



# Modulul 11


## Operatori, delegați și evenimente

Acest modul acoperă trei arii de funcționalitate ce pot fi implementate în clase: operatori, delegați și evenimente.

Operatorii sunt componente de bază ale unui limbaj. Aceștia sunt folosiți pentru a executa operații de manipulare și comparație între variabile.

Delegații specifică contactul dintre un obiect ce apelează o funcție și un obiect ce implementează funcția apelată.

Evenimentele oferă un mecanism prin care o clasă poate notifica clienții săi când o schimbare apare în starea sa.




itacad  
you@technology

Microsoft®  
.NET


## Overview

- ▶ Introducere a operatorilor
- ▶ Supraîncărcarea operatorilor
- ▶ Crearea și folosirea delegaților
- ▶ Definirea și utilizarea evenimentelor



2

- La finalul acestui modul, veți putea să:
- Definiți operatori, pentru a ușura interacțiunea cu structuri sau clase.
  - Folosirea delegaților pentru a decupla apelul metodei de implementarea sa
  - Specificarea de evenimente pentru a permite claselor subscrise să fie notificate de schimbările în starea obiectelor.

 itacad  
you@technology

## Introducere a operatorilor

- ▶ Operatori și metode
- ▶ Operatori predefiniți în C#

Microsoft®  
.NET

3

Operatorii sunt diferiți de metode. Aceștia au cerințe speciale pentru a putea să funcționeze după cum este așteptat. C# are o serie de operatori predefiniți ce pot fi folosiți pentru a manipula tipurile și clasele oferite de către platforma Microsoft .NET

La finalul acestei lecții, veți putea să:

- Identificați de ce C#, ca și celelalte limbaje, are definiți operatori
- Definiți operatori, pentru a ușura lucrul cu structuri și clase

## Operatori și metode

### ► Folosirea metodelor

- Reduce claritatea
- Crește riscul apariției erorilor, atât sintactice, cât și semantice

```
myIntVar1 = Int.Add(myIntVar2,
                    Int.Add(Int.Add(myIntVar3,
                                    myIntVar4), 33));
```

### ► Folosirea operatorilor

- Face expresiile mult mai clare

```
myIntVar1 = myIntVar2 + myIntVar3 + myIntVar4 + 33;
```

4

Scopul operatorilor este de a face expresiile mai clare și mai ușor de înțeles. Ar fi posibil să existe un limbaj de programare fără operatori, ce se bazează pe metode bine definite, dar cu siguranță ar fi un limbaj foarte greoi și neclar.

### Folosirea metodelor

De exemplu, presupunem că operatorul matematic de adunare nu ar fi prezent, și limbajul oferă, în schimb, metoda **Add** a clasei **Int**, ce primește 2 parametri și întoarce rezultatul. Atunci, pentru a aduna două variabile, va trebui să scrieți cod de forma:

```
myIntVar1 = Int.Add(myIntVar2, myIntVar3);
myIntVar2 = Int.Add(myIntVar2, 1);
```

### Folosirea operatorilor

Prin folosirea operatorului matematic de adunare, se pot scrie linii de cod mult mai concise:

```
myIntVar1 = myIntVar2 + myIntVar3;
myIntVar2 = myIntVar2 + 1;
```

Codul ar putea să devină mult mai greu descifrabil dacă ar trebui adunate mai multe valori folosind metoda **Add**:

```
myIntVar1 = Int.Add(myIntVar2, Int.Add(Int.Add(myIntVar3, myIntVar4), 33));
```

De asemenea, riscul apariției erorilor crește foarte mult odată cu complexitatea codului. Deși operatorii sunt implementați ca metode de către C#, sintaxa lor îi face foarte ușor de folosit. Compilatorul C# transformă automat expresiile ce conțin operatori în seria corectă de apeluri de metode.

