Laboratorul 2

Controale avansate în Visual C#

Ce ne propunem astăzi?

În acest laborator ne propunem să implementăm în Visual C# .NET o agendă personală în care vom memora numele, data de naștere, telefonul si adresa persoanelor cunoscute (vezi Figura 1).

Mai pe larg, vom proceda astfel…

Pentru început va trebui să populăm agenda cu date ale persoanelor cunoscute. La adăugarea unei persoane în agendă, aceasta va fi inclusă într-una din următoarele categorii: prieteni, colegi, rude sau diverși (categoria implicită).

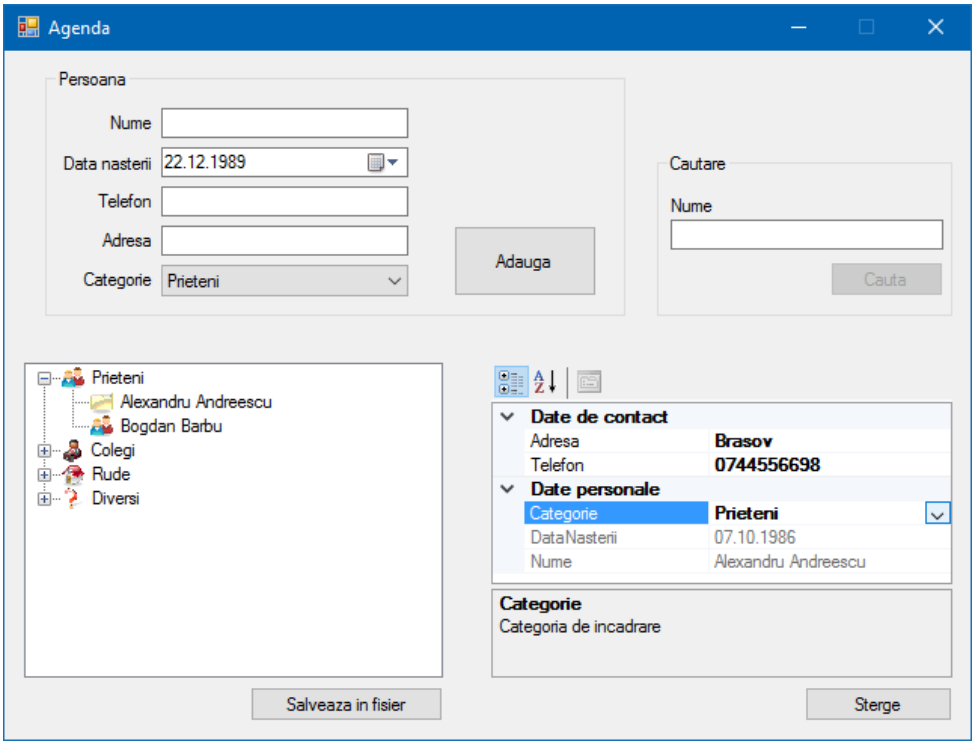

Figura 1. Interfața aplicației Agenda

Vom utiliza în acest laborator două controale noi: TreeView și PropertyGrid. Controlul TreeView permite vizualizarea elementelor sub formă arborescentă, iar controlul PropertyGrid afișarea tuturor proprietăților publice ale unui obiect atașat acestuia.

Lucrul cu proprietăți publice, cel puțin în cazul aplicației curente, este de preferat în locul câmpurilor sau metodelor publice ale clasei, din două motive:

- ne permite accesarea unor câmpuri numai în mod citire (read-only);
- ne permite afișarea acestora în interiorul unui control de tip PropertyGrid.

Din punct de vedere al programării orientate pe obiecte, ca o extensie a principiului încapsulării, este de preferat folosirea proprietăților publice pentru accesarea câmpurilor (variabilelor membre) private ale unei clase. Același principiu era aplicat adeseori și în limbajul C++, atunci când câmpurile private ale unei clase erau accesate din exterior prin intermediul metodelor publice.

Înainte de crearea unei proprietăți se va declara o variabilă membru (asupra căreia va putea opera proprietatea), apoi va fi declarată proprietatea împreună cu accesorii `get` și `set`, care permit returnarea unei valori, respectiv preluarea unei valori. Iată un exemplu de proprietate read-only:

```
public string DataNasterii
{
    get
    {
        return dataNasterii.ToString("dd.MM.yyyy");
    }
}
```

Și un exemplu de proprietate care poate fi accesată atât în citire cât și în scriere:

```
public string Telefon
{
    get
    {
        return telefon;
    }

    set
    {
        telefon = value;
    }
}
```

Controlul TreeView conține și afișează o ierarhie de noduri (elemente de tip `TreeNode`) în colecția de noduri `Nodes`. Un nod conține informație text (de exemplu, numele unei persoane) și poate avea o imagine asociată. Orice nod poate conține la rândul lui alte noduri (noduri copil).

