Laboratorul 1

Primii paşi în Visual C#

Ce ne propunem astăzi?

În laboratorul de astăzi ne propunem crearea unei aplicaţii simple pentru evidenţa studenţilor unei facultăţi. În cadrul acestei aplicaţii vom utiliza câteva dintre facilităţile POO oferite de Visual C#. Din păcate vom arăta numai o mică parte din acestea, urmând să învăţăm mai mult la curs şi pe parcursul lucrărilor de laborator ce vor urma.

Microsoft **Visual Studio .NET** este un mediu de dezvoltare folosit pentru crearea de aplicații în limbajele de programare: C#, Visual Basic, C++ sau F#.

Visual Studio .NET dispune de un editor complex de cod, care se folosește pentru crearea, modificarea și depanarea codului aplicației (poate fi vorba de cod Visual Basic .NET, Visual C# .NET etc.). De asemenea, oferă un set de obiecte și instrumente cu ajutorul cărora se pot realiza cu ușurință interfețe cu utilizatorul pentru Windows, Web, servicii WEB etc.

Visual C# .NET (sau, mai simplu, C#) este un limbaj de programare care face parte din familia de limbaje .NET, după cum o sugerează și numele, care păstrează sintaxa specifică limbajelor din categoria „C". Este un limbaj modern, puternic, complet orientat pe obiecte, cel puțin la fel de valoros ca Visual Basic, Visual C++ sau Java, care permite dezvoltarea atât de aplicații Windows cât și de aplicații și servicii WEB.

O aplicație tipică Windows, realizată în C#, afișează una sau mai multe ferestre conținând obiecte cu care utilizatorul va interacționa. Obiectele vizuale ale aplicației sunt ferestrele (denumite de asemenea „Forms", sau formulare) și controalele desenate pe ele. În cadrul unei aplicații Windows formularele sunt elementele de bază ale interfeței cu utilizatorul. În C# formularele sunt complet orientate pe obiecte, ele reprezentând de fapt clase.

În momentul în care se creează un nou formular, fie deschizând un proiect nou de tip Windows Forms Application, fie adăugând un nou formular proiectului existent (prin comanda *Add Windows Form* din meniul *Project*), i se adaugă de fapt proiectului o clasă care derivă din clasa `System.Windows.Forms.Form`, căreia i se mai spune pe scurt clasa Form, iar noua clasă moștenește toți membrii clasei existente.

Controalele reprezintă suportul interfeței cu utilizatorul, al dialogului cu acesta. Acestea nu pot exista independent de formular, formularul fiind din acest punct de vedere un container pentru controale.

După crearea formularelor și adăugarea controalelor dorite pe acestea, programatorul trebuie să adauge cod sursă și să-l atașeze unor evenimente care să permită utilizatorilor să interacționeze cu programul. De exemplu, dacă se dorește ca la apăsarea unui buton de comandă să se deschidă un formular, în metoda corespunzătoare evenimentului Click asociată acelui buton se va plasa codul corespunzător deschiderii și afișării formularului.

Evenimentele se produc ca urmare a acțiunilor utilizatorului (de exemplu, apăsarea unei taste, efectuarea unui click sau deplasarea cursorului de mouse), a execuției programului, sau pot fi

declanșate de sistem. Producerea unui eveniment (Click, Double Click, Drag&Drop, KeyPress etc.) declanșează execuția unei proceduri eveniment. Utilizatorul poate scrie propriul cod în corpul acestei proceduri. Apelarea unei proceduri eveniment pentru un obiect dat se face automat, dacă obiectul e focalizat și dacă se produce o acțiune care să declanșeze respectivul eveniment. Procedura eveniment poate fi apelată și prin cod, la fel ca orice altă procedură, însă acest lucru nu va duce și la declanșarea evenimentului asociat.

