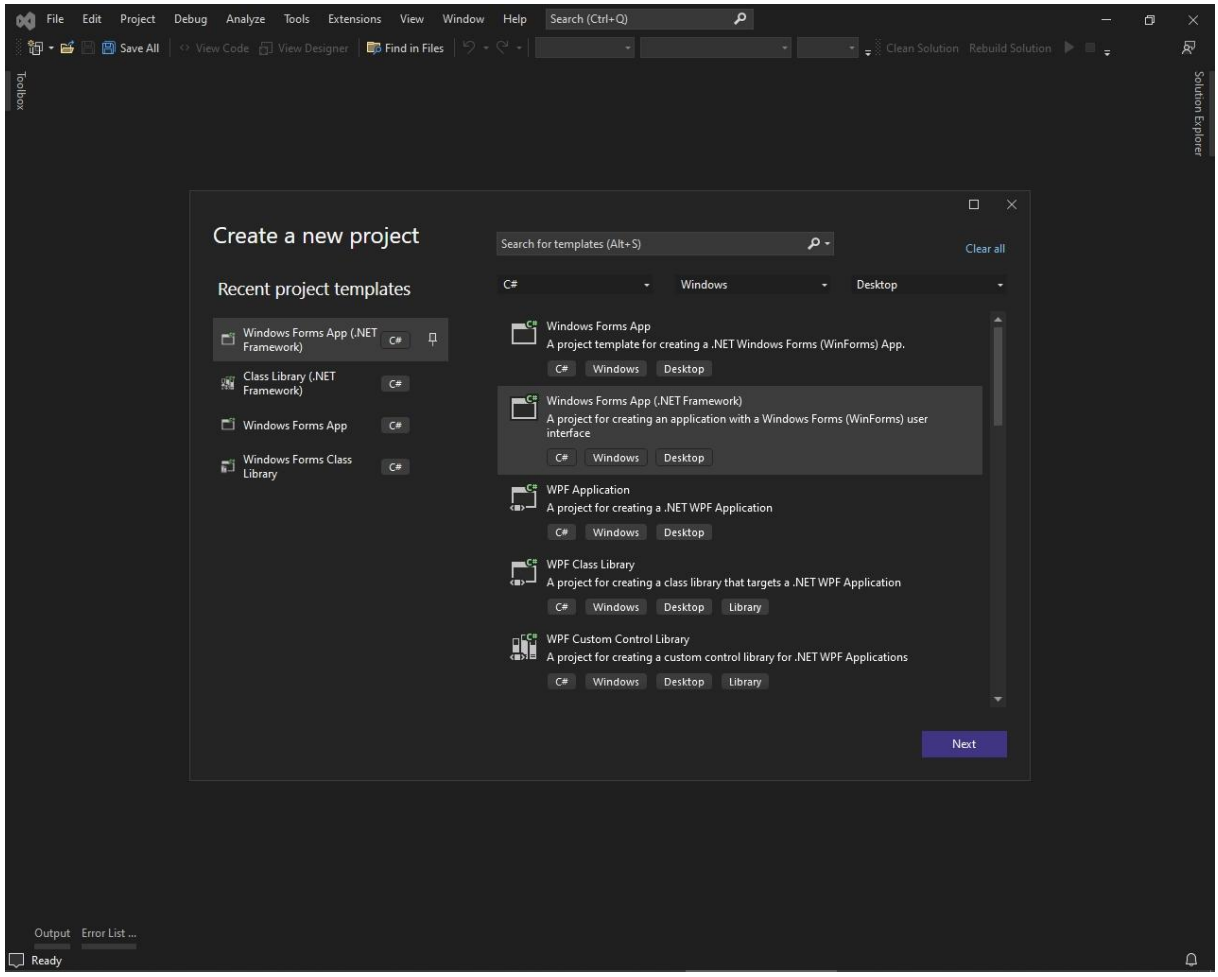
În cod vom folosi variabilele declarate în secţiunea menţionată de aici, anume:

String-ul de conectare la baza de date cu biblioteca:
```
sqlcon.ConnectionString =
                        "server=localhost;initial catalog=bdbiblioteca;user=sa;password=sa123";
```

Obiectul-comandă SQL care va folosi această conexiune:
```
SqlCommand sqlcomm = new SqlCommand();
sqlcomm.Connection = sqlcon;
```

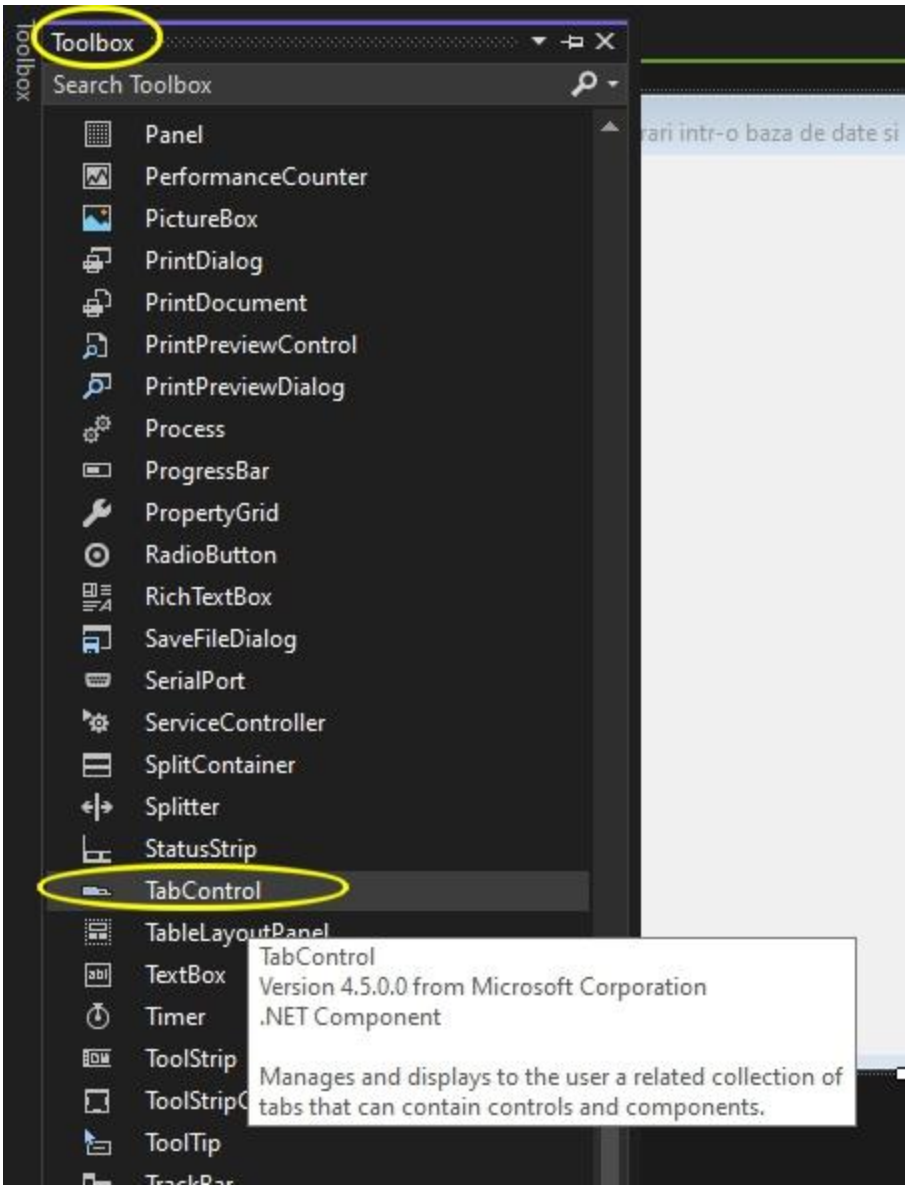
Instrucţiunea SQL cu care specificăm ce bază de date va fi accesată:
```
string stextulcomenziideadaugare_carte = "USE bdbiblioteca;";
```

Pornim Visual Studio.
Ca în exemplele precedente, creăm o soluție/un prioiect pentru aplicație Windows Forms (unde vor fi prezente/vehiculate Form-e):



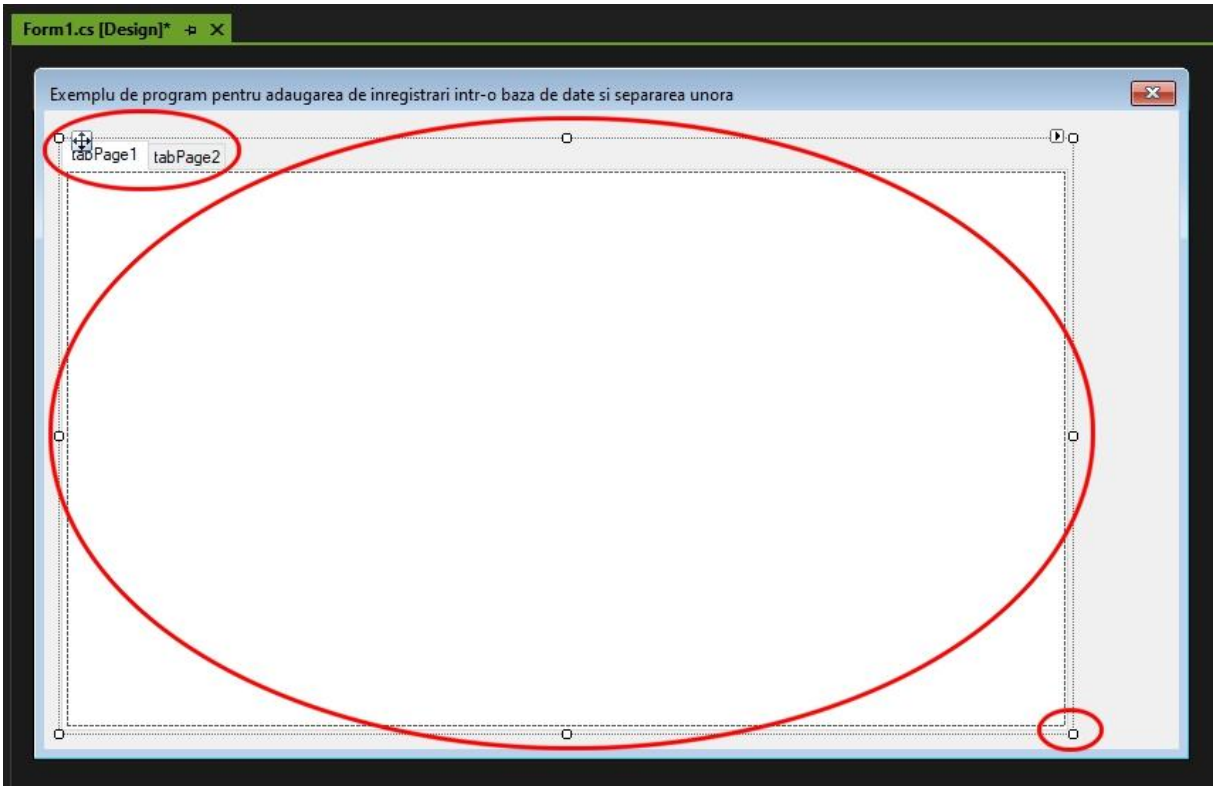
Dăm un nume sugestiv aplicației,
selectăm platforma .NET în versiunea 4.5 (pentru exemplul acesta)
și apoi proiectăm interfața *Form1*

Începem prin a plasa pe *Form1* un **TabControl**:

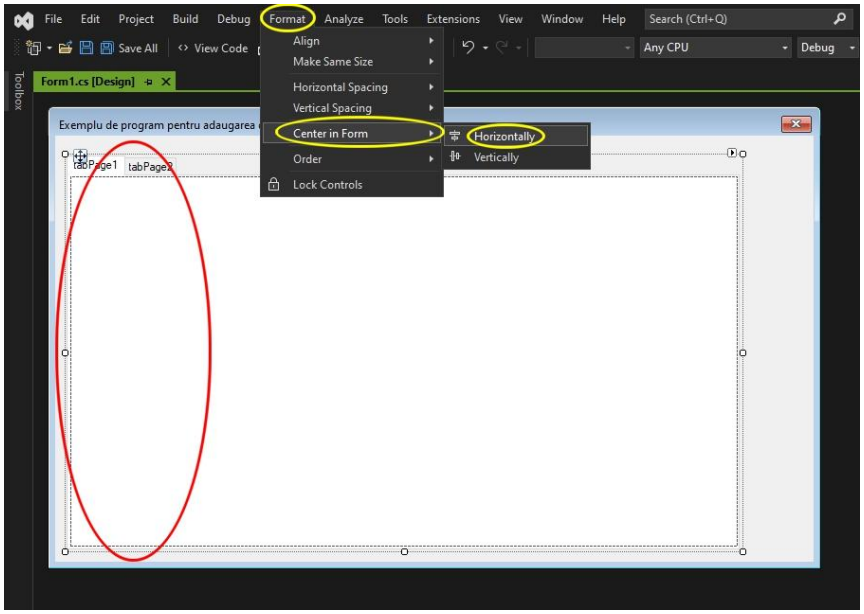


Redimensionăm acest Control, trăgând cu mouse-ul de un colţ al său
(se apasă pe un colţ butonul stâng al mouse-ului fără a fi eliberat, apoi se "trage" de acel colţ)

În imaginea următoare sunt marcate:
- colţul din dreapta-jos de care s-a tras pentru a mări **TabControl**-ul
- două pagini ("*tabPage1*", "*tabPage2*") create implicit la crearea **TabControl**-ului