Exemplul de mai jos ilustrează modul de adăugare a unui nod părinte în rădăcină și a unui nod fiu într-un control TreeView (`treeView1`).

```
//adaugă un nod nou în rădăcină
TreeNode parentNode = new TreeNode();
parentNode.Name = "parent_1";
parentNode.Text = "Prieteni";
parentNode.ImageIndex = 1;
treeView1.Nodes.Add(parentNode);

//adaugă un nod fiu la nodul creat anterior
TreeNode childNode = new TreeNode();
childNode.Name = "child_1";
childNode.Text = "Popovici Iulian";
childNode.ImageIndex = 2;
parentNode.Nodes.Add(childNode);
```

În exemplul de mai sus, proprietatea `Name` a unui nod reprezintă numele acestuia și permite o identificarea mai ușoară a nodului din codul sursă (numai în cazul în care folosiți o convenție bine aleasă). De exemplu, pentru adăugarea nodului copil, nodul părinte ar fi putut fi identificat prin nume, astfel:

```
treeView1.Nodes["parent_1"].Nodes.Add(childNode);
```

Proprietatea `ImageIndex` reprezintă indexul pictogramei afișate de un nod din TreeView (vezi în continuare semnificația acesteia).

Efectul vizual al executării codului de mai sus îl puteți observa în Figura 2.

Figura 2. *Elementele adăugate în controalele TreeView și ListView*

Pentru a-i putea asocia o pictogramă fiecărui nod din TreeView va trebui să adăugăm la formular o componentă de tip ImageList, denumită `imgSmall`, care va conține imagini cu dimensiunea 16x16 pixeli (vezi Figura 3).

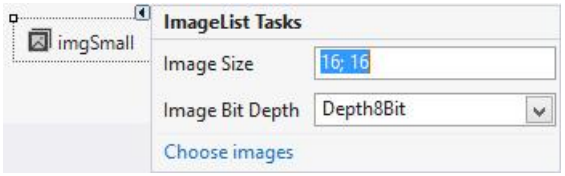

Figura 3. *Setarea unor proprietăți asociate componentei imgSmall*

Pentru a asocia lista de pictograme mici (16x16) cu controlul TreeView setați-i acestuia proprietatea `ImageList` la valoarea „imgSmall”. Pictograma afișată pentru un nod din TreeView este determinată de proprietatea `ImageIndex` a unui element de tip TreeNode și reprezintă indexul pictogramei din lista de imagini asociată controlului.

Toate operațiile asupra nodurilor dintr-un control TreeView se efectuează prin intermediul colecției de noduri `Nodes`. Iată, de exemplu, cum putem să golim conținutul unui control TreeView:

```
treeView1.Nodes.Clear()
```

Pentru memorarea internă a persoanelor incluse în agendă se vor crea (preferabil în fișiere separate):

- clasa Persoana care să stocheze datele specifice acestora;
- enumerarea Categorii pentru încadrarea tuturor persoanelor din agendă;

```
class Persoana
{
    private int index;
    private string nume;
    private DateTime dataNasterii;
    private string telefon;
    private string adresa;
    private Categorie categorie;

    // + un constructor cu parametri

    //aici se vor defini urmatoarele proprietati publice:
    //Index: read-only
    //Nume: read-only
    //DataNasterii: read-only
    //Categorie: read-write
    //Telefon: read-write
    //Adresa: read-write
}

enum Categorie : int
{
    Prieteni,
    Colegi,
    Rude,
    Diversi
};
```

În interiorul clasei formular se va crea o listă generică de persoane:

```
List<Persoana> agenda = new List<Persoana>();
```

Sfaturi utile

Cea mai simplă modalitate de parcurgere a persoanelor din lista generică este utilizarea unei bucle foreach.

```
foreach (Persoana p in agenda)
{
    //p - utilizat pentru parcurgerea elementelor
}
```

Specificarea de atribute pentru proprietățile clasei

Atunci când definiți proprietățile publice ale clasei Persoana, specificați pentru fiecare dintre ele două atribute: *Description* și *Category*. Acestea vă vor fi utile atunci când un obiect de tip Persoana va fi vizualizat în controlul PropertyGrid.

În C# atributele pentru elementele de program se definesc astfel:

```
[Description("Numele complet al persoanei"), Category("Date personale")]
public string Nume
{
    get {return nume;}
}
```

Efectele specificării atributelor *Description* și *Category* pentru proprietățile publice ale clasei Persoana sunt, pe de-o parte, gruparea vizuală a proprietăților pe categorii, iar pe de altă parte, afișarea descrierii la selectarea unei proprietăți în controlul PropertyGrid (vezi Figura 4).