Metodele obiectelor vizuale, spre deosebire de procedurile eveniment, sunt read-only. Ele pot fi doar apelate. Există câteva metode comune tuturor obiectelor vizuale, în rest fiecare obiect având propriile lui metode. Metodele se apelează în felul următor:

```
Nume_Obiect.Nume_Metoda(Lista_Argumente);
```

Atât formularele cât și controalele au asociate un set de proprietăți prin intermediul cărora pot fi setate caracteristici ale acestora, cum ar fi:

- poziția și dimensiunea
- stilul și comportamentul
- aspectele grafice etc.

Unele proprietăți pot fi modificate la proiectare (folosind fereastra de proprietăți), altele pot fi modificate doar în timpul execuției, iar altele permit accesul atât la proiectare cât și la execuție. La execuție, valoarea unei proprietăți poate fi modificată prin instrucțiuni de forma:

```
Nume_Obiect.Nume_Proprietate = Valoare;
```

Realizarea unei aplicații de tip Windows Forms Application în C# implică, în linii mari, parcurgerea următoarelor etape:

- proiectarea mentală sau pe hârtie a interfeței utilizator,
- crearea unui proiect nou,
- crearea de formulare (unul pentru fiecare fereastră a aplicației),
- adăugarea de controale formularelor, folosind caseta de instrumente (ToolBox),
- crearea unei bare de meniu și/sau a unei bare de instrumente (ToolBar) pentru funcțiile principale ale aplicației (opțional),
- setarea proprietăților formularelor și controalelor,
- scrierea codului (declarații de variabile, funcții, proceduri sau clase, respectiv asocierea de cod pentru evenimentele obiectelor vizuale),
- testarea aplicației,
- crearea fișierului executabil și a kit-ului de distribuție.

Noțiuni elementare de Programare Orientată pe Obiecte în C#

Clase și proprietăți

O clasă poate conține câmpuri, metode sau proprietăți. Membrii claselor pot avea oricare din specificatorii de acces: *public*, *protected internal*, *protected*, *internal* sau *private*.

Proprietățile reprezintă modalități de expunere publică a câmpurilor private ale unei clase, putând, la nevoie, să ofere numai drept de citire (read only), numai drept de scriere (write only), sau și drept de citire și de scriere (read-write). Avantajul utilizării proprietăților este acela că ele permit scrierea de cod asociat operațiilor de atribuire și verificarea în acest fel a validității datelor transmise.

În exemplul de mai jos se defineşte clasa Persoana care are două câmpuri private şi două proprietăţi publice care accesează câmpurile private `nume` şi `varsta`. Proprietatea `NumePersoana` poate fi accesată atât în mod citire cât şi în mod scriere, pe când proprietatea `VarstaPersoana` poate fi accesată numai în mod citire.

```
class Persoana
{
    private string nume;
    private byte varsta;

    public string NumePersoana     //proprietate read-write
    {
        get { return nume; }
        set { nume=value; }
    }

    public byte VarstaPersoana     //proprietate read-only
    {
        get { return varsta; }
    }
}
```

Interfeţe

O interfaţă nu poate conţine cod executabil, numai semnături de metode, proprietăţi, evenimente sau indexatori. O clasă sau structură care implementează interfaţa trebuie să implementeze membrii specificaţi în definiţia interfeţei. Codul din fiecare proprietate şi metodă implementată poate să difere de la clasă la clasă, semantica fiecărei metode fiind păstrată. Faptul că fiecare clasă poate implementa aceeaşi proprietate sau metodă într-un mod diferit reprezintă baza pentru comportamentul polimorfic.