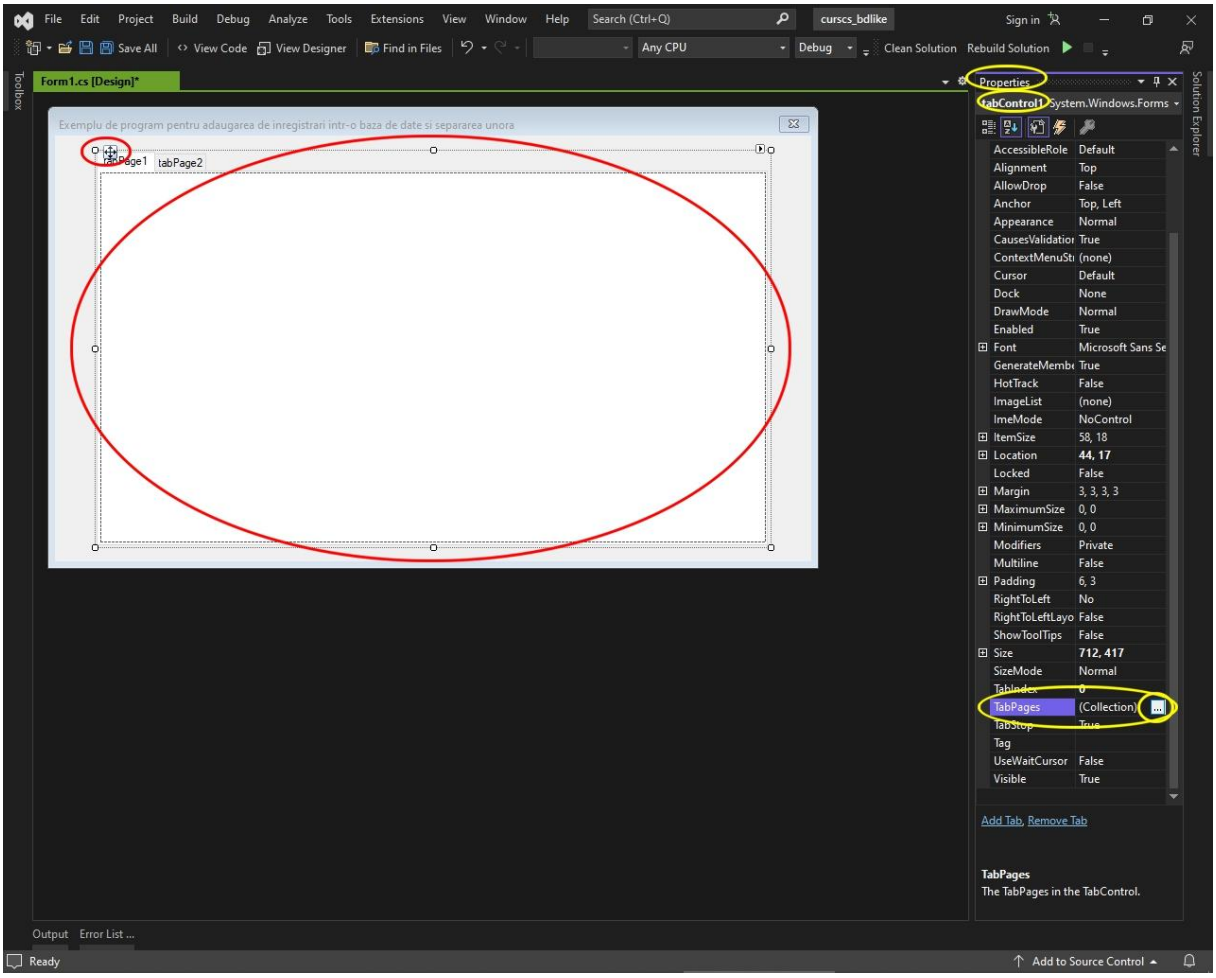


Repoziţionăm **TabControl**-ul în centrul Form-ei:
Pentru aceasta folosim meniul "*Format*"
de unde alegem intrarea "*Center in Form*"
şi apoi opţiunea "*Center Horizontally*"
desigur că apoi putem selecta şi opţiunea de centrare pe verticală, pentru a-l centra complet pe Form-ă:



Schimbăm titlul şi alte caracteristici ale celor două pagini din secţiunea *Properties* a **TabControl**-ului:

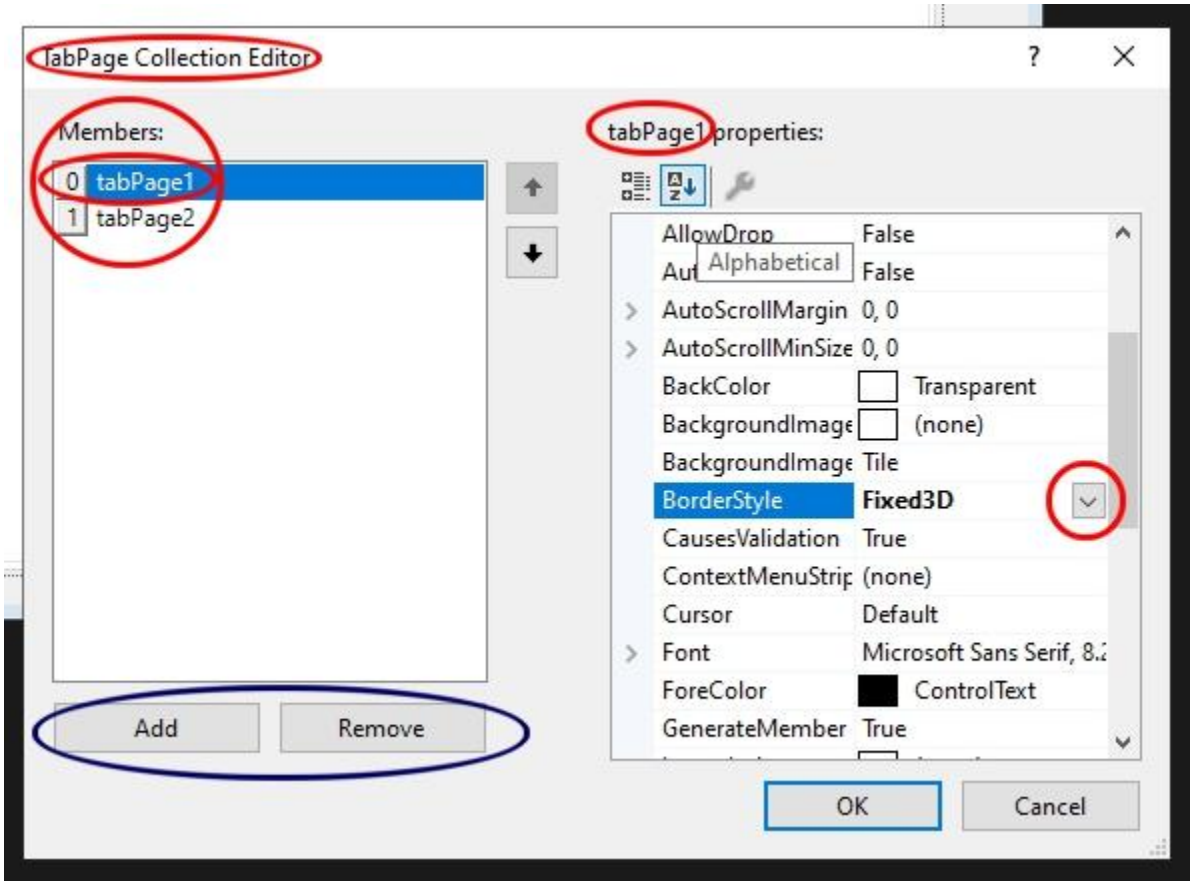
ATENŢIE
Putem selecta (click-stânga)
- atât TabControl-ul cu totul
- cât şi pagini individuale



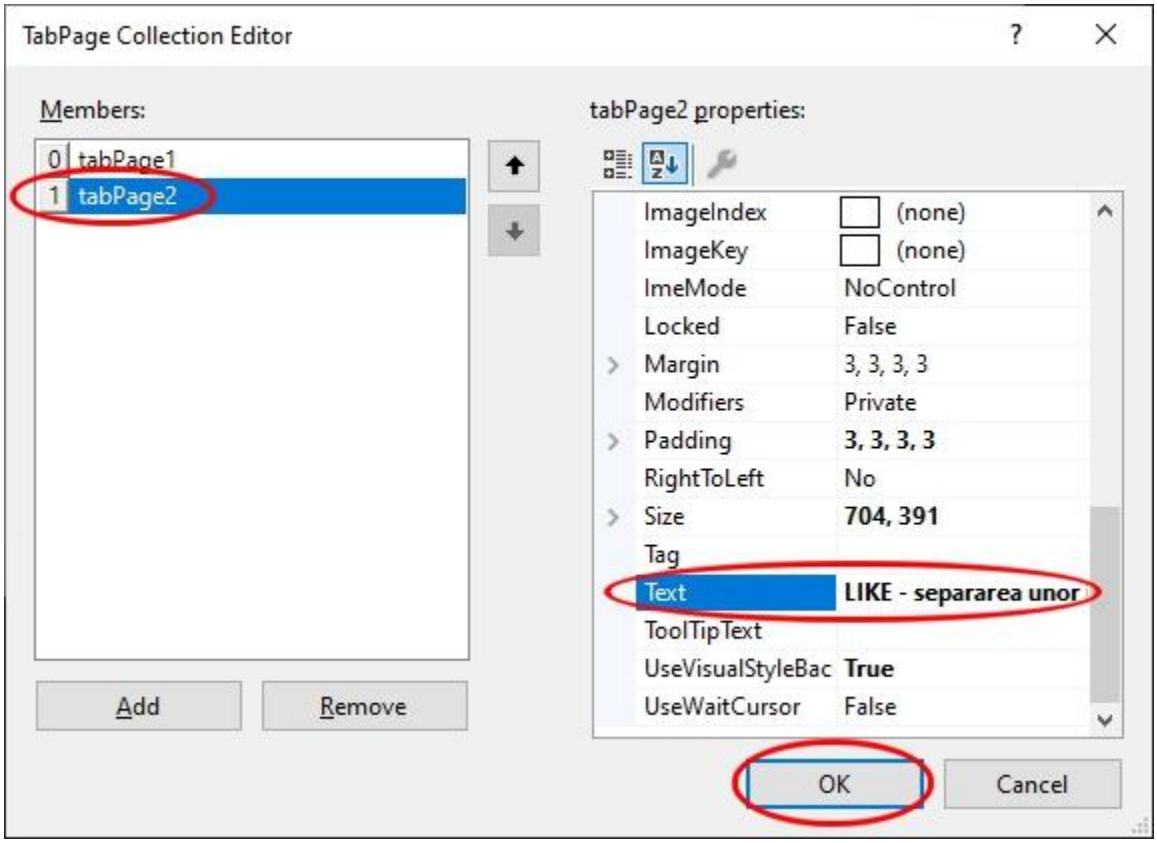
În imaginea următoare se remarcă:

- butonul cu care se adaugă o pagină la **TabControl**-ul selectat pe Form-ă
- butonul "*Remove*" cu care se şterge pagina selectată în listă

Schimbăm tipul marginii (*BorderStyle*) celor două pagini:



Schimbăm *textul (titlul)* celor două pagini:



De fiecare dată când facem un program cu care vor fi accesate baze de date
este bine să vizualizăm conţinutul tabelelor de interes –
cel puţin despre câmpuri:

- denumirea
- tipul
- lungimea (pentru câmpuri de tip text, de exemplu nvarchar)

şi chiar (o parte din) conţinutul unui tabel,

- ca înregistrări
- sau numărul acestora (rareori)

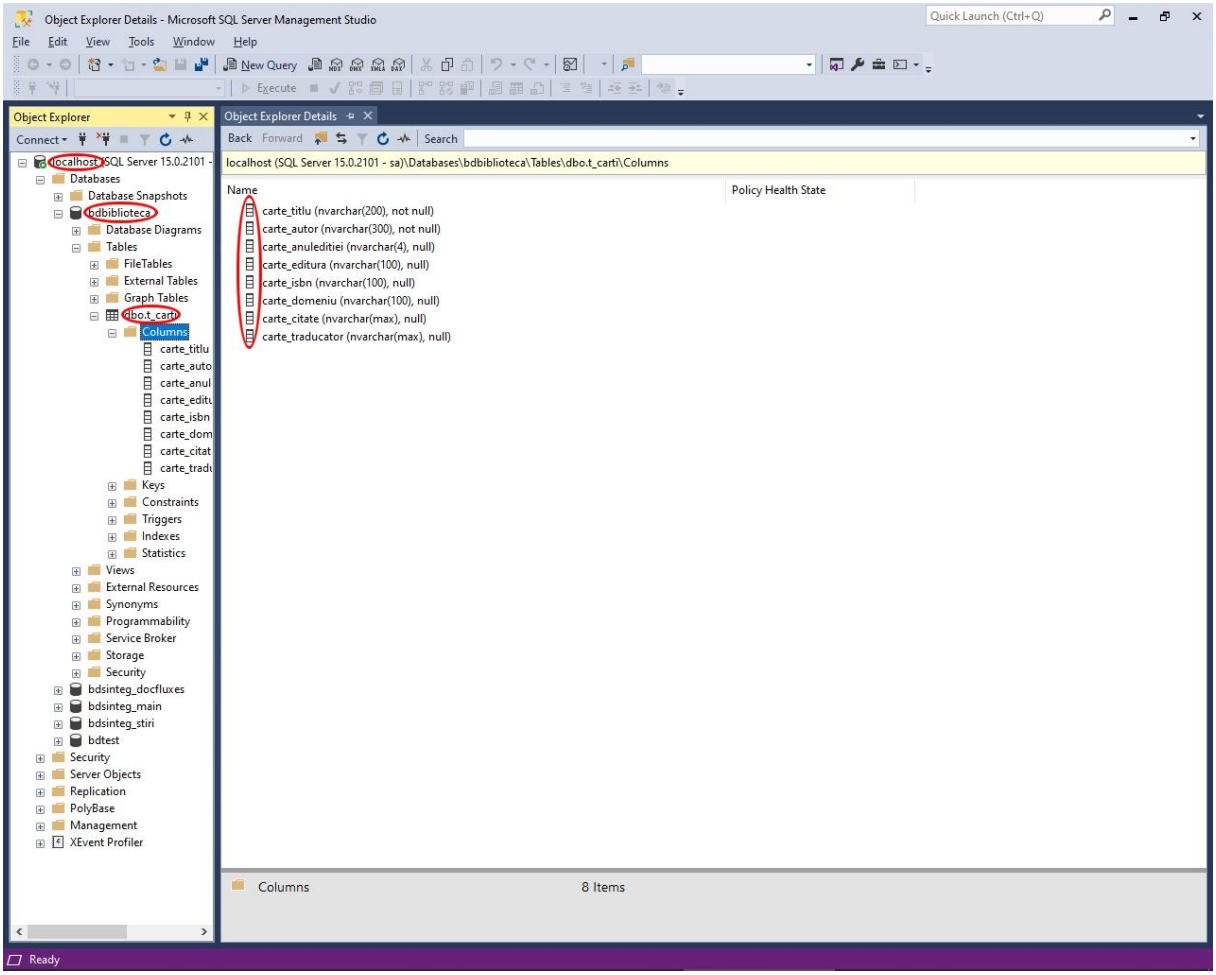
eventual cu SQL Server Management Studio (*SSMS*)

Aceasta trebuie să se facă fiindcă operaţiunile cu o bază de date, fie şi simple pe o bază de date simplă, sunt importante fiindcă presupun accesarea unor informaţii structurate urmând reguli stricte.