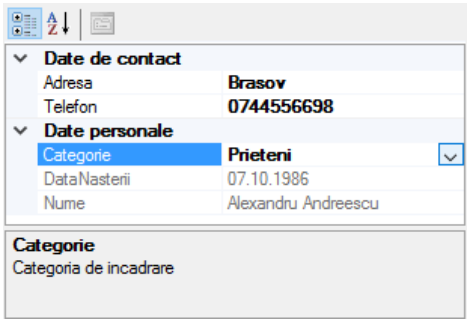

Figura 4. Efectele specificării atributelor Description *și* Category *pentru proprietățile clasei* Persoana

Observați faptul că proprietățile read-only Nume și DataNasterii sunt marcate cu culoare gri, ceea ce indică faptul că acestea nu pot fi modificate, ci doar vizualizate.

Dorim de asemenea ca proprietatea Index a clasei Persoana să nu fie afișată în controlul PropertyGrid, de aceea îi vom seta atributul *Browsable* false.

```
[Description("Index"), Browsable(false)]
public int Index
{
    get { return index; }
}
```

Lucrul cu controlul PropertyGrid

Datele cunoscuților din agendă vor fi memorate într-o listă generică de obiecte de tip Persoana, fiind afișate în TreeView, iar nodul selectat în controlul PropertyGrid. Iată cum afișați în PropertyGrid proprietățile unui anumit obiect.

```
Persoana p = new Persoana(/*...*/);
```

```
PropertyGrid1.SelectedObject = p;  //afisarea unei persoana in PropertyGrid1
```

În urma operației de căutare în agendă, dacă persoana căutată este găsită, se vor afișa detaliile acesteia în controlul PropertyGrid. Căutarea se va face după numele complet al persoanei.

La operația de ștergere a unei persoane din agendă veți afișa un dialog de confirmare a operației (vezi Figura 5).

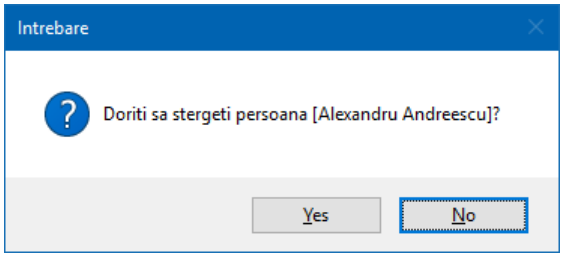

Figura 5. Dialogul de confirmare a ștergerii unei persoane din agendă

Afișarea unui astfel de dialog se face astfel:

```
if (MessageBox.Show("Doriti sa stergeti persoana [" + p.Nume + "]?",
 "Intrebare", MessageBoxButtons.YesNo, MessageBoxIcon.Question,
 MessageBoxDefaultButton.Button2) == DialogResult.Yes)
{
    // actiunea de stergere a persoanei p
}
```

Salvarea și încărcarea datelor din fișier

După introducerea persoanelor în listă, se dorește salvarea acesteia sub formă de fișier text și deschiderea fișierului cu editorul implicit de text al sistemului de operare. Datele din fișier vor fi formatate precum se poate observa în Figura 6.

