Programarea Aplicațiilor Windows – curs 6

Conf. dr. Cristian CIUREA
Departamentul de Informatică și Cibernetică Economică
Academia de Studii Economice București
cristian.ciurea@ie.ase.ro

Agenda

1. Lucrul cu meniuri
2. Validarea datelor
3. Gestiunea excepțiilor

Meniuri

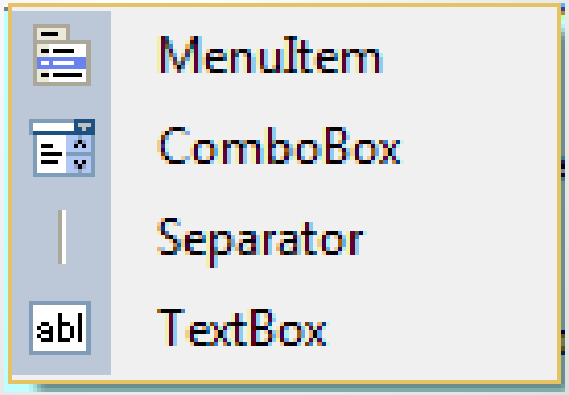
În C# există două tipuri de meniuri:

- **principale**, inserabile prin controlul MenuStrip (sau MainMenu în funcție de versiunea de Visual Studio);
- **contextuale**, inserabile prin controlul ContextMenuStrip (sau ContextMenu în funcție de versiunea de Visual Studio).

Meniuri

Elementele componente ale unui meniu (principal sau contextual) sunt:

- MenuItem
- ComboBox
- Separator
- TextBox



Meniuri

- Meniurile contextuale se activează după execuția programului, la click dreapta pe mouse.
- Meniurile contextuale pot fi atașate unuia sau mai multor controale.
- În general, un meniu contextual se atașeaza simultan mai multor controale pentru ca fiecare control să beneficieze de opțiunile oferite de meniu.

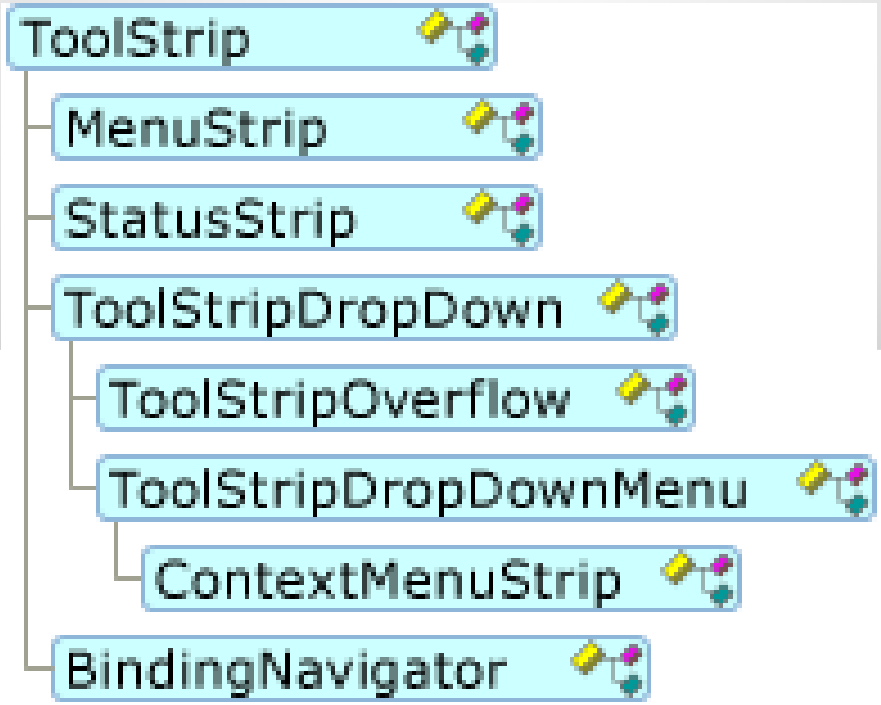
Meniuri

Atașarea meniului la un control se face prin punerea numelui meniului contextual dorit în proprietatea "ContextMenuStrip" a controlului la care se dorește atașarea meniului.

În cazul atașării meniului la mai multe controale, se verifică proprietatea "SourceControl" pentru a identifica controlul care a activat meniul.