De exemplu, pentru a face o medie aritmetică trebuie să ştim

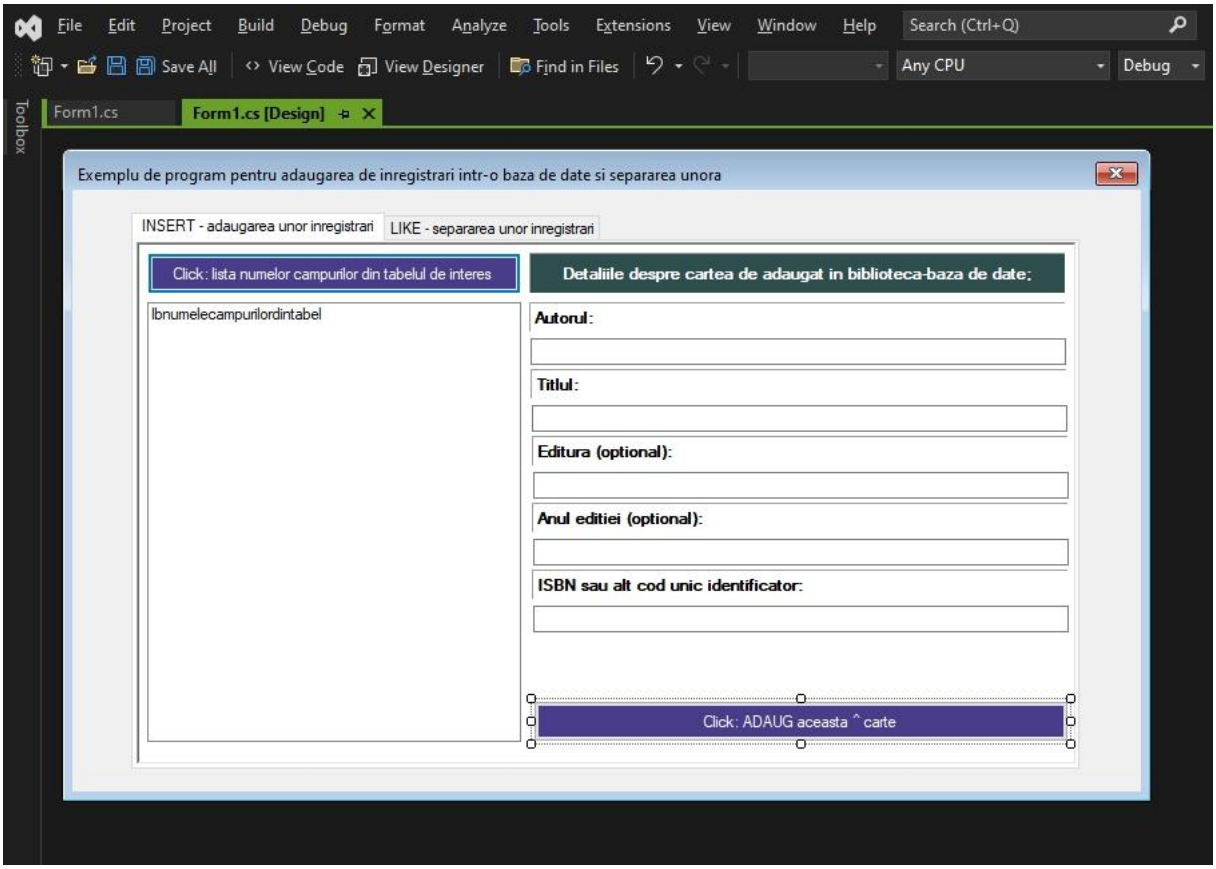
- în primul rând dacă în acel tabel există câmpuri ale căror valori se pretează la operaţii matematice
- ce anume câmpuri sunt de tip numeric
- sau care anume pot fi convertite la un tip numeric
- câte valori sunt NULL-e

În imaginea următoare se vede cum cu *SSMS* s-a accesat baza de date cu care vom lucra
în acest exemplu – anume,
"bdbiblioteca"
unde se află informaţii despre cărţile din "biblioteca virtuală":



Pe pagina "*tabPage1*" vom adăuga cărţi în bibliotecă (în fapt, adăugând informaţii despre fiecare carte).

Pe pagina "*tabPage1*" plasăm-tragem Controale din Toolbox astfel încât
pagina să arate ca în imaginea următoare,
sau similar cu aceeaşi funcţionalitate sugerată de imagine:

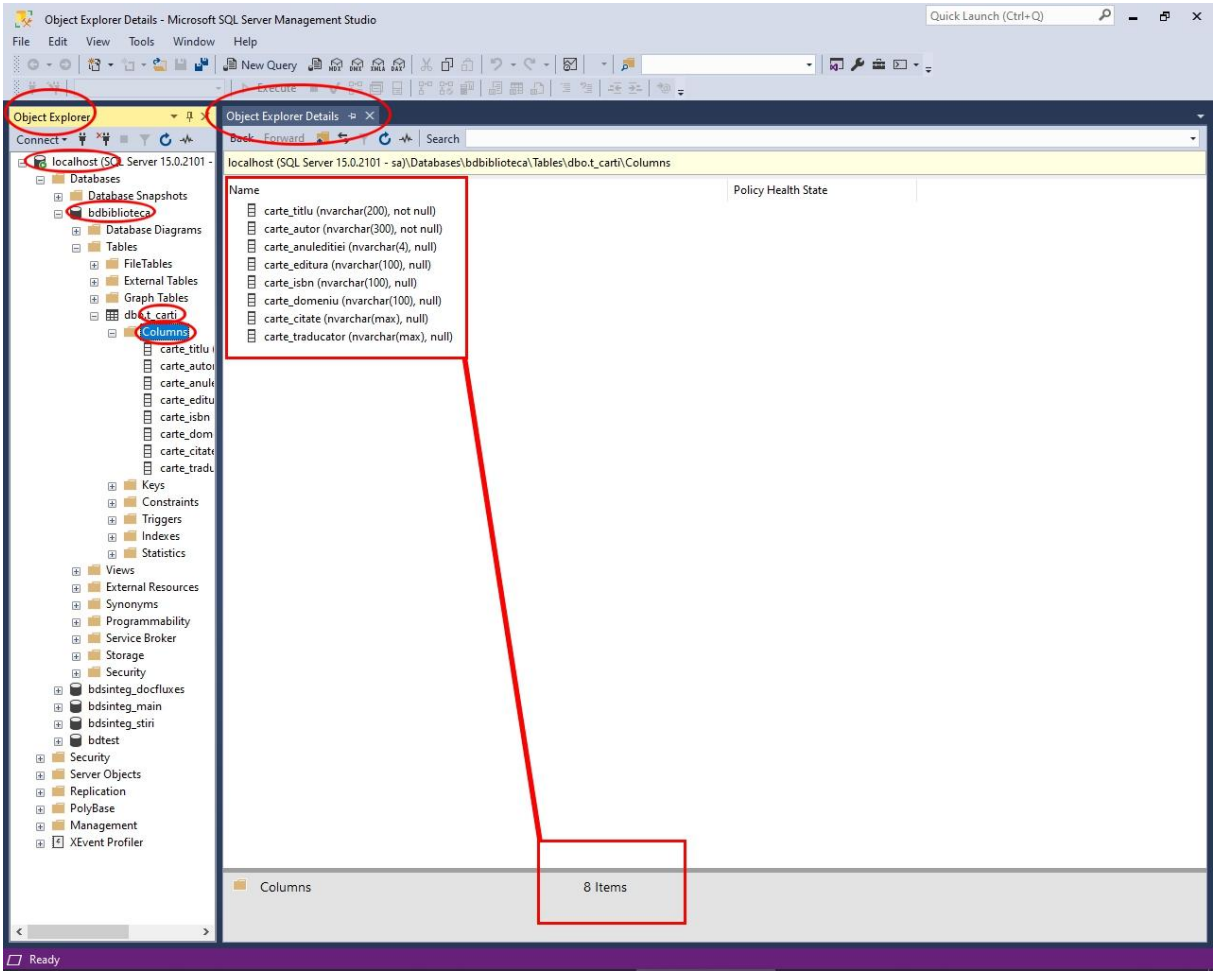


Să scriem codul pentru a adăuga informaţii în baza de date-bibliotecă.

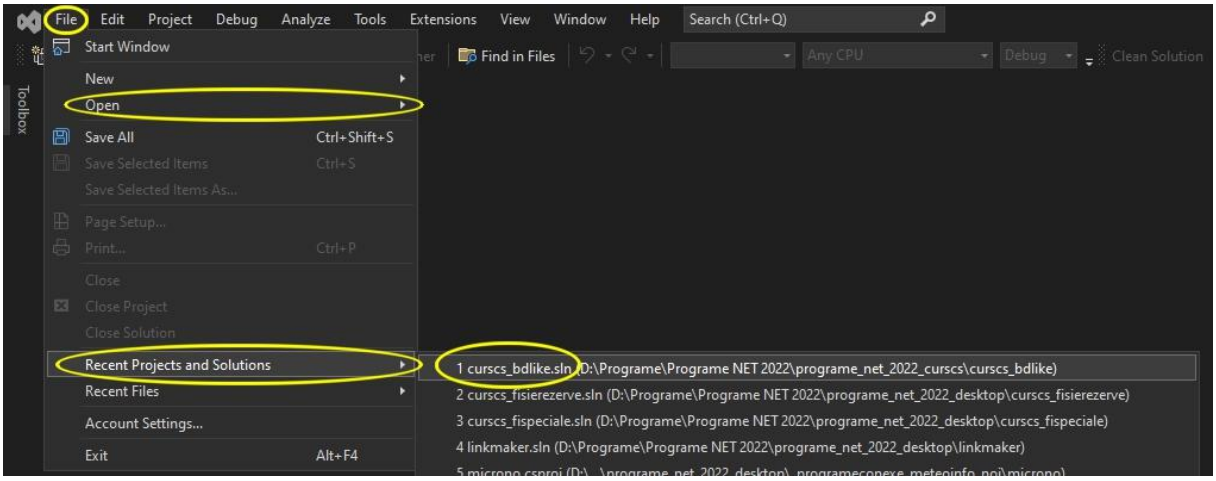
Fiindcă vrem să adăugăm înregistrări, vom folosi comanda INSERT.

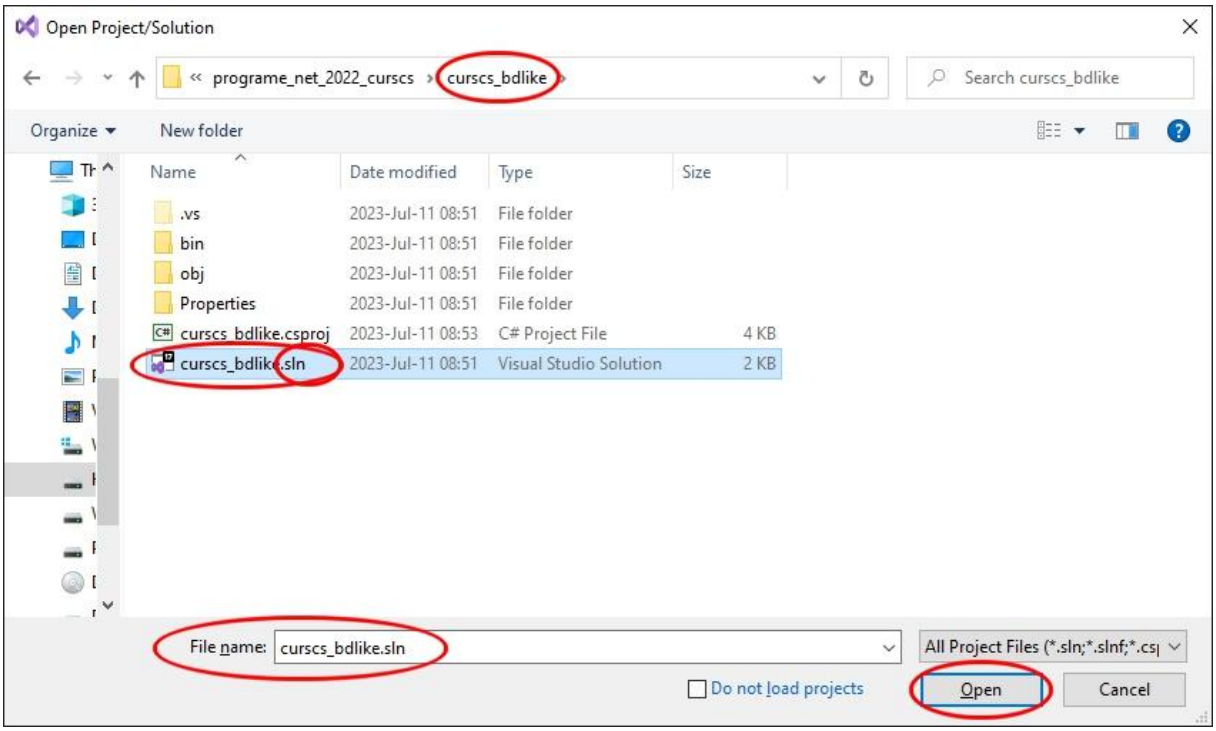
Pentru a şti pe tot timpul scrierii codului care anume este structura tabelului respectiv:

