# Programarea Aplicațiilor Windows – curs 3

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

# Agenda

1. Conceptul de delegate
2. Evenimente
3. Introducere în Windows Forms
4. Exemplificare aplicație Windows

# Delegate

**Delegate** este tipul asociat unui obiect care încapsulează o referință la o metodă și este echivalentul conceptului de pointer de funcție în C++.

# Delegate

DELEGATE:

- un nou tip de referință în C#

```
delegate tip_return nume_referinta(lista parametri);
```

```
public delegate void FireAlarmDelegate(object sender, EventArgs e);
```

- echivalent pointerului la funcție din C++

```
tip_return (*nume_pointer) (lista parametri);
```

- facilitează definirea pointerilor la functii

```
nume_referinta pFunctie;
```

# Delegate

Referința poate fi modificată dinamic, delegatul permițând apelul mai multor implementări ale unor metode care au același prototip.

```
private void btn1_Click(object sender, EventArgs e)
{
      MessageBox.Show("S-a facut click pe buton!");
}
```

```
private void MetodaMea(object sender, EventArgs e)
{
      MessageBox.Show("Click prin MetodaMea()");
}
```

# Delegate

Delegatul permite legarea întârziată (**late binding**), adică amânarea precizării funcției de apelat până la momentul execuției.

```
btn1.Click += new EventHandler(btn1_Click);

btn1.Click -= new EventHandler(btn1_Click);

btn1.Click += new EventHandler(MetodaMea);
```

# Evenimente

O clasă eveniment înglobează:

- o instanță de delegate, căreia i se pune în față "event";

```
public event FireAlarmDelegate FireAlarmEvent;
```

- o funcție care activează delegatul, adică apelează o listă de funcții.

```
public void SunaAlarma(EventArgs e)
{
      FireAlarmEvent(this, e);
}
```

# Evenimente

EVENIMENT:

- reprezintă un vector/listă de delegați (pointeri de funcții)

```
event tip_delegate nume_event;
```

- facilitează execuția tuturor funcțiilor printr-un singur apel

```
nume_event([parametrii]);
```

- suportă operatorii aritmetici **+=** și **-=** pentru adăugare/ștergere de delegați din eveniment

```
nume_event += new TipDelegate(metoda);
```

# Evenimente

Fără folosirea delegatului, evenimentul ar fi fost legat de o clasă și de numele unei singure metode pe care trebuie să o apeleze.

Prin utilizarea delegatului, mai multe clase pot subscrie la același eveniment și pot folosi metode denumite diferit, deoarece apelul lor se realizează prin pointer/referință.

# Evenimente

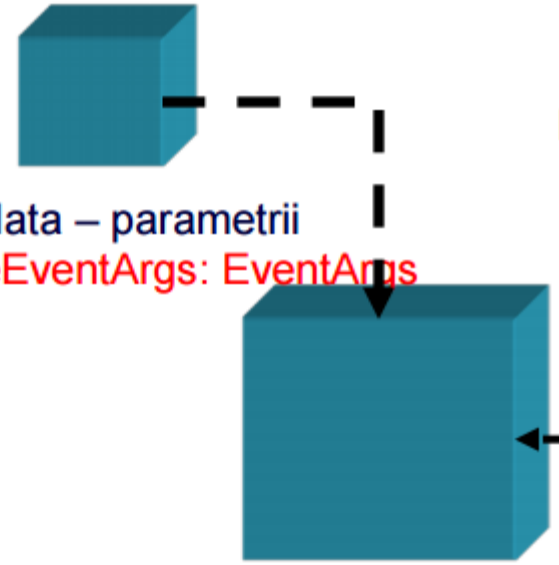
- un eveniment reprezintă un mesaj trimis de un obiect pentru a anunța o acțiune (*user interaction* - mouse click, button click sau *program logic* – funcție din program).
- în modelul de tratare a evenimentului din .NET, obiectul care lansează evenimentul nu știe ce obiect sau ce metodă va primi și va gestiona (handle); din acest motiv este nevoie de un element intermediar între sursă și destinație – *delegat* (pointer la funcție).

# Evenimente

Pentru a lansa un eveniment este nevoie de 3 componente:

- obiectul ce lansează event-ul;
- tip delegat ce definește event-ul;
- obiectul ce definește conținutul mesajului.