Meniuri

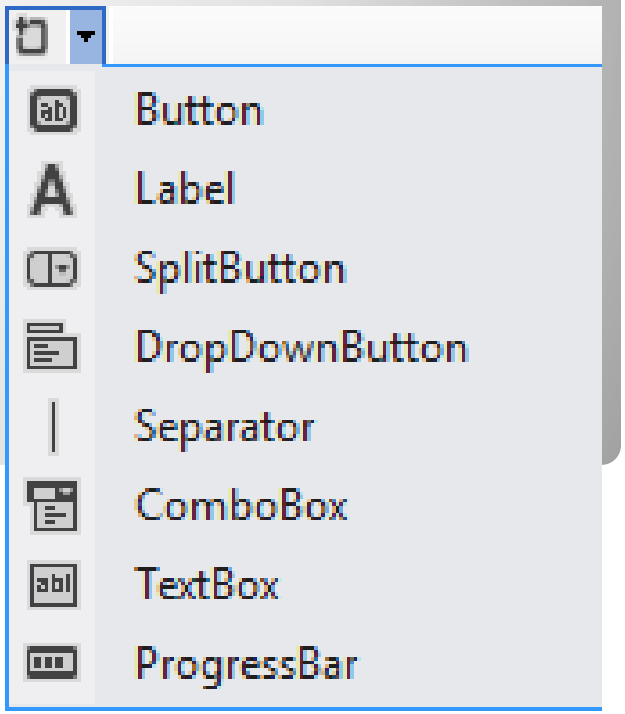
ToolStrip este clasa de bază abstractă pentru clasele MenuStrip, StatusStrip și ContextMenuStrip.



Meniuri

În cadrul unui control ToolStrip se pot adăuga următoarele componente:

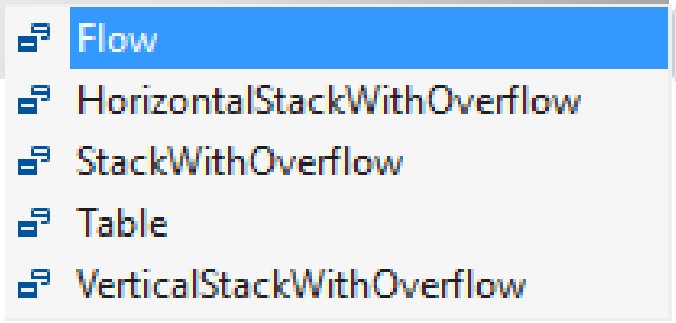
- ToolStripButton
- ToolStripSeparator
- ToolStripLabel
- ToolStripDropDownButton
- ToolStripSplitButton
- ToolStripTextBox
- ToolStripComboBox



Meniuri

Controlul StatusStrip înlocuiește controlul StatusBar și aduce în plus opțiuni de design, precum și proprietatea "Spring" care permite unui ToolStripStatusLabel să ocupe tot spațiul disponibil.

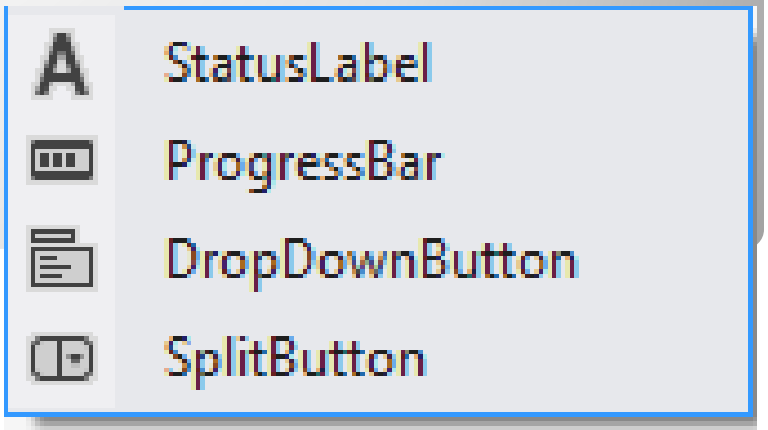
```
statusStrip1.LayoutStyle = ToolStripLayoutStyle.Table;
```



Meniuri

În cadrul unui control StatusStrip se pot adăuga următoarele componente:

- ToolStripStatusLabel
- ToolStripDropDownButton
- ToolStripSplitButton
- ToolStripProgressBar



Validarea datelor

Validarea datelor este:

- **simplă**, prin punerea pe true a proprietății CausesValidation a unui control se solicită ca în momentul sosirii pe control să se declanșeze funcția de validare asociată controlului respectiv.
- **încrucișată**, folosită în cazul validării simultane a mai multor controale, în situația în care fiecare control cere și așteaptă să fie validat controlul precedent.

Validarea datelor

Validarea simplă presupune execuția funcțiilor declarate pentru tratarea evenimentelor **Validating** și **Validated**.