Puteţi defini o interfaţă în C# astfel:

```
interface Interfata
{
    string Denumire { get; set; }
    int Valoare { get; }
    void Init();
}
```

Iată cum ar arăta o clasă care implementează interfaţa definită mai sus:

```
class Clasa : Interfata
{
    string nume;
    int n;

    string Interfata.Denumire
    {
        get { return nume; }
        set { nume = value; }
    }
    int Interfata.Valoare
    {
        get { return n; }
    }
    void Interfata.Init()
    {
    }
}
```

Moștenire

În C#, o clasă poate moșteni numai o singură clasă de bază. Totuși, o clasă poate implementa mai multe interfețe. Pentru a putea observa modul de utilizare a moștenirii în C#, vom defini o clasă de bază:

```
class Persoana
{
    public string Nume;
    public string Prenume;

    //constructor cu parametri
    public Persoana(string nume, string prenume)
    {
        Nume = nume;
        Prenume = prenume;
    }

    public string NumeComplet()
    {
        return Nume + " " + Prenume;
    }
}
```

Pentru a deriva clasa Angajat din Persoana aveți nevoie să o declarați ca în exemplul de mai jos. Observați modul în care constructorul clasei derivate apelează explicit constructorul clasei de bază:

```
//clasa Angajat moștenește clasa Persoana
class Angajat : Persoana
{
    public  int salariu;
    public Angajat(int salar, string nume, string prenume)
        : base(nume, prenume)
    {
    }
}
```

Partea practică a laboratorului

În cadrul acestui laborator se va crea în C# o aplicație de tip Windows Forms Application a cărei interfață grafică o puteți vedea în Figura 1. Aplicația va permite:

- crearea unei structuri de clase și a unei liste generice de studenți,
- afișarea într-un control de tip ListBox a studenților din cadrul unei facultăți,
- adăugarea unui nou student,
- ordonarea studenților după nume,
- ordonarea studenților după medie,
- afișarea studenților dintr-un anumit an de studiu,
- căutarea unui student după nume,
- ștergerea studentului selectat în controlul de tip ListBox.