## Operatori predefiniți în C#

### Categorii de operatori

Aritmetic	Acces al membrilor
Logic	Indexare
Concatenare de stringuri	Cast
Shiftare	Condițional
Relaționali	Delegați
Atribuire	Creare de obiecte
Incrementare&decrementare	Informații asupra tipului
Overflow control	Adrese de memorie

5

Limbajul C# oferă un set extins de operatori. În continuare este o listă completă:

Categoria operatorului	Operatori
Aritmetic	<code>+, -, *, /, %</code>
Logic	<code>&amp;,  , ^, !, ~, &amp;&amp;,   , true, false</code>
Concatenare de stringuri	<code>+</code>
Shiftare	<code>&lt;&lt;, &gt;&gt;</code>
Relaționali	<code>==, !=, &lt;, &gt;, &lt;=, &gt;=</code>
Atribuire	<code>=, +=, -=, *=, /=, %=, &amp;=,  =, &lt;&lt;=, &gt;&gt;=</code>
Incrementare&decrementare	<code>++, --</code>
Overflow control	<code>checked, unchecked</code>
Acces al membrilor	<code>.</code>
Indexare	<code>[ ]</code>
Cast	<code>()</code>
Condițional	<code>?:</code>
Delegați	<code>+, -</code>
Creare de obiecte	<code>new</code>
Informații asupra tipului	<code>is, sizeof, typeof</code>
Adrese de memorie	<code>*, -, [ ], &amp;</code>

Operatorii sunt folosiți pentru construirea expresiilor. Scopul unei mari părți din operatori este ușor de înțeles. De exemplu, operatorul de adunare (+) în expresia  $10 + 5$  va executa suma aritmetică, returnând 15.

Alte operații posibil să nu fie la fel de intuitive ca acestea, pe când altele sunt definite prin cuvinte cheie, nu simboluri, dar funcționalitatea lor cu tipurile de date și clasele oferite de platforma .NET este complet definită.

### **Operatori cu definiții multiple**

Un aspect ce poate induce în eroare programatorul este folosirea aceluiași simbol pentru a executa operații diferite. De exemplu, operatorul + în expresia  $10+5$  este clar un operator matematic. Se poate determina sensul operatorului doar din contextul în care acesta este folosit.

De exemplu, în contextul:

“Hello” + “world”

se observa că sensul operatorului este de a concatena stringuri.


itacad  
you@technology

Microsoft®  
.NET

## Supraîncărcarea operatorilor

- ▶ Introducere asupra supraîncărcării operatorilor
- ▶ Supraîncărcarea operatorilor relaționali
- ▶ Supraîncărcarea operatorilor logici
- ▶ Supraîncărcarea operatorului de conversie
- ▶ Supraîncărcarea multiplă a operatorilor

6

Mulți operatori C# execută funcții bine definite pe clase și alte tipuri de date. Totuși, unii operatori oferți de C# pot fi redefiniți de programator, pentru a putea fi aplicați asupra claselor și structurilor definite de acesta. Într-un fel, se poate spune că programatorul definește proprii săi operatori. Acest proces se numește supraîncărcarea operatorilor.

Nu toți operatorii C# predefiniți se pot supraîncărca. Operatorii matematici unari, binari, operatorii logici pot fi supraîncărcați fără nici o problemă. Operatorii de atribuire, însă, nu pot fi supraîncărcați direct. Ei însă se bazează pe evaluarea operatorilor aritmetici, logici și de shiftare, care pot fi supraîncărcați.

La finalul acestei lecții, veți putea:

- să supraîncărcați operatori logici, relaționali și de conversie
- să supraîncărcați un operator de mai multe ori

## Introducere asupra supraîncărcării operatorilor

- ▶ Supraîncărcarea operatorilor
  - ▶ Definiți proprii operatori doar când este nevoie
- ▶ Sintaxa operatorilor
  - ▶ operator **op**, unde **op** este operatorul ce este supraîncărcat
- ▶ Exemplu:

```
public static Time operator+(Time t1, Time t2)
{
    int newHours = t1.hours + t2.hours;
    int newMinutes = t1.minutes + t2.minutes;
    return new Time(newHours, newMinutes);
}
```

7

Deși operatorii simplifică operațiile, trebuie definiți doar când are sens să faceți acest lucru. Operatorii ar trebui supraîncărcați când clasa sau structura reprezintă o informație asemănătoare din punct logic, de exemplu, unui număr, și va fi folosită în acest fel. Un operator trebuie să fie foarte clar în folosire. Nu ar trebui să existe mai multe posibile interpretări asupra ce înseamnă.

De exemplu, nu este recomandată supraîncărcarea operatorului (++) pentru o clasă **Angajat** deoarece sensul acestei operații nu este foarte clar. Ce înseamnă “incrementarea unui angajat”? Dacă înseamnă promovarea angajatului, este mult mai bine să se definească o metodă **Angajat.Promovare()** ce face acest lucru.