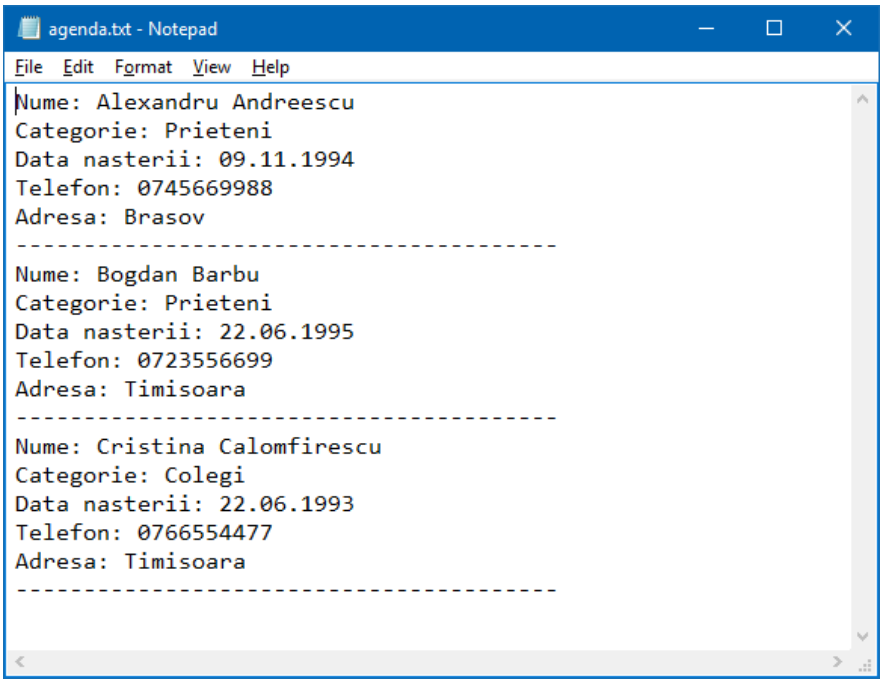

Figura 6. Afișarea datelor salvate în fișier

Pentru crearea unui fișier text și scrierea de date în acesta este nevoie de un obiect de tip StreamWriter (tipul este definit în spațiul de nume `System.IO`). Pentru scrierea unei linii de text se apelează metoda `WriteLine` a obiectului de tip StreamWriter. Funcția `Process.Start` (din

spațiul de nume `System.Diagnostics`) lansează în execuție un proces nou, permițând eventual transmiterea parametrilor pentru apelul din linie de comandă a acestuia. Va fi nevoie de utilizarea a două directive `using` în fișierul sursă.

```
using System.IO;
using System.Diagnostics;
//...

// Scrierea in fisier
string dir = Application.StartupPath;
StreamWriter sw = new StreamWriter(dir + "\\agenda.txt", false);
//...
sw.WriteLine("Nume: " + p.Nume);
//...
sw.Close();
Process.Start("notepad.exe", dir + "\\agenda.txt");
```

Operarea cu date de tip enumerare

Într-o enumerare elementele au asociate o denumire și un index numeric. Daca nu precizați în declarația enumerării în mod explicit un index pentru fiecare element, acestea vor primi automat câte un index numeric începând cu 0. Operarea cu elementele unei enumerări poate ridica unele probleme. Astfel:

a) Pentru utilizarea ca text a denumirii unui element dintr-o enumerare folosiți metoda `ToString()`.

```
//obtine denumirea elementului enumerarii
sw.WriteLine("Categorie: " + p.Categorie.ToString());
```

b) Pentru conversia pornind de la indexul numeric la unul din elementele enumerării folosiți operatorul cast.

```
//conversie int->Categorii
Categorii f = (Categorii)(cmbCategorie.SelectedIndex+1);
```

c) Pentru a afla elementul unei enumerări pornind de la denumirea lui utilizați metoda `Enum.Parse`.

```
//conversie string -> Categorii
Categorii f = (Categorii)Enum.Parse(typeof(Categorii), "Prieteni");
```

Sfaturi utile

Dată fiind formatarea datelor din fișierul text, pentru obținerea unei părți dintr-un șir de caractere obținut la citirea unei linii din fișier, veți avea nevoie să utilizați metoda `Substring` *a clasei* `System.String`.

Modul de lucru sugerat

- La pornirea aplicației se vor adăuga prin cod (procedura eveniment `Form1_Load`) în controlul TreeView cele patru categorii de persoane (patru noduri părinte).
- La adăugarea unei persoane noi în agendă, ea va fi adăugată întâi în lista generică `agenda` apoi în controlul TreeView ca nod fiu al categoriei din care face parte.
- La click în controlul TreeView pe un nod categorie nu se va întâmpla nimic.

- La click în controlul TreeView pe o persoană, se vor afișa în controlul PropertyGrid detaliile persoanei selectate (vezi Figura 4).
- Operația de adăugare/ ștergere a unei persoane în/din listă atrage după sine refacerea conținutului controalelor TreeView respectiv PropertyGrid (conform conținutului listei generice).
- La adăugarea unei persoane noi în agendă trebuie verificat ca nu cumva acea persoană să fie deja introdusă. Verificarea existenței în agendă a unei persoane se va face după numele acesteia. Pentru căutarea în agendă a persoanelor după un anumit criteriu, în cazul în care doriți să evitați parcurgerea tuturor elementelor din listă, utilizați metoda `Find` a listei generice. Mai multe informații despre utilizarea acesteia găsiți în help-ul Visual Studio la o căutare după **List<T>.Find(Predicate<T>)**.

Cu ce ne-am ales?

Prin aplicația dezvoltată în cadrul laboratorului de astăzi am reușit să ne familiarizăm cu modul de utilizare a controalele TreeView și PropertyGrid, extrem de utile pentru realizarea unor aplicații cu interfețe grafice complexe. Totodată am învățat cum putem să scriem date într-un fișier de tip text.

Resurse

[1] Windows Forms overview, https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview

[2] Building Windows Forms Applications With C#, https://www.c-sharpcorner.com/UploadFile/84c85b/building-windows-forms-application-with-C-Sharp/

[3] Programming in C# Jump Start, https://mva.microsoft.com/en-us/training-courses/programming-in-c-jump-start-14254#!