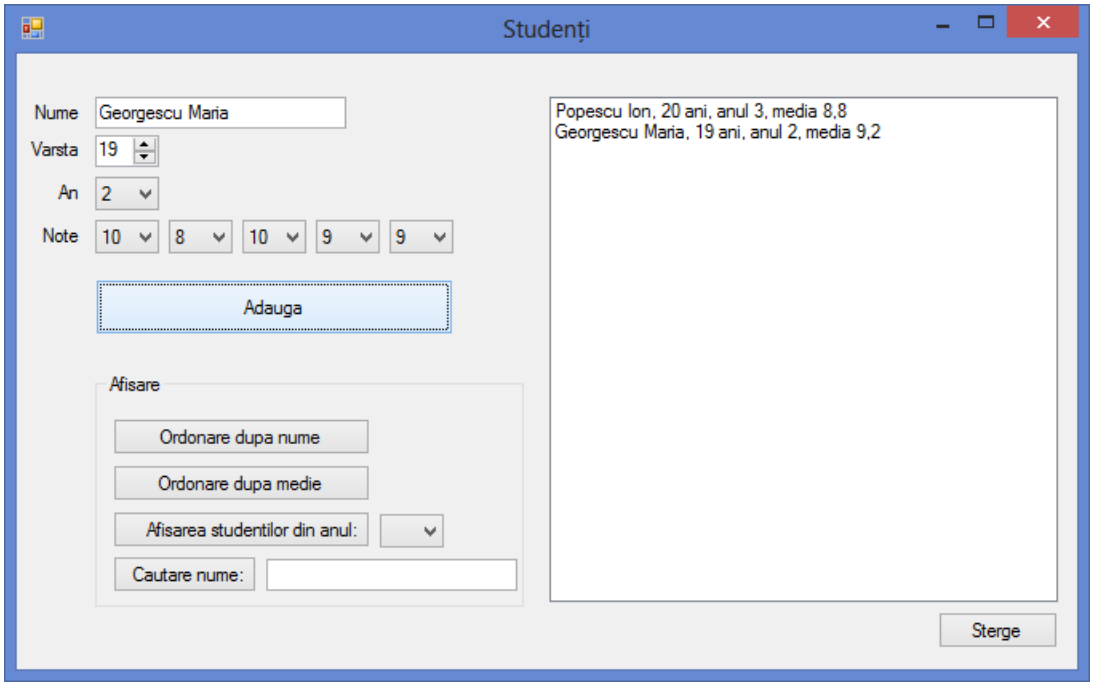

Figura 1. Fereastra aplicaţiei

Sfaturi utile

Nu pierdeţi timpul încercând să construiţi o interfaţă grafică extraordinar de spectaculoasă. Axaţi-vă în principal asupra aspectului funcţional al aplicaţiei. Încercaţi să implementaţi toate cerinţele aplicaţiei, apoi testaţi aplicaţia şi eliminaţi posibilele erori!

Mod de lucru

Iată paşii care trebuie urmaţi pentru dezvoltarea cu succes a aplicaţiei:

1. Se creează un proiect Visual C# nou de tip Windows Forms Application. Reţineţi care este directorul în care va fi salvat proiectul, iar dacă este nevoie modificaţi-l.
2. Se desenează interfaţa grafică a aplicaţiei prin adăugarea de controale pe fereastra prezentată în Figura 1. Controalele pot fi adăugate pe suprafaţa ferestrei numai prin selectarea lor în prealabil din panoul ToolBox situat în stânga ferestrei Visual Studio (vezi Figura 2).

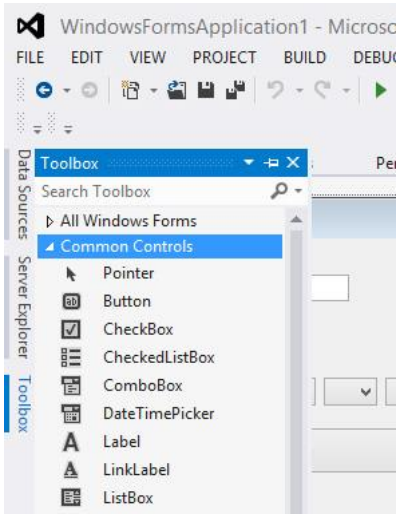

Figura 2. Panoul ToolBox.

Principalele tipuri de controale care apar pe suprafața ferestrei sunt:

- *TextBox*: pentru introducerea numelui studentului.
- *NumericUpDown*: pentru selectarea vârstei.
- *ComboBox*: pentru selectarea anului de studiu și a notelor studentului.
- *Label*: pentru etichetele text din stânga controalelor amintite mai sus.
- *Button*: pentru butoanele de adăugare a studenților în listă, ordonare, afișare și căutare a lor.
- *GroupBox*: pentru controlul care grupează butoanele. Acesta este un control de tip container (conține și grupează alte controale) și trebuie adăugat pe fereastră înaintea controalelor conținute.
- *ListBox*: pentru lista care va afișa datele studenților.

3. Se modifică proprietățile ferestrelor și controalelor din panoul Properties (implicit situat în partea dreapta-jos a ferestrei Visual Studio). De exemplu, pentru modificarea textului afișat pe suprafața controalelor se modifică proprietatea `Text`. Proprietatea `Name` se referă la numele controlului cu care acesta poate fi referit prin cod. Pentru popularea controalelor ComboBox cu valori se folosește proprietatea `Items`.

Observație:

- ✓ Pe parcursul dezvoltării aplicației veți avea nevoie să folosiți atât designerul de interfețe grafice cât și editorul de cod. Pentru a comuta între ele puteți accesa comenzile de meniu View -> Code și View -> Designer.

4. Se definesc două clase: clasa Persoana (clasă de bază) și clasa Student (clasă derivată).
   - Clasa Persoana va avea ca membri:
     - Variabile protected: `string nume`, `byte varsta`
     - Un constructor cu parametri pentru inițializarea câmpurilor clasei.
   - Clasa Student va avea ca membri:
     - Variabile private: `byte an`, `byte[] note` (vector de 5 valori byte). Vectorul note se va declara în felul următor:

```
byte[] note = new byte[5];
```

     - Proprietăți publice read-only: `byte AnStudiu`, `string NumeStudent`
     - Metode publice: `float Medie()`, `string AfisareStudent()`
     - Un constructor cu parametri pentru inițializarea câmpurilor clasei. Atenție la modul în care acesta va apela explicit constructorul clasei de bază!