- pornim SQL Server Management Studio (SSMS)
- desfăşurăm ramura "Tables" a bazei de date "bdbiblioteca"
    - pentru a vedea câmpurile acestuia
    - adică structura unei înregistrări
        - încât să scriem corect comanda INSERT



Pornim Visual Studio
Deschidem proiectul-soluţia programului acestuia pentru baze de date
(dacă am folosit deja acest program:
meniul File -> Recent Projects and Solutions ->
iar dacă îl deschidem prima oară:
meniul File -> Open -> Project/Solution... şi se alege fişierul .sln al acestui program)_





Butonul care iniţiază adăugarea de înregistrări
- se află pe pagina "tabPage1", ca în imaginea de mai sus care prezintă interfaţa
- a fost redenumit la "binsert"
- are Text-ul: "Click: ADAUG aceasta ^ carte"

Să scriem codul pentru Evenimentul *Click* al acestui buton:

Dublu-click pe "binsert" pentru a apela generarea automată de catre Visual Studio a metodei asociate Evenimentului Click la care răspunde acest buton:

În metoda generată automat scriem:

```
SqlConnection sconn = new SqlConnection();
sconn.ConnectionString =
        "server=localhost;user=sa;password=sa123;initial catalog=bdbiblioteca;";
```

Înainte de a lucra cu o bază de date iniţializăm/declarăm
- ***conexiunea la acea bază de date, ca obiect SqlConnection***
- ***şirul/Stringul/textul de conectare propriu-zis care va deschide această conexiune***

Apoi iniţializăm/declarăm obiectul-comandă SQL propriu-zisă
care va fi trimisă serverului pentru executare

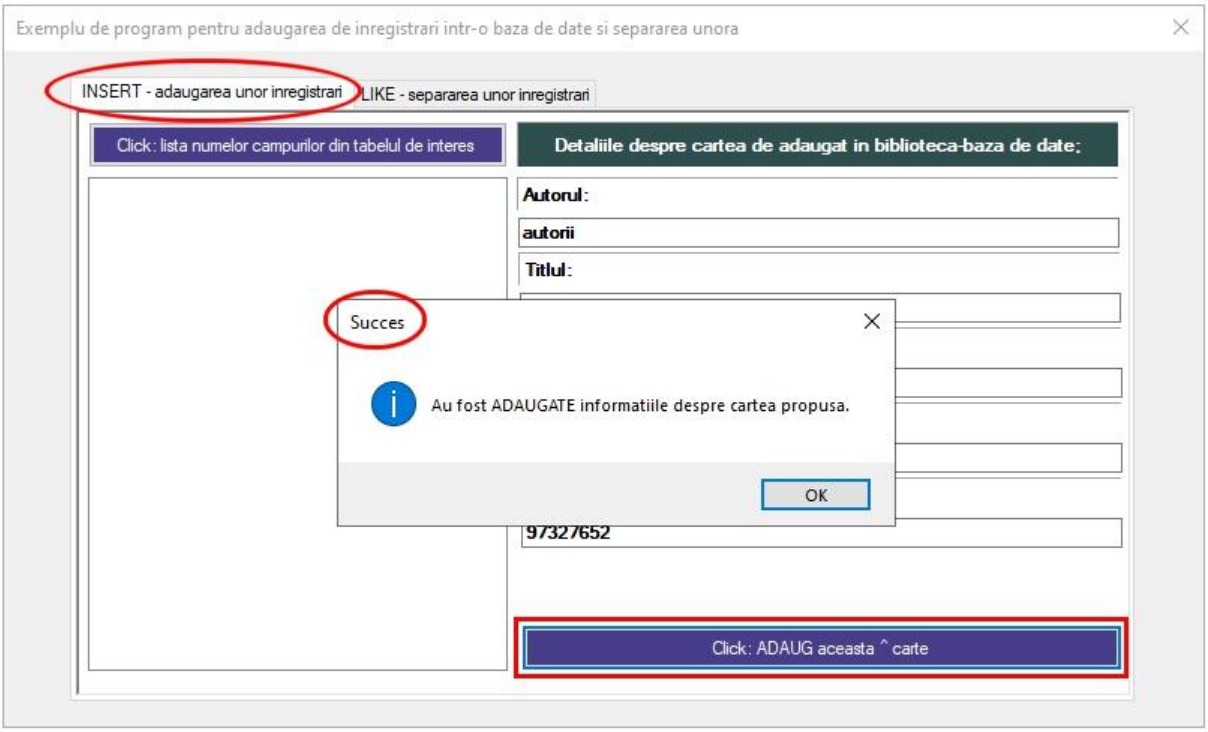
```
SqlCommand scom = new SqlCommand();
scom.Connection = sconn;
```

Înainte de a trimite către server o comandă SQL specificăm
- ***conexiunea pe care va fi trimisă***
- ***textul propriu-zis al comenzii***

Acum scriem codul pentru adăugarea unei înregistrări:
- plasăm instrucţiunile într-un bloc *try – catch – finally*
  - în secţiunea *finally* vom distruge obiectele *SqlConnection* şi Sql*Command*
    indiferent dacă au fost folosite
    fiindcă le-am iniţializat, le-am declarat, le-am atribuit valori

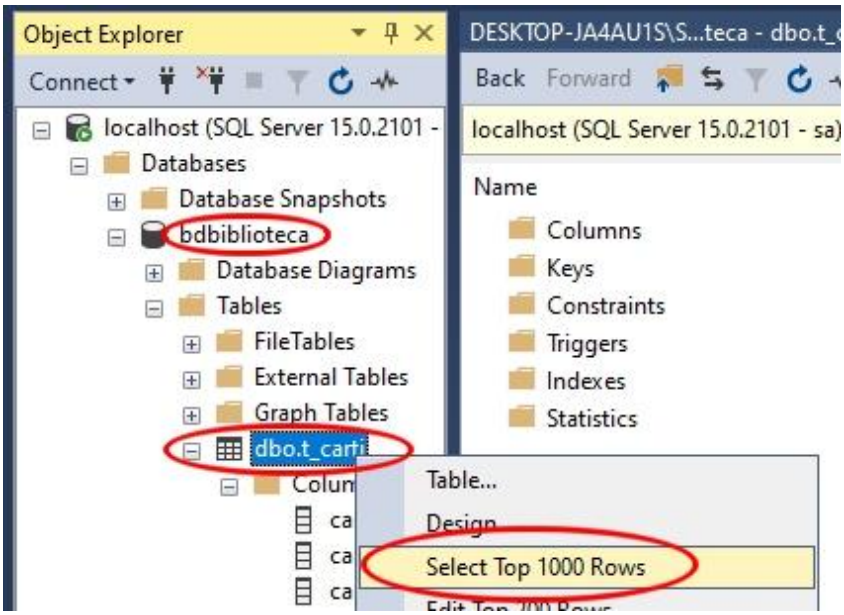
Codul se află în proiectul "curscs_bdlike" din arhiva .zip cu toate proiectele-programele prezentate aici.

Am adăugat o carte:

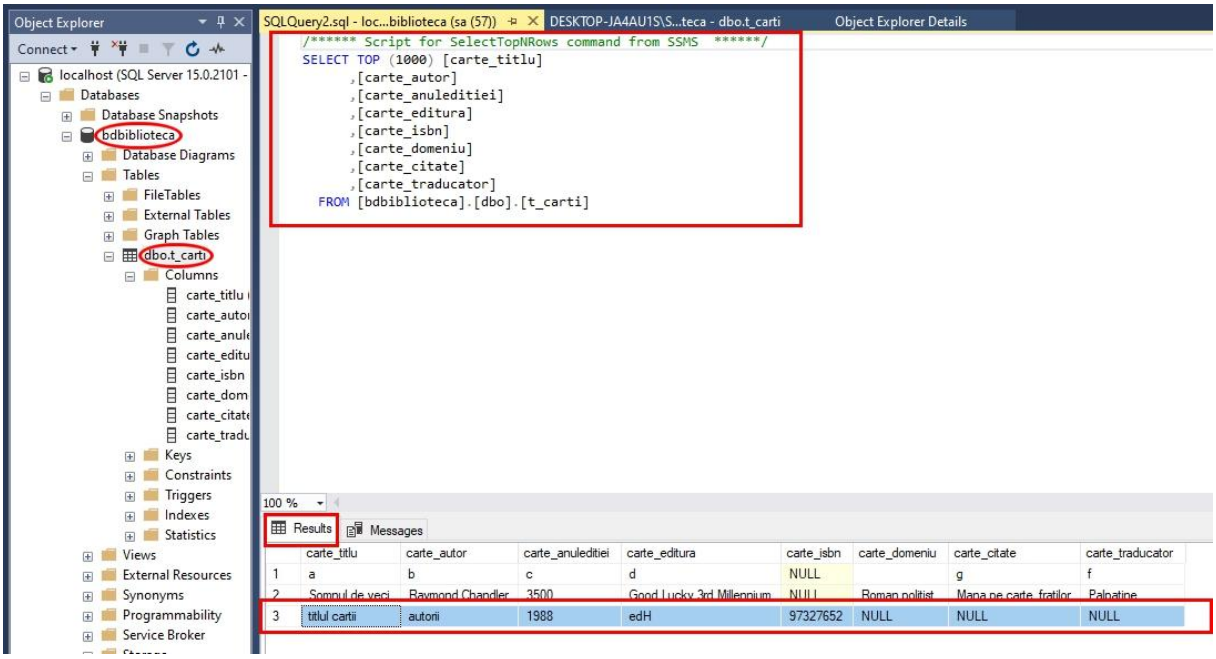


Vrem să verificăm prezenţa informaţiilor adăugate ca înregistrare:

- mergem înapoi în SSMS
- click-dreapta pe numele tabelului "t_carti"
- alegem comanda "Select Top 1000 Rows"



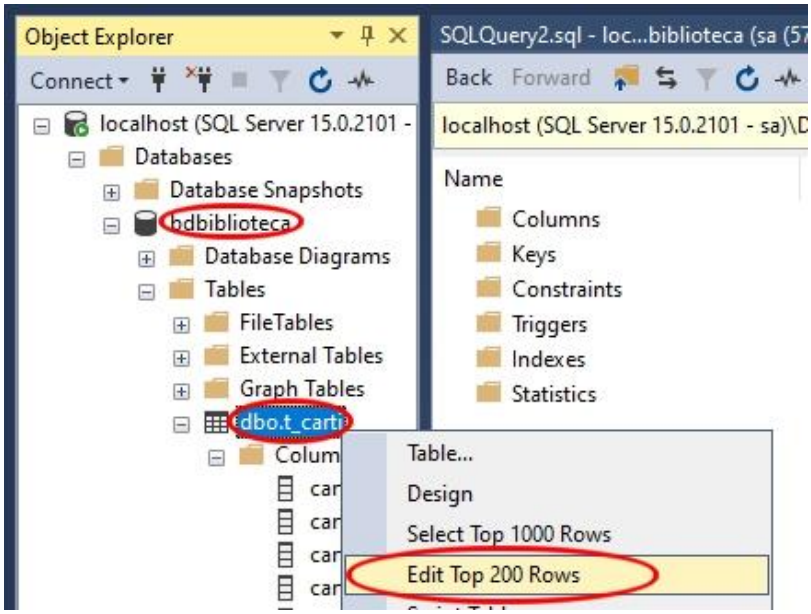
Într-adevăr, vedem că înregistrarea a fost adăugată la cele din tabelul "t_carti":



Dacă vrem să ştergem înregistrări
fără a folosi comanda DELETE:

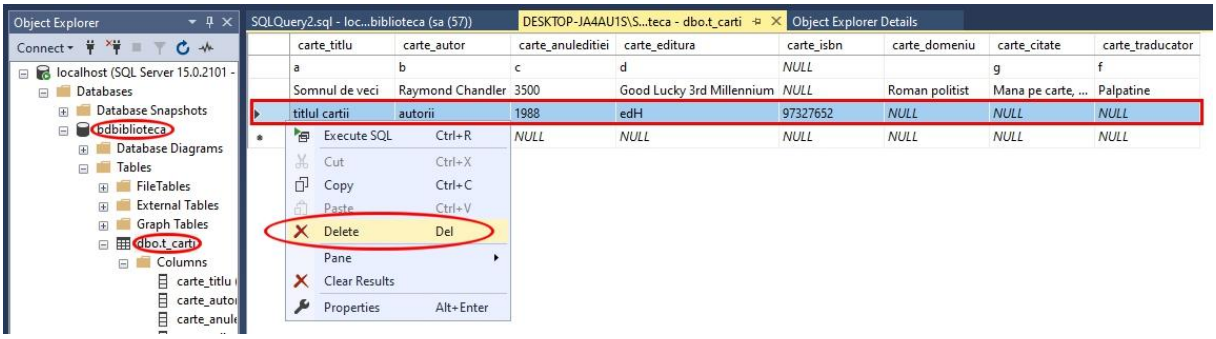
În SSMS:

- click-dreapta pe numele tabelului din care vrem să ştergem înregistrări
- alegem comanda "Edit Top 200 Rows"