Evenimentul Validating se produce la încercarea de părăsire a unui control pentru a trece pe un alt control care are proprietatea CausesValidation pe true.

Validarea datelor

Evenimentul Validated se declanșează după Validating, dar nu înainte de părăsirea controlului și numai dacă nu a fost revocat evenimentul Validating.

Validarea simplă este utilă doar pentru testarea izolată a unui singur control.

Validarea datelor

În cazul validării încrucișate, toate controalele și formularul trebuie să aibă proprietatea CausesValidation pe false pentru a nu declanșa validări individuale.

Gestiunea validării mai multor controale simultan se realizează cu ajutorul unui control de tip ErrorProvider.

Gestiunea excepțiilor

Gestiunea situațiilor neprevăzute care apar în momentul execuției unui program este implementată în C# prin intermediul excepțiilor.

Gestiunea excepțiilor

- Excepție – situație în care prelucrarea anumitor date de intrare nu este gestionată sau nu este posibilă (ex: împărțire la 0, citire în afara unui masiv)
- Permite gestiunea situațiilor excepționale care conduc la terminarea imediată a programului
- Necesară pentru a realiza programe robuste și fiabile
- Implementată prin **try**, **catch** și **throw**
- Permite gestiunea erorilor de sistem si a erorilor definite de programator

Gestiunea excepțiilor

Din punctul de vedere al bibliotecii de clase, erorile sunt împărțite, pe cel mai înalt nivel, în două categorii:

- **excepții de aplicație** (Application Exception), generate de aplicațiile utilizator;
- **excepții de sistem** (System Exception), generate de către mașina virtuală CLR.

Ambele sunt derivate din clasa generică Exception.

Gestiunea excepțiilor

Cea mai simplă modalitate de a gestiona o eroare constă în gruparea instrucțiunilor suspectate că ar produce respectiva eroare într-un bloc “try”. Tratarea acestei erori se face într-un bloc “catch”, asociat blocului “try”.

Exceptiile necesită tratamente diferite în funcție de tipul lor. Este important să identificăm tipul erorii și să oferim mai multe blocuri “catch”, conținând cod de tratare distinct pentru fiecare tip de eroare în parte.

Gestiunea excepțiilor

```
try
        { //secventa prelucrari }
catch(exceptie_tip_1)
        { //secventa prelucrari specifice }
catch(exceptie_tip_2)
        { //secventa prelucrari specifice }
catch(Exception)
        { //secventa prelucrari generale }
finally
        { //secventa prelucrari obligatorii }
```

Gestiunea excepțiilor

blocul **try{...}**

- contine secvența de prelucrări care generează excepții;
- are asociat minim un bloc “catch”;
- între blocul “try” și blocurile “catch” asociate nu există alte instrucțiuni;
- un bloc “try” trebuie să aibă asociat cel puțin un bloc “catch” sau măcar un bloc “finally”.

Gestiunea excepțiilor

blocul **catch(tip_exceptie exc)**

- gestionează o excepție de tipul anunțat;
- tip_exceptie reprezintă instanța unei clase derivate din Exception (ArithmeticException, DivideByZeroException, etc.)

blocul **catch(Exception exc)**

- gestionează toate tipurile de excepții

Gestiunea excepțiilor

blocul **finally{...}**

- conține secvența de prelucrări care se execută indiferent dacă blocul "try" a generat sau nu excepții și dacă acestea au fost sau nu tratate în blocurile "catch"
- blocul "finally" este unic, executându-se întotdeauna, indiferent care din blocurile "try" sau "catch" s-a executat. El urmărește aducerea la o stare coerentă și consistentă a aplicației.

Gestiunea excepțiilor

Blocurile "catch" sunt definite în ordine crescătoare a generalității excepțiilor tratate

```
try { … }
catch(exceptie_tip_1) {…}
catch(exceptie_tip_2) {…}
…
catch(Exception e) {…}
```

Gestiunea excepțiilor

Rezolvarea unei excepții se realizează prin căutarea unui bloc "catch" specific erorii produse și, numai dacă acesta nu există, se caută blocuri "catch" asociate excepțiilor plasate din ce în ce mai sus în ierarhia de derivare.

Se recomandă plasarea blocurilor "catch" în ordinea de la erori specifice către erori din ce în ce mai generale.

Gestiunea excepțiilor

Instrucțiunea “throw” este folosită pentru a declanșa în mod explicit o anumită excepție definită de programator.