### Sintaxa pentru supraîncărcarea operatorilor

Toți operatorii sunt metode publice, statice ale căror nume are o formă specială. Toți operatorii se numesc operator**op**, unde **op** specifică ce operator este supraîncărcat. De exemplu, metoda pentru supraîncărcarea operatorului de adunare este **operator+**.

Parametrii pe care operatorul îi primește și tipul pe care aceasta îl întoarce trebuie să fie definite. Toți operatorii aritmetici întorc o instanță a clasei și manipulează obiecte de tipul clasei.

### Exemplu

În continuare, considerăm structura **Time**. O valoare de tip **Time** este alcătuită din două numere întregi: unul reprezentând orele, cel de-al doilea minutele. Codul următor demonstrează implementarea operatorului + pentru a aduna două instanțe ale clasei **Time**



```
public struct Time
```

```
{
```

```
    public Time(int minutes) : this(0, minutes)
```

```
    {
```

```
    }
```

```
    public Time(int hours, int minutes)
```

```
    {
```

```
        this.hours = hours;
```

```
        this.minutes = minutes;
```

```
        Normalize( );
```

```
    }
```

```
    // Arithmetic
```

```
    public static Time operator+(Time lhs, Time rhs)
```

```
    {
```

```
        return new Time(lhs.hours + rhs.hours, lhs.minutes + rhs.minutes);
```

```
    }
```

```
    public static Time operator-(Time lhs, Time rhs)
```

```
    {
```

```
        ...
```

```
    }
```

```
    ...
```

```
    // Helper methods
```

```
    private void Normalize( )
```

```
    {
```

```
        if (hours < 0 || minutes < 0)
```

```
        {
```

```
            throw new ArgumentException("Time too small");
```

```
        }
```

```
        hours += (minutes / 60);
```

```
        minutes %= 60;
```

```
    }
```

```
    private int TotalMinutes( )
```

```
    {
```

```
        return hours * 60 + minutes;
```

```
    }
```

```
    private int hours;
```

```
    private int minutes;
```

```
}
```


itacad  
you@technology

Microsoft  
.NET

## Supraîncărcarea operatorilor relaționali

- ▶ Operatorii relaționali trebuie supraîncărcați în perechi
  - ▶ < și >
  - ▶ <= și >=
  - ▶ == și !=
- ▶ Suprascrieți metoda **Equals** dacă se supraîncarcă operatorii == și !=
- ▶ Suprascrieți metoda **GetHashCode** dacă se suprascrie metoda **Equals**

8

Operatorii relaționali trebuie supraîncărcați în perechi. Fiecare operator operațional trebuie să fie definit împreună cu antonimul său logic. Asta înseamnă că, dacă se supraîncarcă <, trebuie supraîncărcat și > și vice-versa. Similar, != trebuie să fie supraîncărcat împreună cu == și <= cu >=.

**Obs.** Se recomandă, din motive de consecvență, să se creeze o metodă **Compare** întâi și să definiți operatorii plecând de la această metodă.

### Suprascrierea metodei Equals

Dacă se supraîncarcă operatorii == și !=, ar trebui să suprascrieți, de asemenea, metoda **Equals** pe care clasa Dvs. o moștenește de la clasa Object. Aceasta pentru a asigura consecvență atunci când două obiecte ale clasei sunt comparate, fie folosind ==, fie folosind **Equals**. Astfel, se evită situații în care == întoarce un rezultat și Equals altul.

### Suprascrierea metodei GetHashCode

Metoda GetHashCode (de asemenea moștenită de la Object) este folosită pentru a identifica o instanță a clasei dacă este stocată într-un HashTable. Două instanțe ale aceleiași clase pentru care metoda **Equals** întoarce **true** ar trebui să aibă aceeași valoare întregă ca hash. Implicit, acest lucru nu se întâmplă. De aceea, dacă suprascrieți metoda **Equals**, va trebui, de asemenea, să suprascrieți metoda **GetHashCode**.

```
public struct Time
{
    ...
    // Equality
    public static bool operator==(Time lhs, Time rhs)
    {
        return lhs.Compare(rhs) == 0;
    }
    public static bool operator!=(Time lhs, Time rhs)
    {
        return lhs.Compare(rhs) != 0;
    }
    // Relational
    public static bool operator<(Time lhs, Time rhs)
    {
        return lhs.Compare(rhs) < 0;
    }
    public static bool operator>(Time lhs, Time rhs)
    {
        return lhs.Compare(rhs) > 0;
    }
}
```


itacad  
you@technology

Microsoft®  
.NET

## Supraîncărcarea operatorilor logici

- ▶ Operatorii **&&** și **||** nu pot fi supraîncărcați direct
  - ▶ Ei sunt evaluați în funcție de operatorii **&**, **|**, **true** și **false**, care pot fi supraîncărcați
  - ▶ **x && y** este evaluat la  $T.\text{false}(x) ? x : T.\&(x,y)$
  - ▶ **x || y** este evaluat la  $T.\text{true}(x) ? x : T.|(x,y)$