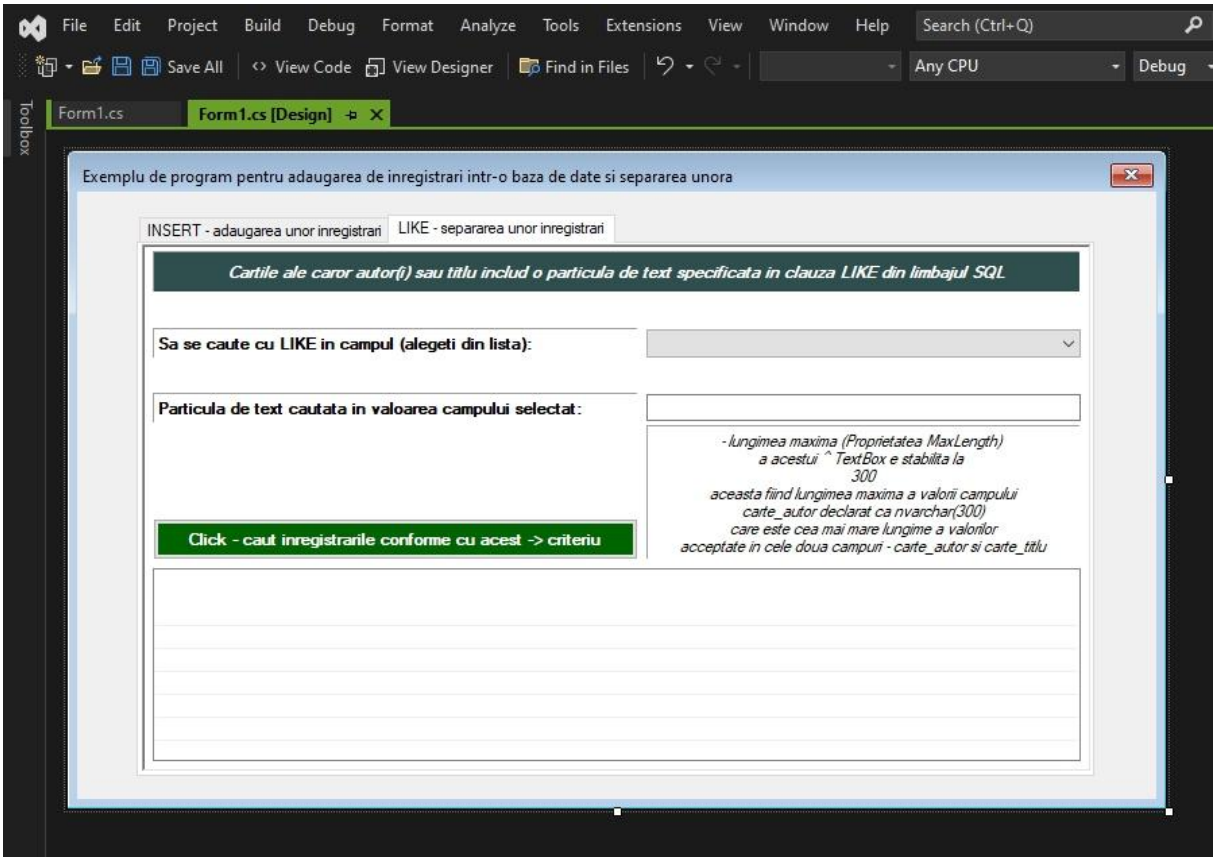


Apoi în lista înregistrărilor astfel afişate:

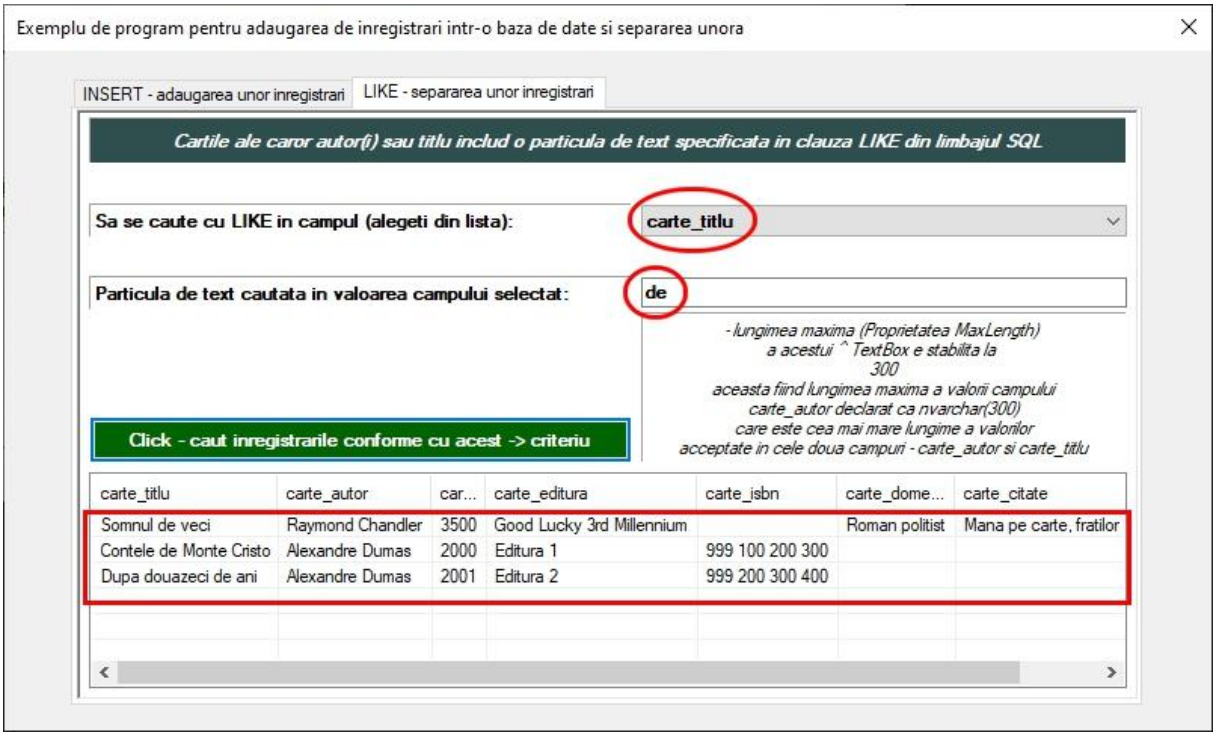
- click-dreapta în celula dinaintea celor cu valorile câmpurilor
- din meniul contextual care apare astfel se alege comanda "Delete":



Să proiectăm pagina cealaltă a *TabControl*-ului –
cea unde vom solicita înregistrările care se conformează unui criteriu de evaluare specificat cu clauza LIKE.



Rezultatul căutării particulei "de" în tabelul t_carti este:



Codul şi comentariile care explică aproape toate instrucţiunile se află în proiectul/soluţia
"curscs_bdlike.sln" din directorul/folderul omonim
din arhiva .zip asociată acestui document.

Principalul element care va fi folosit este

SqlDataReader

Cum s-a arătat mai sus:

SqlDataReader parcurge colecţia/lista înregistrărilor pe rând, de la prima la ultima înregistrare
aşa cum au fost obţinute după o comandă SELECT
din blocul de instrucţiuni specificate în textul ***SqlCommand*** asociate (Proprietatea **CommandText**).

Odată definit şi creat un ***SqlDataReader***,
valorile câmpurilor care alcătuie fiecare înregistrare
pot fi obţinute prin apelarea:

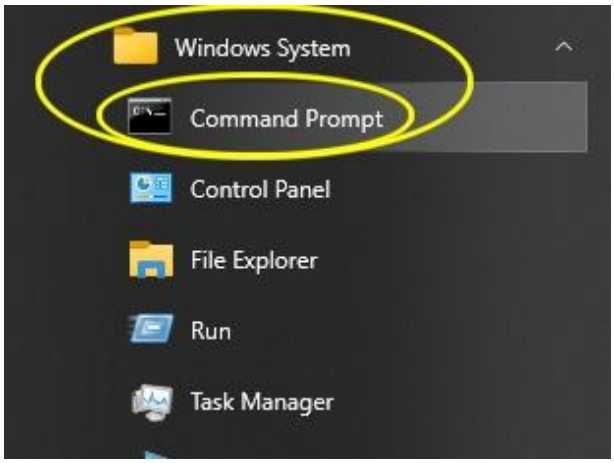
- numelui acelui câmp
- indexului/poziţiei în lista câmpurilor, numărând de la 0 încolo.

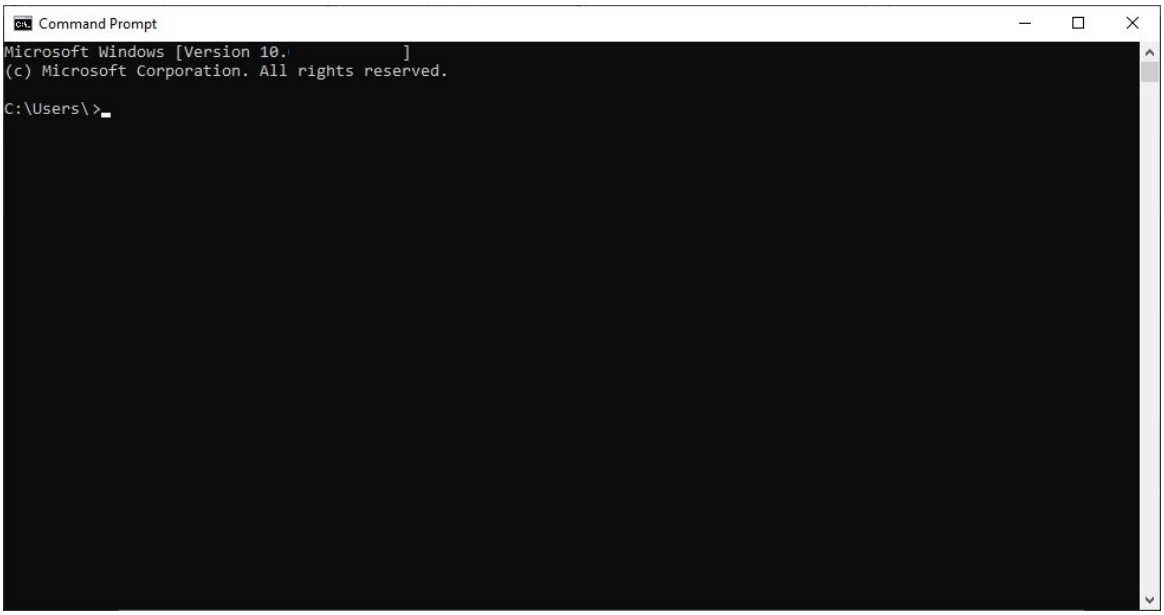
Crearea unei aplicaţii de tipul / pentru Consolă

Aceste aplicaţii diferă fundamental de cele cu *Form-e*, prin aceea că:

- **nu au interfaţă grafică** alcătuită din, sau care conţine
  - Dialoguri (*Form*-e) şi implicit Controale (cu contur) ca *Button*, *TextBox*, *Label* şi restul
- interacţiunea cu utilizatorul este limitată la
  - furnizarea de către acesta unor valori numerice sau text – inclusiv /caractere de control/ ca *Escape* şi *Enter* (trimis prin apăsarea tastelor *Esc/ape*, respectiv *Enter*) trimise
    - fie din aşa-numita **linie de comandă**
    - fie ca **argument(e)** de pornire (de start) a aplicaţiei atunci când aceasta este invocată (lansată, pornită) prin însuşi numele (denumirea) sa
      - de exemplu,
        - *aplicatie.exe argument1 argument2*

O aplicaţie de consolă tipică, integrată în sistemul de operare *Windows®™* este "Command Prompt" care într-o versiune de *Windows®™ 10* este astfel listată după apăsarea butonului *Start*:





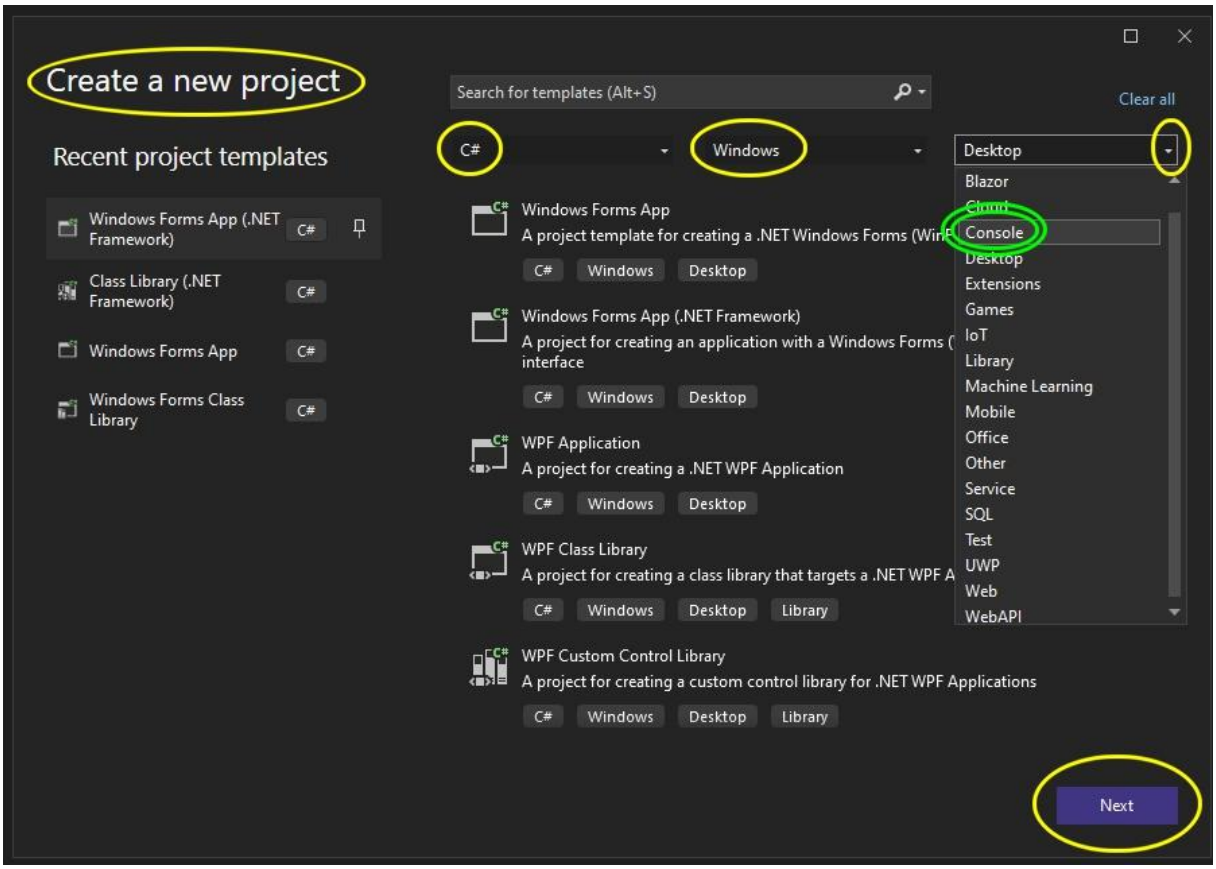
Să creăm o astfel de aplicaţie cu Visual Studio.