Observații:

- ✓ Proprietățile read-only `NumeStudent` și `AnStudiu` din clasa Student, dau acces în mod citire la datele studentului, care altfel nu ar fi fost accesibile din afara clasei.
- ✓ Cele două clase pot fi create fie în fișierul Form1.cs în care este implementată clasa Form1 (adăugând la namespace cele două clase noi), fie în fișiere distincte, prin comanda de meniu Project -> Add Class.
- ✓ Observați modul în care este declarată clasa Form1. Ea este declarată ca parțială deoarece Visual Studio .NET va plasa codul aferent interfeței grafice într-un fișier separat: Form1.Designer.cs, care devine în timpul compilării parte a aceleiași clase. Acest fișier conține instrucțiuni pentru desenarea ferestrei și a tuturor controalelor componente.

5. Se implementează o listă generică de obiecte de tip Student. O listă generică reprezintă o listă de obiecte care pot fi accesate prin index. Listele generice oferă metode pentru căutare, sortare și manipulare a elementelor din listă.

Exemplu de utilizare:

```
//declarare lista generica
private List<Student> lista = new List<Student>();

...

Student s = new Student(/*lista parametri*/);
lista.Add(s);  //adaugarea unui element in lista

...

lista[0].AfisareStudent();  //accesarea metodelor unui element din lista
```

Toate operațiile asupra studenților se vor efectua întâi asupra listei generice (adăugare, ștergere, sortare) și de abia după aceea studenții din listă vor fi afișați în controlul ListBox (preferabil, într-o buclă foreach):

```
foreach(Student s in lista)
    lstStudenti.Items.Add(s.AfisareStudent());
```

Datele preluate din controalele de tip ComboBox sunt de tip string, astfel că va fi nevoie de conversia lor explicită la un tip de date numeric. Iată cum va arăta, de exemplu, apelul constructorului clasei Student:

```
Student s = new Student(Convert.ToByte(cmbAn.Text), n, txtNume.Text,
                        (byte)numVarsta.Value);
```

6. Se vor prevedea modalități de verificare a validității datelor introduse de la tastatură (cazul variabilelor numerice), prin utilizarea blocurilor `try…catch`.

Exemplu de utilizare:

```
try
{
    //instructiunile care pot provoca aparitia unei erori
}
catch (Exception ex)
{
    //Tratarea erorii si eventual afisarea mesajului de eroare:
    MessageBox.Show(ex.Message);
}
```

7. Sortarea studenților se va face prin metoda `sort` a listei generice. Pentru fiecare tip de sortare (după nume și după medie) trebuie definită câte o clasă care să implementeze interfața IComparer și să supraîncarce metoda `Compare` pentru a-i da sensul dorit. Interfața IComparer are un singur membru: metoda publică `Compare`, care primește doi parametri de tipul obiectelor ce trebuie sortate și va returna -1, 0 sau 1 în funcție de rezultatul comparării celor doi membri (mai mic, egal, mai mare).

```
class ComparaMedie : IComparer<Student>
{
    public int Compare(Student x, Student y)
    {
        if (x.Medie() > y.Medie())
            return 1;
        else if (x.Medie() == y.Medie())
            return 0;
        else
            return -1;
    }
}
```

Cu ce ne-am ales?

Prin aplicația dezvoltată în cadrul laboratorului de astăzi am reușit să ne familiarizăm pe de-o parte cu stilul de programare Visual C# .NET, iar pe de altă parte am luat contact cu o parte din facilitățile POO oferite de acest limbaj.

Bibliografie

C# Reference: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index