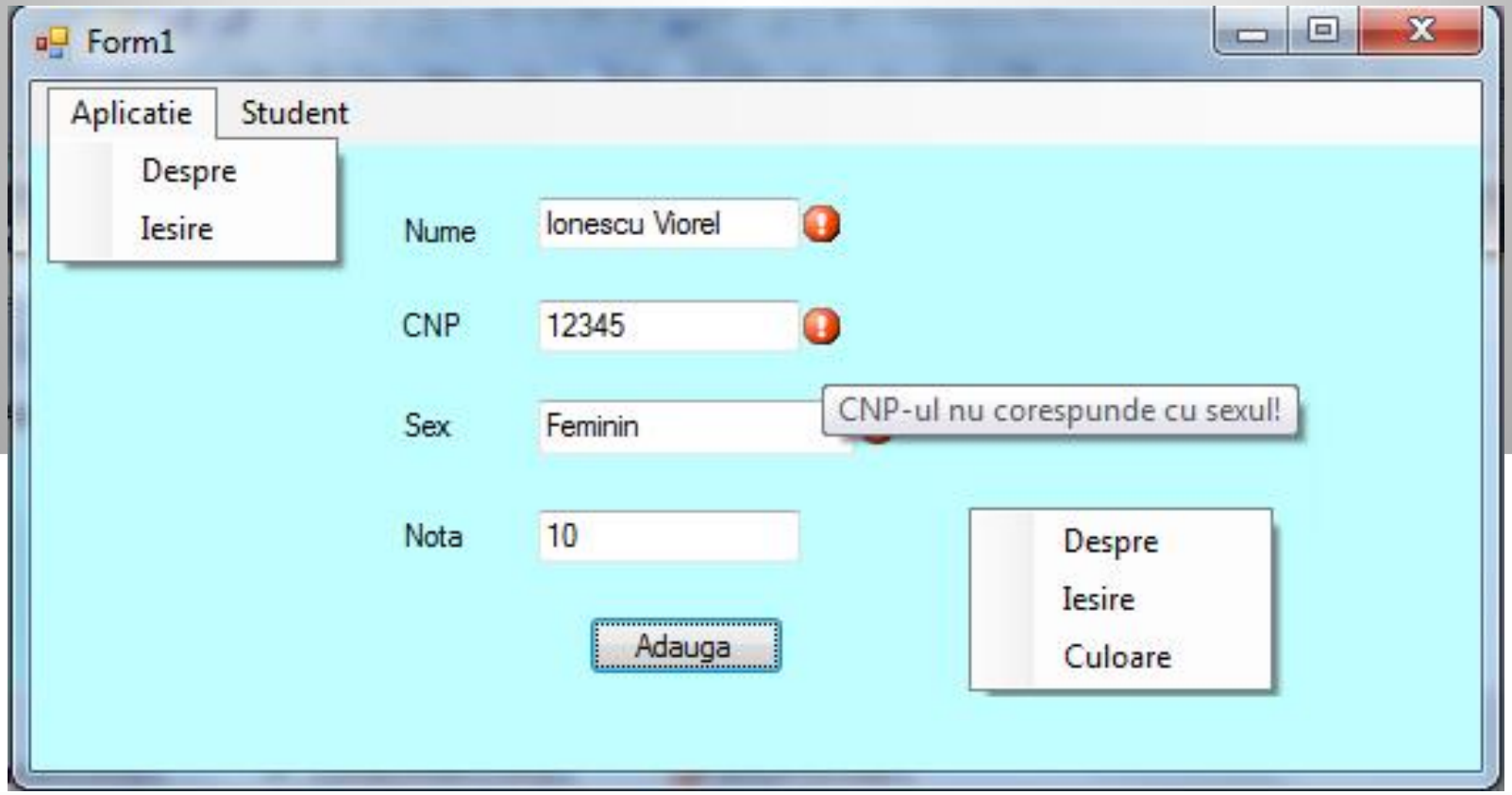
Nu este indicat un apel “throw” din interiorul unui bloc “finally”, deoarece în acel moment există deja încă o excepție care așteaptă să fie tratată.

Gestiunea excepțiilor

- Blocurile try-catch-finally pot fi incluse în alte blocuri try
- Programatorul poate defini propriile excepții prin clase derivate din Exception
- Instrucțiunea throw generează orice tip de excepție

Gestiunea excepțiilor

Aplicație preluare informații studenți – meniuri și validări



Bibliografie

[1] I. Smeureanu, M. Dârdală, A. Reveiu – *Visual C# .NET*, Editura CISON, București, 2004.

[2] C. Petzold – *Programming Microsoft Windows with C#*, Microsoft Press, 2002.

[3] L. O'Brien, B. Eckel – *Thinking in C#*, Prentice Hall.

[4] J. Richter – *Applied Microsoft .NET Framework Programming*, Microsoft Press, 2002.

[5] http://acs.ase.ro/paw