9

Operatorii **&&** și **||** nu pot fi supraîncărcați direct. Totuși, ei sunt evaluați în funcție de operatorii **&**, **|**, **true** și **false**, care pot fi supraîncărcați.

Dacă variabilele **x** și **y** sunt amândouă de tipul **T**, operatorii logici sunt evaluați după cum urmează:

- **x && y** este evaluat la **T.false(x) ? x : T.&(x,y)**

Expresia se traduce: Dacă **x** este fals după cum este definit operatorul **false** în **T**, răspunsul este **x**; altfel, răspunsul este acela al aplicării operatorului **&** al lui **T** peste **x** și **y**

- **x || y** este evaluat la **T.true(x) ? x : T.|(x,y)**

Expresia se traduce: Dacă **x** este adevărat după cum este definit operatorul **true** în **T**, răspunsul este **x**; altfel, răspunsul este acela al aplicării operatorului **|** al lui **T** peste **x** și **y**

## Supraîncărcarea operatorilor de conversie

### ► Supraîncărcarea operatorilor de conversie

```
public static explicit operator Time (float hours)
{ ... }
public static explicit operator float (Time t1)
{ ... }
public static implicit operator string (Time t1)
{ ... }
```

### ► Dacă o clasă definește un operator al conversiei către string, ar trebui să suprascrie metoda **ToString**

10

Puteți defini operatori de conversie implicați sau expliți pentru clasele Dvs. și să definiți operatori de cast care sunt folosiți pentru a converti date de la un tip la altul. În continuare sunt câteva exemple ale supraîncărcării operatorilor:

- explicit operator Time (int minutes)

Acest operator transformă un **int** într-un **Time**. Este explicit deoarece nu toate **int**-urile pot fi convertite, furnizarea unui argument negativ rezultând într-o excepție

- explicit operator Time (float minutes)

Acest operator transformă un **float** într-un **Time**. Este explicit deoarece nu toate **float**-urile pot fi convertite, furnizarea unui argument negativ rezultând într-o excepție

- implicit operator int (Time t1)

Acest operator transformă un **Time** într-un **int**. Este implicit deoarece toate obiectele de tip **Time** pot fi convertite în siguranță la **int**.

- explicit operator float (Time t1)

Acest operator transformă un **Time** într-un **float**. Este explicit deoarece, deși toate obiectele de tip **Time** pot fi convertite în siguranță la **float**, reprezentarea valorii în virgulă mobilă nu va fi tot timpul exactă. (acest risc există tot timpul când se lucrează cu valori în virgulă mobilă)

- implicit operator string (Time t1)

Acest operator convertește un **Time** într-un **string**. Acest cast este implicit deoarece nu există riscul pierderii de informație în urma conversiei.

### Suprascrierea metodei ToString

Regulile de design al claselor spun că, din motive de consecvență, dacă o clasă are un operator de conversie la string, ar trebui suprascrisă metoda **ToString**, care să implementeze aceeași operație. Multe clase și metode din namespace-ul **System** – Console.WriteLine de exemplu – folosesc **ToString** pentru a crea o variantă printabilă a obiectului.

### Exemple de cod

Următorul exemplu de cod demonstrează implementarea unor operatori de conversie. De asemenea demonstrează o metodă de a implementa metoda **ToString**.

```
public struct Time
{
    ...
    // Conversion operators
    public static explicit operator Time (int minutes)
    {
        return new Time(0, minutes);
    }
    public static explicit operator Time (float minutes)
    {
        return new Time(0, (int)minutes);
    }
    public static implicit operator int (Time t1)
    {
        return t1.TotalMinutes( );
    }
    public static explicit operator float (Time t1)
    {
        return t1.TotalMinutes( );
    }
    public static implicit operator string (Time t1)
    {
        return t1.ToString( );
    }
    // Inherited virtual methods (from Object)
    public override string ToString( )
    {
        return String.Format("{0}:{1:00}", hours, minutes);
    }
    ...
}
```