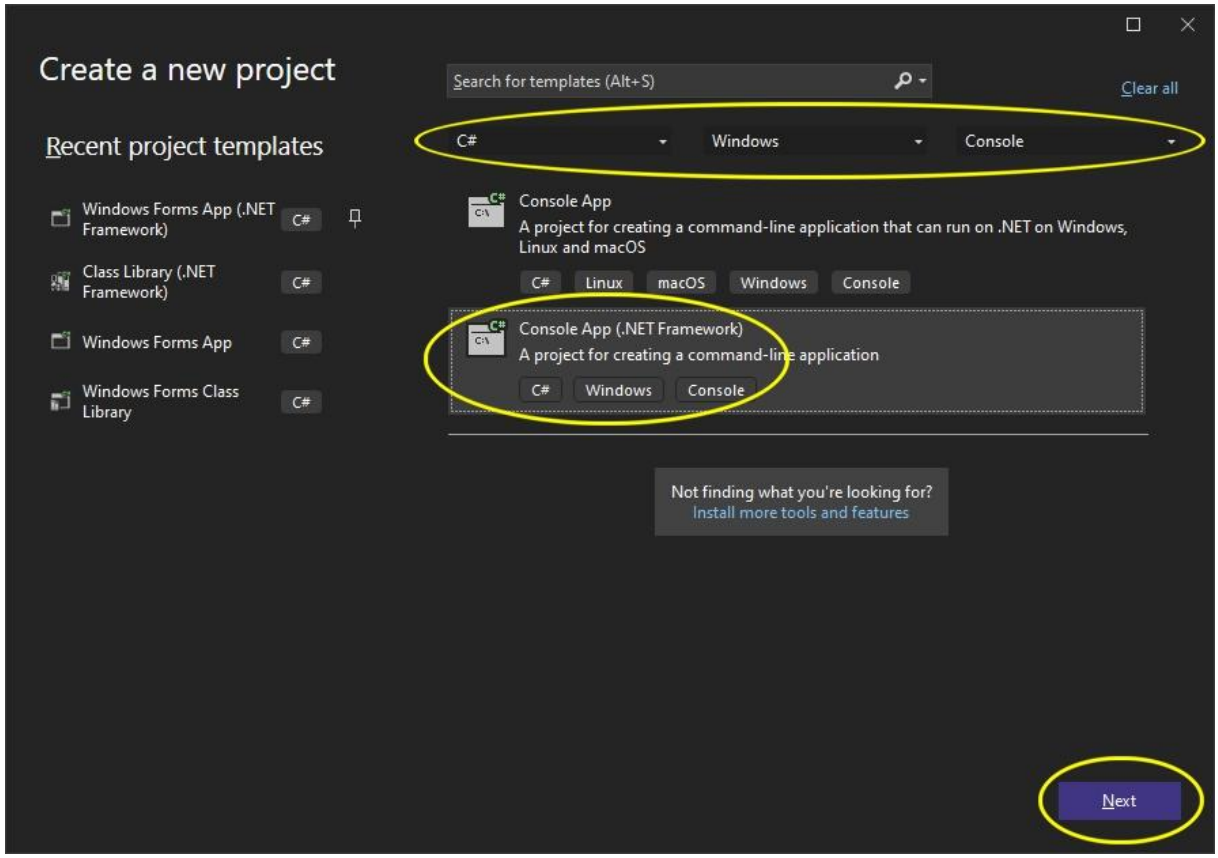
Vom solicita afişarea datei şi orei curente
într-o astfel de casetă simplă de dialog.

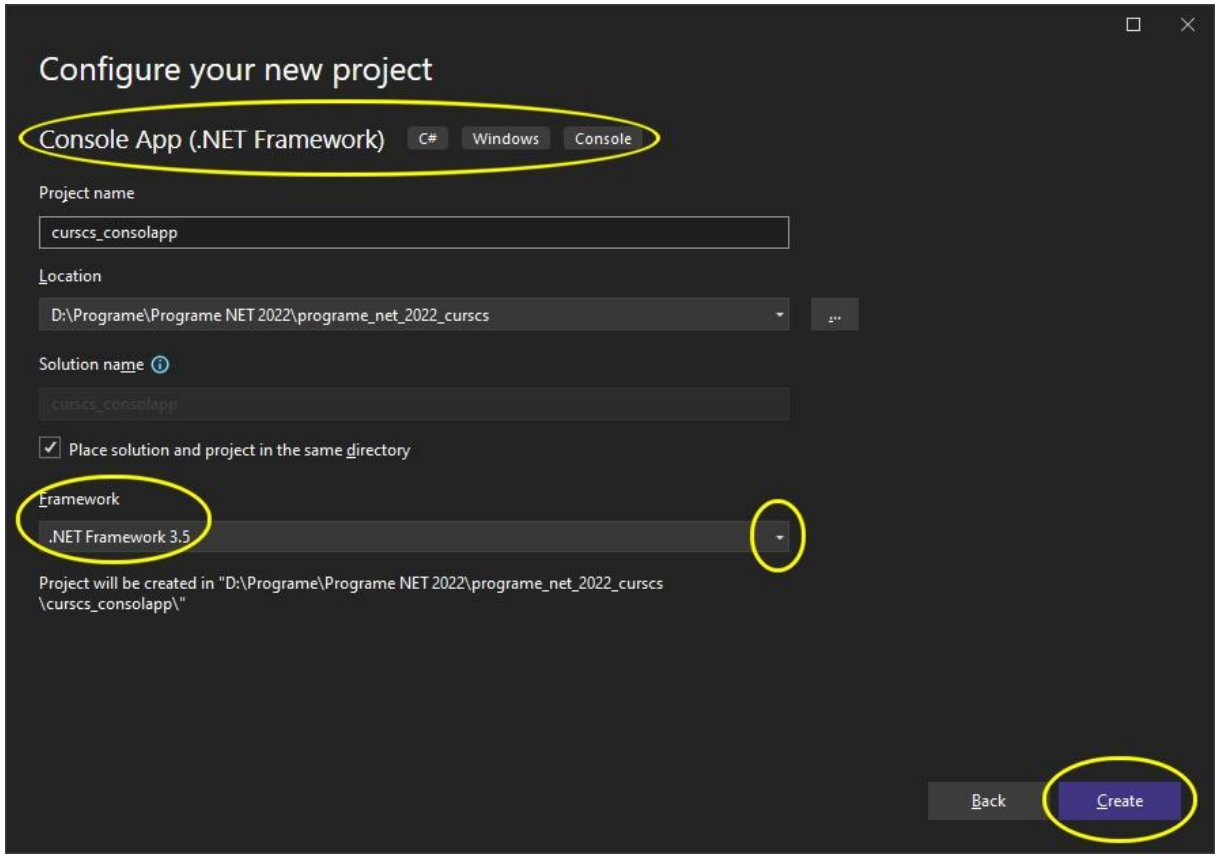
Click pe *New... Project...*

Din oferta de tipuri de aplicaţii se alege astfel:
în dreapta, "Console".

Deci, vrem să construim o aplicaţie (un program)
de tip *Consolă*,
scrisă în limbajul C#
şi destinată sistemului de operare *Windows*:




Create a new project
Search for templates (Alt+S)
Clear all
Recent project templates
C#
Windows
Console
Windows Forms App (.NET Framework)
C#
Class Library (.NET Framework)
C#
Windows Forms App
C#
Windows Forms Class Library
C#
Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C#
Linux
macOS
Windows
Console
Console App (.NET Framework)
A project for creating a command-line application
C#
Windows
Console
Not finding what you're looking for?
Install more tools and features
Next

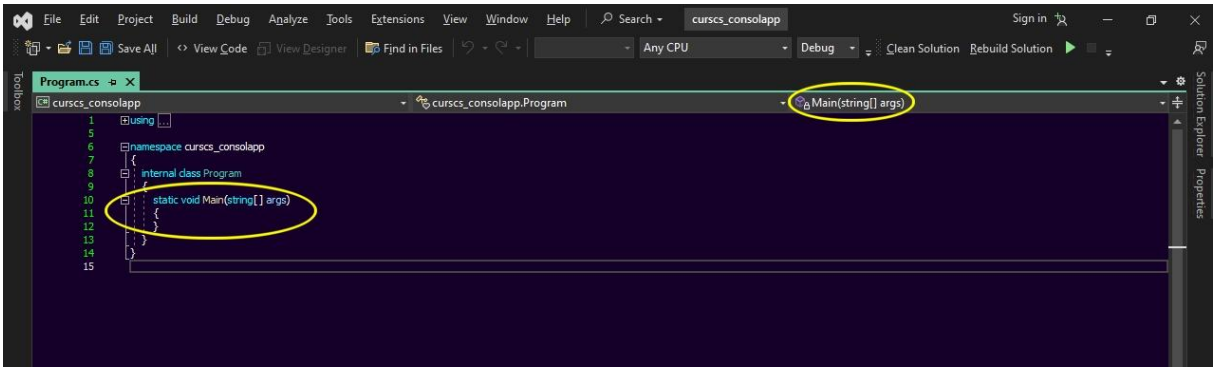


Observăm că, spre deosebire de o aplicaţie *Windows Forms*,
nu este afşată imediat o *Form*-ă,
pur şi simplu fiindcă aplicaţiile de tipul/pentru *Consolă* nu conţin implicit *Form*-e,
cu atât mai puţin una de start.

În schimb, este afişat codul aplicaţiei unde se vede **punctul de intrare în aplicaţie**:
adică, metoda care este executată la pornirea aplicaţiei.

Punctul de intrare în aplicaţie

- este denumit implicit **Main**
- şi este o metodă cu un număr nespecificat de argumente primite ca tablou de variabile *String*
    - tablou care poate avea dimensiune 0 zero,
        - anume, aplicaţia poate porni fără a primi vreun argument
            - trimis de către utilizator
                - ca o **înşiruire de texte separate prin spaţiu** după denumirea sa



Să scriem cod în această metodă, ***Main***:

Vom scrie cod pentru afişarea datei şi orei curente:

Întreg codul aplicaţiei acesteia este:

```
#region Usings
using System;
#endregion //Usings

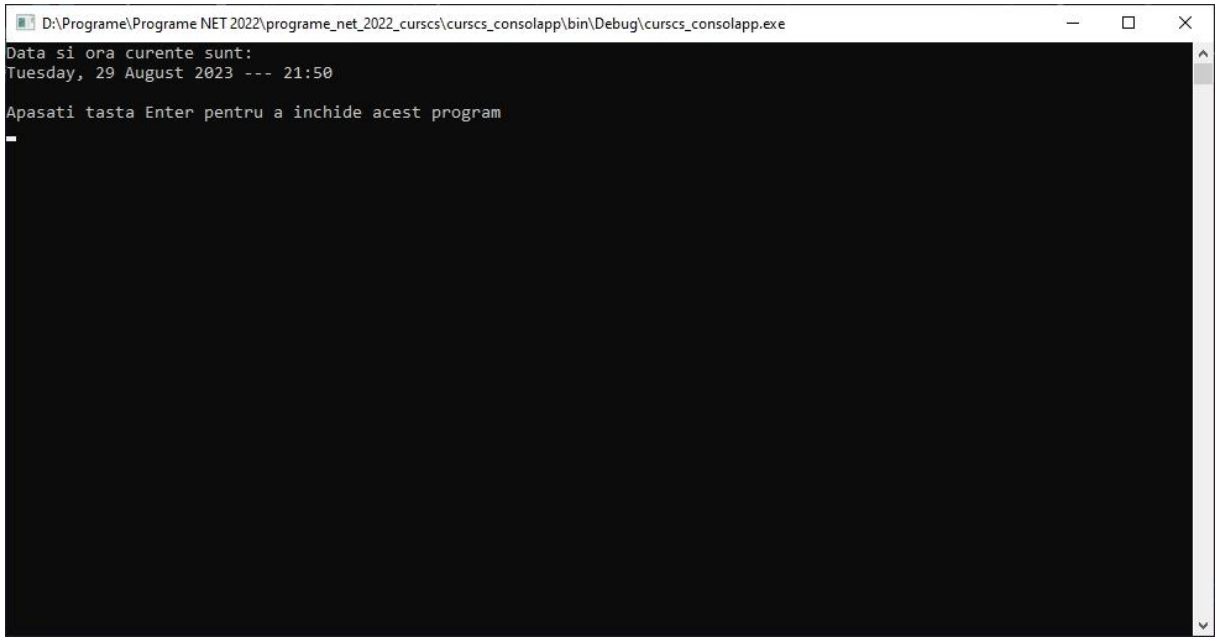
namespace curscs_consolapp
{
	internal class Program
	{
		static void Main(string[ ] args)
		{
			string sdataoracurente =
				DateTime.Now.ToString("dddd, dd MMMM yyyy --- HH:mm");
			//
			Console.WriteLine("Data si ora curente sunt:");
			Console.WriteLine(sdataoracurente);
			//
			Console.WriteLine();
			//
			Console.WriteLine("Apasati tasta Enter pentru a inchide acest program");
			//
			//se asteapta apasarea tastei Enter (Read Line - citeste, asadar, o linie goala)
			Console.ReadLine();
			//dupa care programul (aplicatia aceasta de Consola) va fi inchis(a)
		}
	}  //sfarsitul clasei Program
}  //sfarsitul programului curscs_consolapp
```