# Evenimente



event data – parametrii
Ex: class FireEventArgs: EventArgs

delegat – defineste evenimentul
Ex: public delegate void FireEventHandler(object sender, FireEventArgs e);

event sender – lanseaza evenimentul
Ex: class FireAlarm{ }

# Evenimente

Tratarea unui eveniment se poate face:

- prin atașarea unei metode la delegatul specific evenimentului:

```
MouseDown += new
MouseEventHandler(MetodaMea);
```

- prin suprascrierea unei metode protected moștenită din clasa de bază:

```
protected override void
OnMouseDown(MouseEventArgs e)
{
      base.OnMouseDown(e);
      //...
}
```

# Evenimente

Pentru a oferi soluții personalizate de tratare a evenimentului se definește o clasă de argumente derivată din `EventArgs`:

```
public class FireAlarmEventArgs : EventArgs
{ … }
```

# Evenimente

Funcția care activează delegatul devine:

```
public void SunaAlarma(FireAlarmEventArgs e)
{
      FireAlarmEvent(this, e);
}
```

# Evenimente

Metoda pentru abonare la eveniment din clasa care oferă soluții pentru evenimentul `FireAlarm` devine:

```
public void DoSomething(object sender, FireAlarmEventArgs e)
{
        if (e.risc < 5)
        {
                Console.WriteLine("Alarma incendiu in camera " + e.camera + " Contactati telefonic camera!" + "Posibil tigara aprinsa!");
        }
        else Console.WriteLine("Alarma incendiu in camera " + e.camera + " Contactati departamentul pompieri. " + "Evacuati cladirea!");
}
```

# Introducere în Windows Forms

Sub biblioteca de clase Microsoft Foundation Classes se întâlnesc aplicații Windows cu următoarele tipuri de vizualizări:

- **Forms** (formular), specializată pentru introducere date într-o machetă;
- **TreeView** (arborescentă), specializată în afișarea structurilor expandabile de date;
- **EditView** (editare), specializată în lucrul cu texte;
- **View** (vizualizare generică), specializată în reprezentări grafice.

# Introducere în Windows Forms

Atât formularul, cât și controalele aferente au o serie de **proprietăți** și **evenimente**.
Cele mai importante proprietăți ale obiectului **Form** sunt:

- Name,
- Text,
- BackgroundImage,
- BackColor,
- Font.

# Introducere în Windows Forms

Cele mai importante evenimente ale obiectului **Form** sunt:

- Load,
- Paint,
- MouseDown,
- MouseUp,
- MouseMove,
- DragEnter,
- DragDrop,
- KeyPress.

# Introducere în Windows Forms

Clasa **Form**, fiind derivată din System.Windows.Forms.ContainerControl, se comportă ca un container care poate susține diferite controale (TextBox, Label, Button, etc.).

# Introducere în Windows Forms

Pentru adăugarea unui control pe formular:

- declararea în clasa Form a unei referințe:

```
private Button btn;
```

- instanțierea controlului:

```
btn = new Button();
btn.Text = "Click aici";
btn.Size = new Size(60,40);
btn.Location = new Point(80,100);
```

# Introducere în Windows Forms

Pentru adăugarea unui control pe formular:

- adăugarea controlului la colecția de controale a formularului:

```
Controls.Add(btn);
```

- atașare metode tratare evenimente:

```
btn.Click += new EventHandler(btn_Click);
```

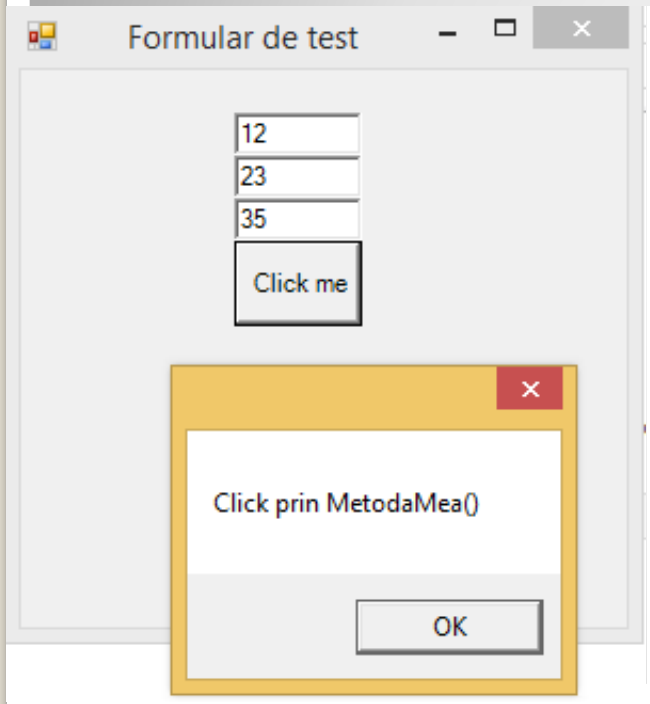
# Introducere în Windows Forms

Compilare linie de comandă:

```
csc /t:winexe form1.cs

csc /t:winexe /r:System.dll
/r:System.Windows.Forms.dll
/r:System.Drawing.dll form1.cs
```

# Introducere în Windows Forms




```
private void btn1_Click(object sender, EventArgs e)
{
    MessageBox.Show("S-a facut click pe buton!");
}

private void MetodaMea(object sender, EventArgs e)
{
    btn1.Click -= new EventHandler(btn1_Click);
    MessageBox.Show("Click prin MetodaMea()");
}

private void Suma(object sender, EventArgs e)
{
    tb3.Text = (Convert.ToInt32(tb1.Text)+
    Convert.ToInt32(tb2.Text)).ToString();
}
```

# 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