## Supraîncărcarea multiplă a operatorilor

- Același operator poate fi supraîncărcat de mai multe ori

```
public static Time operator+(Time t1, int hours)
{...}

public static Time operator+(Time t1, float hours)
{...}

public static Time operator-(Time t1, int hours)
{...}



public static Time operator-(Time t1, float hours)
{...}
```

11

Se poate supraîncărca același operator de multe ori pentru cazul în care poate să primească diferiți parametri și să se comporte diferit în funcție de aceștia. La compilare, se ia decizia privind ce metodă se va apela în funcție de tipul parametrilor cu care a fost invocat operatorul.

### Exemplu de cod

```
public struct Time
{
    ...
    public static Time operator+(Time t1, int hours)
    {
        return t1 + new Time(hours, 0);
    }
    public static Time operator+(Time t1, float hours)
    {
        return t1 + new Time((int)hours, 0);
    }
    public static Time operator-(Time t1, int hours)
    {
        return t1 - new Time(hours, 0);
    }
    public static Time operator-(Time t1, float hours)
    {
        return t1 - new Time((int)hours, 0);
    }
    ...
}
```

 itacad  
you@technology  
  


## Indentificați greșala

- ▶ 

```
public bool operator != (Time t1, Time t2)
{ ... }
```
- ▶ 

```
public static operator float (Time t1)
{ ... }
```
- ▶ 

```
public static Time operator += (Time t1, Time t2)
{ ... }
```
- ▶ 

```
public static bool Equals(Object obj)
{ ... }
```
- ▶ 

```
public static int operator implicit(Time t1)
{ ... }
```

12

1. Operatorii trebuie să fie statici deoarece ei aparțin claselor și nu instanțelor. Definiția operatorului != ar fi trebuit să fie:

```
public static bool operator != (Time t1, Time t2)
{ ... }
```

2. Lipsește tipul operatorului. Acesta trebuie să fie fie **implicit**, fie **explicit**. Codul ar trebui să arate așa:

```
public static implicit operator float (Time t1)
{ ... }
```


3. Operatorul += nu poate fi supraîncărcat. Totuși, += este evaluat folosind operatorul +, care poate fi supraîncărcat
4. Metoda **Equals** trebuie să fie o metodă a instanței, nu a clasei, deci nu statică. Dar, de asemenea, ar trebui să suprascrie metoda moștenită de la **Object**, deci ar trebui adăugat cuvântul cheie **override**.

```
public override bool Equals(Object obj)
{ ... }
```

5. Au fost inversate cuvintele cheie **int** și **implicit**. Declarația corectă este următoarea:

```
public static implicit operator int(Time t1)
{ ... }
```



  
itacad  
you@technology  
  
Microsoft®  
.NET

## Crearea și folosirea delegaților

- ▶ Scenariu: Brutărie
- ▶ Analiza problemei
- ▶ Crearea delegaților
- ▶ Folosirea evenimentelor

13

Delegații permit scrierea de cod ce poate schimba dinamic metodele pe care le apelează. Această caracteristică oferă flexibilitate ce permite metodelor să varieze independent codul pe care îl invocă.

La finalul acestei lecții, veți putea să:

- analizați un scenariu în care folosirea delegaților se dovedește utilă
- definiți și folosiți delegați

În continuare, codul prezentat este scris astfel încât să fie ușor de înțeles care este semnificația acestuia în raport cu situația care se dorește a fi implementată programatic. Detalii care nu au legătură cu conceptele principale sunt omise în cod.

## Definire brutărie

- ▶ Brutăria Alma produce periodic pâine și covrigi

```
class Brutarie
{
    private int paine, covrigi;

    public int VindePaine(int cantitate) { ... }
    public int VindeCovrigi(int cantitate) { ... }
    public bool ProduceDisponibile() { ... }

    ...
}


static void Main( )
{
    Brutarie brutAlma = new Brutarie();
}
```

14

Clasa **Brutarie** are rolul de a gestiona o brutărie prin păstrarea de informații precum cantitatea de pâine și covrigi existente la un moment dat. În cadrul brutăriei se pot cumpăra produse disponibile. Comportamentul referitor la producția de pâine și covrigi este ascuns.