Citim linie cu linie:

```
string sdataoracurente = DateTime.Now.ToString("dddd, dd MMMM yyyy --- HH:mm");
```

Obţinem un obiect de tip:	**DateTime**
Din care extragem data şi ora curente:	**DateTime.Now**
Pe care îl formatăm ca text cu ***ToString()*** la:	**"dddd, dd MMMM yyyy --- HH:mm"**
	dddd: numele zilei în Cultura curentă a *Windows* de exemplu, "Tuesday"
	dd: numărul în lună al zilei de exemplu, "23"
	MMMM: numele lunii în Cultura curentă a *Windows* de exemplu, "June"
	yyyy: anul, formatat cu 4 cifre în Cultura curentă a *Windows* de exemplu, "2023"

Să lansăm programul: Tasta F5:

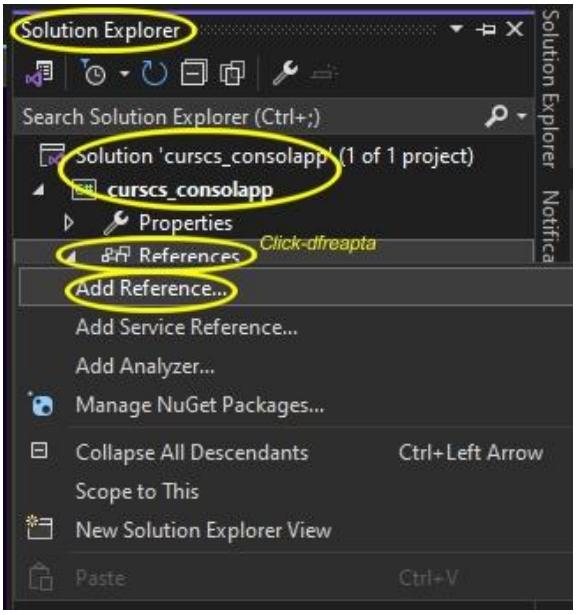


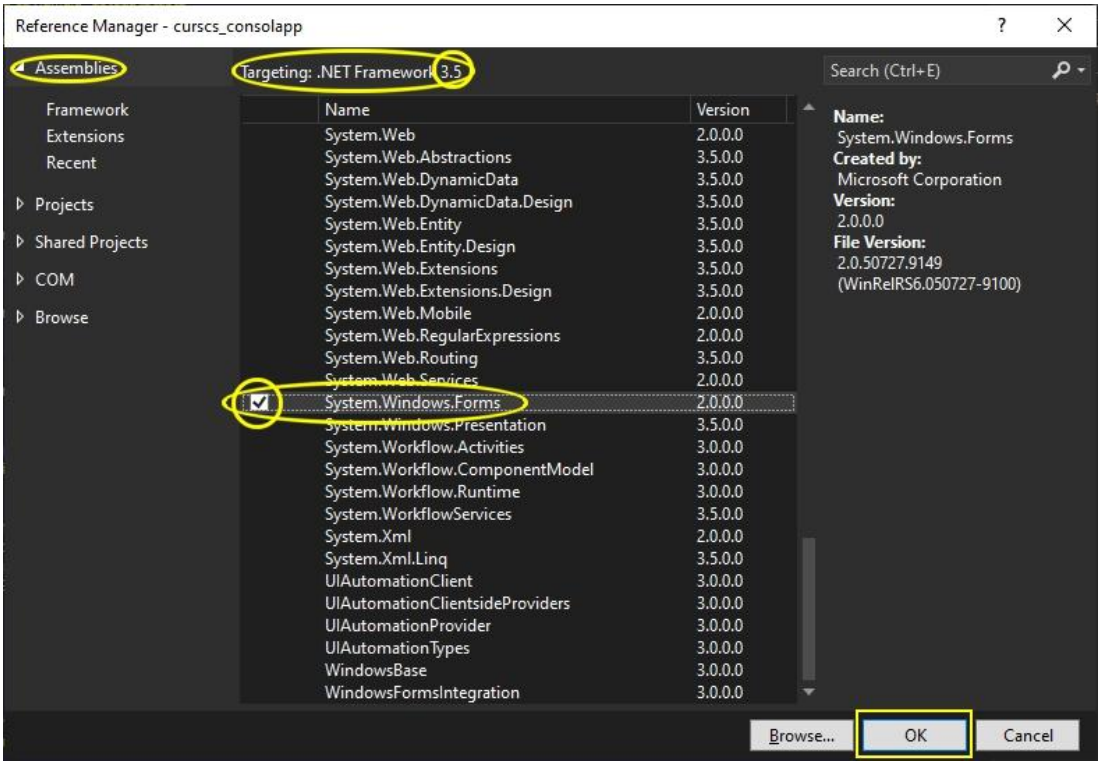
Apăsarea tasta Enter anunţă aplicaţia că a fost citită o linie (un rând) (metoda *ReadLine*)
şi după aceasta aplicaţia se închide – după *ReadLine()* nu este nicio altă instrucţiune.

Astfel, aplicaţia nu rămâne deschisă, afişată pe ecran ca o aplicaţie de tipul Windows Forms – ci se închide imediat ce toate instrucţiunile din* Main *sau/şi* apelate din Main *au fost executate.

O astfel de aplicaţie simplă poate fi extinsă prin referirea la diverse spaţii de nume în afară de cel accesibil în mod implicit, ***System***:

Aceasta se face adăugând o Referinţă la fiecare spaţiu de nume la secţiunea *References* a soluţiei/proiectului:





Astfel, în secţiunea *using* a unei astfel de aplicaţii pot fi incluse referinţe la spaţii de nume
ca în orice alt tip de aplicaţie,
de exemplu,

putem obţine acces la membrii spaţiului de nume ***Windows.Forms***
printre care

- ***MessageBox*** pentru afişarea unor casete de dialog – ***MessageBox.Show("Mesaj");***
- ***Application*** pentru apelarea comenzii de închidere a aplicaţiei – ***Application.Exit();***

using System.Windows.Forms;

sau la spaţiul de nume ***System.Data.SqlClient***
printre care

- ***SqlConnection*** pentru deschiderea unei conexiuni la un server (*RDBMS*) de baze de date
- ***SqlDataReader*** pentru parcurgerea unui set de înregistrări dintr-un tabel al unei baze de date obţnut cu o comandă SQL **SELECT**

using System.Data.SqlClient;

În imaginea precedentă se vede cum am adăugat o Referinţă către spaţiul de nume
System.Windows.Forms

Astfel, membrii acelui spaţiu de nume vor fi disponibili pentru utilizare în această aplicaţie de Consolă.

Schimbăm codul, întrucât vrem să apelăm membri ai *System.Windows.Forms*:

```
#region Usings
using System;
using System.Windows.Forms;
#endregion //Usings


namespace curscs_consolapp
{
	internal class Program
	{
		static void Main(string[ ] args)
		{
			string sdataoracurente =
				DateTime.Now.ToString("dddd, dd MMMM yyyy --- HH:mm");
			//
			Console.WriteLine("Data si ora curente sunt:");
			Console.WriteLine(sdataoracurente);
			//
			Console.WriteLine();
			//
			Console.WriteLine("Apasati tasta Enter pentru a continua cu o intrebare");
			//
			//aplicatia asteapta apasarea tastei Enter (Read Line - citeste, asadar, o linie goala):
			Console.ReadLine();
```

```
                //
                //apoi
                //aplicatia asteapta apasarea unei taste - metoda Console.ReadKey() –
                //tasta pe care o vom evalua:
                Console.WriteLine("Acum apasati tasta Y sau tasta N\r\n"+
                        "pentru a afisa un mesaj cu System.Windows.Forms.MessageBox()");
                Console.WriteLine("- sau apasati tasta Esc/ape incat aplicatia sa fie inchisa");
        //
        //definim o --- eticheta --- cu numele "asteaptatasta"
        //pentru fluxul programului
        //care va putea fi referita cu instructiunea
        //goto
        //
        asteaptatasta:
                //in acest punct definit de eticheta
                //ne vom putea intoarce referindu-ne la el cu instructiunea goto
                ConsoleKeyInfo cki = Console.ReadKey();
                switch (cki.Key)
                {
                        case System.ConsoleKey.Y:
                                //accesam spatiul de nume System.Windows.Forms
                                //specificat in References ale aplicatiei:
                                System.Windows.Forms.MessageBox.Show("Tasta Y");
                                break;
                        case System.ConsoleKey.N:
                                //accesam spatiul de nume System.Windows.Forms
                                //specificat in References ale aplicatiei:
                                System.Windows.Forms.MessageBox.Show("Tasta N");
                                break;
                        case System.ConsoleKey.Escape:
                                //accesam spatiul de nume System.Windows.Forms
                                //specificat in References ale aplicatiei:
                                System.Windows.Forms.Application.Exit();
                                break;
                        default:
                                //nu s-a apasat nicio tasta din cele evaluate in acest bloc switch
                                //asa ca se asteapta in continuare
                                goto asteaptatasta;
                                break;
                                //aici, break este inutil
                                //fiindca inaintea sa este instructiunea goto care intrerupe fluxul
                        comenzilor succesive
                }
                //
                //de vreme ce in sectiunea using
                //am specificat ca vom avea nevoie de spatiul de nume System.Windows.Forms
                //aceasta cale spre membrul MessageBox poate fi stearsa:
                //MessageBox.Show(...
        }


    } //sfarsitul clasei Program
} //sfarsitul programului curscs_consolapp
```

Corespondenţa între tablouri

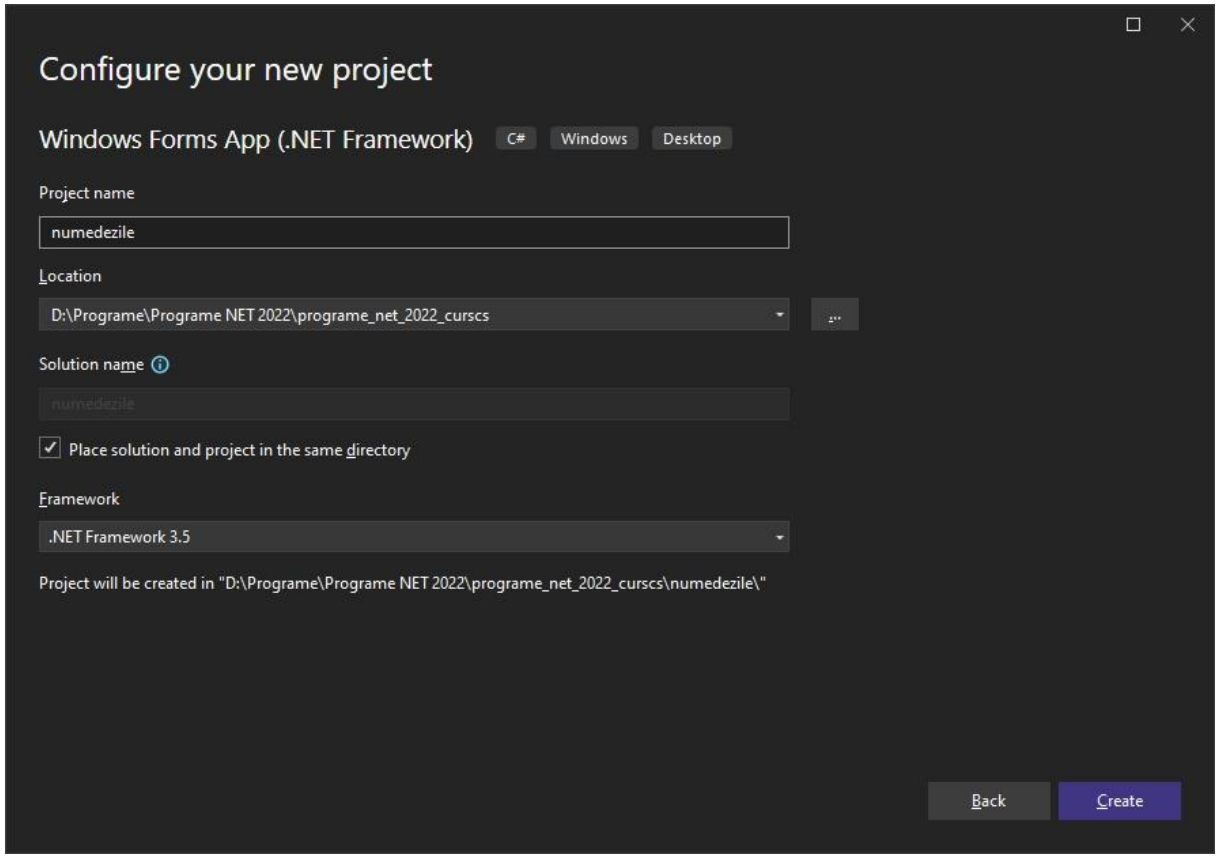
Vom crea un program unde va fi exemplificată punerea în corespondenţă a elementelor din două tablouri.

Vom folosi denumirile zilelor săptămânii, deci în cod vom lucra cu:
- 7 zile
- acestea fiind 7 variabile sau constante de tip *String*
- plasate fiind aceste 7 elemente într-un tablou de *String*-uri, adică o variabilă de tip *string[]*

Vom crea 2 două tablouri – anume,
- unul cu denumirile în limba română
- unul cu denumirile în limba engleză

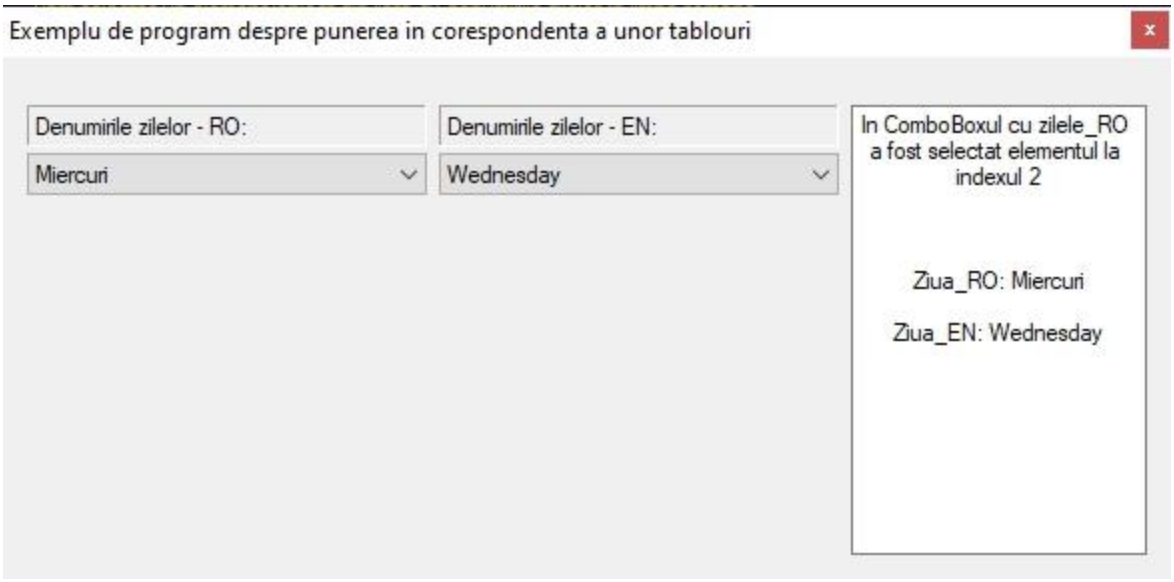
(pot fi oricare alte limbi, codul/modul de lucru va fi acelaşi)

Lansăm Visual Studio,
File -> New... Project...
creăm o soluţie Windows Forms,
o denumim "numedezile" (ca în arhiva ZIP cu proiectele din acest tutorial)
pe Form1 plasăm Controale care să ne permită accesul la fluxul programului



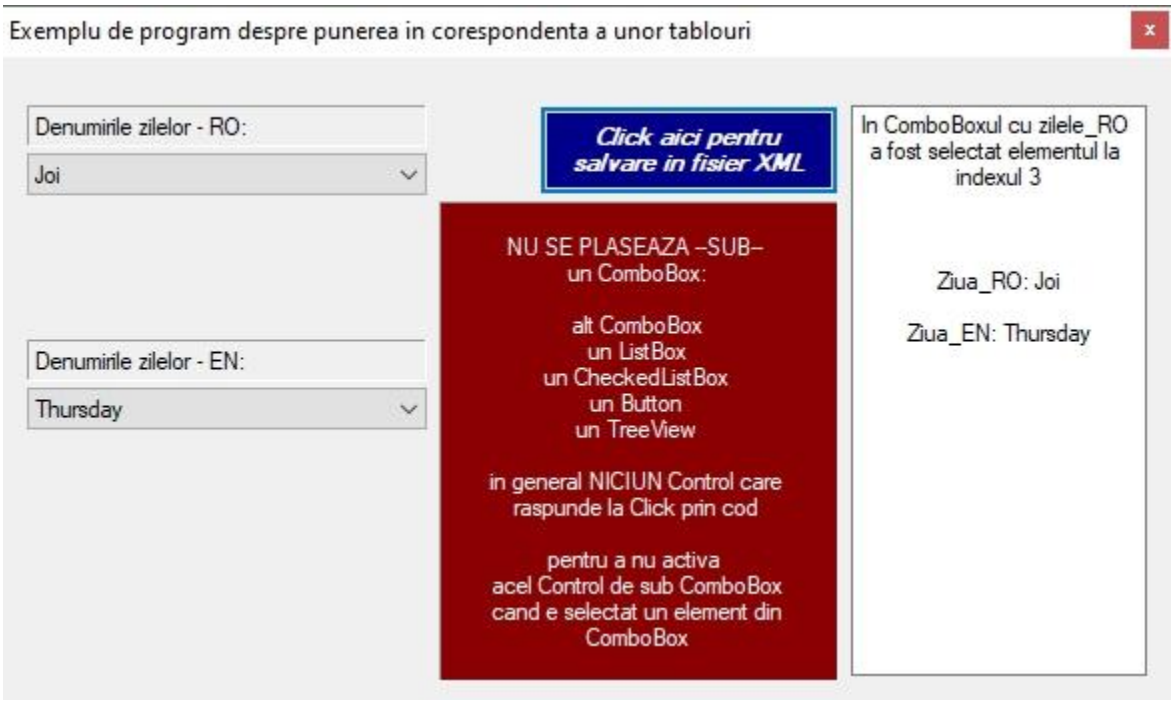
Metoda corespunzătoare Evenimentului *SelectedChanged* al ComboBox-ului
cu denumirile zilelor în limba română este:

```
private void cbzilero_SelectedIndexChanged(object sender, EventArgs e)
{
        //am selectat un element/obiect din lista cu denumirile zilelor in limba romana
        //1 - aflam indexul/pozitia acestui obiect in lista
        //2 - aflam ce element se afla la acel index/pozitie in tabloul cu denumirile in limba engleza
        //3 - si in ComboBoxul cu denumirile in limba engleza selectam obiectul de la acel index in tablou
        //
        //
        //1:
        int i_indexselectatcb_zile_ro = cbzilero.SelectedIndex;
        //2:
        int i_indexulintablou_zile_en = i_indexselectatcb_zile_ro;
        //3:
        cbzileen.SelectedIndex = i_indexulintablou_zile_en;
        //
        //in TextBox scriem un mesaj informativ despre elementul selectat:
        //intai stergem textul existent, desi va fi inlocuit prin comanda .Text =...;
        tbinfoelement.Clear();
        //daca indexul este mai mare decat -1, scriem mesajul:
        if (cbzilero.SelectedIndex > -1)
        {
                tbinfoelement.Text = "In ComboBoxul cu zilele_RO a fost selectat elementul la indexul " +
                                        i_indexselectatcb_zile_ro.ToString();
//adaugam += 4 randuri libere dupa textul existent:
                tbinfoelement.Text += "\r\n\r\n\r\n\r\n";
                tbinfoelement.Text += "Ziua_RO: " + cbzilero.Items[i_indexselectatcb_zile_ro].ToString();
                //adaugam += 2 randuri libere dupa textul existent
                tbinfoelement.Text += "\r\n\r\n";
                //folosim variabila indexului in tabloul cu denumirile zilelor in limba romana:
                tbinfoelement.Text += "Ziua_EN: " + cbzileen.Items[i_indexulintablou_zile_en].ToString();
                //
                //
                //
                //
                //codul acesta este pentru exemplificarea folosirii aceluiasi index in 2 doua tablouri
                //avand ambele aceeasi dimensiune
                //
                //codul este corect daca al doilea tablou, cel in care "se cauta"
                //are dimensiune cel putin egala cu cea a primului tablou
                //adica, elementul al 7-lea al saptelea de la indexul 6 in primul tablou
                //nu va fi in relatie directa dupa indexul 6 in al doilea tablou
                //cand acesta din urma are numai 6 sase elemente,
                //deci cand cel mai mare index in al doilea tablou este 5
        }
}
```



Acum salvăm într-un fişier XML denumirile zilelor.

Modificăm interfaţa, adăugând un Button pe care-l redenumim "**bxmlfilesave**"
pentru iniţierea acelei operaţiuni:



Codul comentat pentru salvarea denumirilor zilelor este prezentat în continuare (3 pagini)
şi este în proiectul/soluţia "numedezile" din arhiva ZIP asociată acestui tutorial.

Operaţiunea se face în metoda asociată Evenimentului *Click* al butonului din imaginea precedentă.

```
private void bxmlfilesave_Click(object sender, EventArgs e)
{
        //aici este cod pentru salvarea intr-un fisier XML
        //a denumirilor in ambele limbi ale zilelor
        //
        //
        //vom salva documentul XML intr-un fisier .xml
        //--- fara a folosi un Control de tipul SaveFileDialog ---- plasat pe aceasta Form-a ----
        //ci vom crea un astfel de Control chiar aici
        //
        //
        //pentru a putea accesa --- in aceasta clasa (Form1) ---
        //functionalitatea XML oferita de platforma .NET,
        //adica membrii spatiului de nume System.Xml,
        //am inserat instructiunea
        //using System.Xml;
        //in sectiunea using de la inceputul acestei clase
        //
        //
        //pregatim documentul XML in care vom scrie ce vrem:
        //
        XmlDocument xd = new XmlDocument();
        //nodul optional de initiere, initializare, a documentului:
        XmlNode xn = xd.CreateXmlDeclaration("1.0", "UTF-8", "yes");
        xd.AppendChild(xn);
        //
        //cream nodul-radacina a documentului XML
        //care nod va sustine - sub care se vor afla - toate celelalte noduri:
        //
        XmlNode xnroot = xd.CreateNode(XmlNodeType.Element, "zile", "");
        xd.AppendChild(xnroot);
        //
        //parcurgem elementele din fiecare ComboBox
        //si cream cate un nod pentru fiecare zi
        //nod ale carei atribute vor fi:
        //unul - denumirea zilei in limba romana
        //celalalt - denumirea aceleiasi zile in limba engleza
        //
        //asadar, in decursul iteratiei ne vom referi la acelasi index in ambele ComboBox-uri
        //
        //
        for (int i_indexincbox_ro = 0; i_indexincbox_ro < cbzilero.Items.Count; i_indexincbox_ro++)
        {
                XmlNode xn_zi = xd.CreateNode(XmlNodeType.Element,
                                "ziua_" + (i_indexincbox_ro + 1).ToString(), "");
                //^ numele fiecarui nod va fi "ziua_"
                //urmat de numarul zilei obtinut din suma
                //indexului in ComboBox (primul index fiind 0) la care se adauga 1
                //asadar, ziua_1, ziua_2 samd.
                //
                //acum cream Atributele acestui nod:
                //
                //atributul a carui Valoare va fi denumirea in limba romana:
                XmlAttribute xa = xd.CreateAttribute("denumire_zi_ro");
```

```
                //valoarea acestui atribut este denumirea zilei de la indexul in ComboBoxul_RO:
                xa.Value = cbzilero.Items[i_indexincbox_ro].ToString();
                //plasam acest Atribut in nodul acestei zile:
                xn_zi.Attributes.Append(xa);
                //
                //atributul a carui Valoare va fi denumirea in limba engleza:
                //denumim aceasta variabila tot "xa", adica o redefinim pe cea dinainte:
                xa = xd.CreateAttribute("denumire_zi_en");
                //valoarea acestui atribut este denumirea zilei de la indexul in ComboBoxul_EN:
                xa.Value = cbzileen.Items[i_indexincbox_ro].ToString();
                //plasam acest Atribut in nodul acestei zile:
                xn_zi.Attributes.Append(xa);
                //
                //acum plasam nodul acestei zile din iteratie,
                //cu tot cu Atributele sale tocmai adaugate,
                //sub nodul-radacina al Documentului XML
                //deci de la prima denumire in ComboBox-ul cu denumirile in limba romana
                //--- care are acelasi numar de elemente cu ComboBox-ul cu denumirile in limba engleza
                //--- si ale carei elemente corespund fiecare cu elementele din cealalta ComboBox ---
                //
                xnroot.AppendChild(xn_zi);
        }
        //odata adaugate nodurile /cu toate elementele din ComboBox-uri,
        //salvam Documentul XML:
        //
        //
        //incepem a lucra cu fisiere, asa ca plasam codul respectiv intr-un bloc try-catch(-finally):
        //
        try
        {
                //cream un Control de tipul SaveFileDialog:
                SaveFileDialog sfd = new SaveFileDialog();
                sfd.Title = "Specificati numele fisierului XML si locatia/folderul sau";
                sfd.Filter = "Fisiere XML (*.xml)|*.xml";
                //^ la stanga | se afla textul de afisat in ComboBox-ul cu tipurile fisierelor /accesibile
                //^ la dreapta | se afla extensia/extensiile fisierelor de tipul astfel selectat
                //
                //ne asiguram ca nu ni se propune niciun nume pentru fisier
                //de regula, ar fi numele ultimului fisier salvat cu acest SaveFileDialog:
                sfd.FileName = "";
                //
                //afisam acest SaveFileDialog pentru ca utilizatorul sa specifice numele si locatia fisierului:
                if (sfd.ShowDialog() != DialogResult.Cancel)
                {
                        //daca a fost specificat/scris un nume care nu consta numai din spatii libere/blankuri
                        //pe care le eliminam de la inceput si de la sfarsit cu metoda Trim()
                        //atunci salvam Documentul XML in fisierul cu acel nume:
                        if (sfd.FileName.Trim().Length > 0)
                        {
                                xd.Save(sfd.FileName);
                                //anuntam succesul incercarii de salvare a Documentului XML
                                //in fisierul specificat:
                                MessageBox.Show("Documentul XML cu denumirile zilelor in cele doua
limbi\r\na fost salvat ca:\r\n\r\n" + sfd.FileName);
                        }
```

```
            }
        }
        catch (Exception ex)
        {
            //nu s-a putut
            //- fie crea Controlul SaveFileDialog
            //- fie crea fisierul cu acel nume
            //-- in acea locatie
            //- fie scrie in acel fisier
            //- fie salva fisierul
            //aflam ce s-a intamplat
            //afisand detaliile erorii "ex" cu metoda
            //.ToString()
            MessageBox.Show("EROARE la salvarea Documentului XML:\r\n\r\n" + ex.ToString());
        }
}
```