Alma este un exemplu de brutărie, prin urmare este o instanță a clasei **Brutărie**. Periodic, cantitatea de pâine și covrigi va crește, după va scădea de fiecare dată când se efectuează cumpărături. Cele 2 metode, **VindePaine** și **VindeCovrigi** returnează numărul de produse achiziționate. Acesta poate fi mai mic decât valoarea specificată atunci când nu mai sunt produse disponibile.

Metoda **ProduceDisponibile** va putea fi folosită pentru a verifica dacă brutăria are la dispoziție covrigi și pâine de vânzare. În caz afirmativ, se întoarce ca răspuns **true**.

 itacad  
you@technology  
  
Microsoft  
.NET

## Scenariul de brutărie

- ▶ Problema
  - ▶ Copiii vor să cumpere pâine sau covrigi doar dacă sunt calde
  - ▶ Cum pot afla ei când sunt gata pâinea sau covrigii?
- ▶ Soluții posibile
  1. Copiii ar putea veni să verifice, din când în când, dacă produsele sunt gata
  2. Brutăria anunță copiii care s-au înscris când trebuie să vină

15

Pentru a înțelege funcționarea delegaților, considerați exemplul unei brutării pentru care delegații reprezintă o soluție bună

### Problemă

Într-o brutărie, angajații produc în fiecare zi pâine și covrigi calzi, pentru a le vinde. Copiii sunt cei mai nerăbdători să cumpere, fie covrigi, fie pâine. Problema este că ei vor să le cumpere imediat ce sunt scoase din cuptor. Cum vor ști când să vină la brutărie?

### Soluții posibile

O soluție ar fi ca acești copii să vină, din când în când, să verifice dacă pâinea sau covrigii sunt gata. În acest caz, problema este că s-ar pierde mult timp pe drum, copiii fiind nevoiți să vină la brutărie de mai multe ori pe zi.

O a doua soluție ar putea fi ca un angajat al brutăriei să îl sune pe fiecare copil să îl anunțe că produsele sunt gata. Dar asta ar însemna ca brutăria să angajeze o persoană special pentru asta.

## Soluția I

- ▶ Fiecare copil cunoaște o anumită brutărie
- ▶ Acesta verifică dacă există pâine caldă, respectiv covrigi calzi, după care cumpără cantitatea dorită

```
class Copil
{
    private Brutarie brutarieSursa;

    public Copil(Brutarie specificaBrutarie)
    { brutarieSursa = specificaBrutarie; }

    public void CumparaPaine(int cantitate) { }
    public void CumparaCovrigi(int cantitate) { }
    public bool VerficaBrutarie()
    { return brutarieSursa.ProduseDisponibile(); }
}
```

16

În funcție de locația copilului, acesta preferă o anumită brutărie. Instanța brutăriei este transmisă prin intermediul constructorului și poate fi distinctă de la un copil la altul.

Copiii vor verifica dacă brutăria are stoc disponibil. În cazul în care există și produsele sunt calde, aceștia vor cumpara ceea ce doresc prin intermediul metodelor **CumparaPaine** și **CumparaCovrigi**.

Pentru simplitate, comportamentul individual al fiecărui copil este redat pentru toți copiii în metoda **Main**. În acest fel, se poate observa mai bine modul în care se desfășoară activitatea acestora.

În general, clasa **Copil** simbolizează o entitate din programarea orientată obiect care depinde de finalizarea execuției unui cod care se găsește în altă parte, așa cum este clasa **Brutarie**. În plus, există scenarii bazate pe evenimente în care anumite entități din program se execută doar atunci când este nevoie, la momentul potrivit.

## Soluția I - continuare

```
static void Main(string[] args)
{
    Brutarie brutAlma = new Brutarie();
    Copil c1 = new Copil(brutAlma);
    Copil c2 = new Copil(brutAlma);
    while (true)
    {
        if (c1.verificaBrutarie())
            c1.CumparaCovrigi(10);
        if (c2.VerificaBrutarie())
            c2.CumparaPaine(2);

        Asteapta(); //copiii se joaca
    }
}
```

17

În metoda **Main** se inițializează brutăria Alma, după care se crează 2 copii. Fiecare copil va prefera sa cumpere produse de la brutăria Alma.

Deoarece copiii nu vor ști exact momentul în care brutăria a terminat de copt produsele de panificație, aceștia vor verifica de mai multe ori până când găsesc momentul potrivit. În cazul în care ajung prea tarziu, aceștia vor continua să verifice brutări.



## Soluția I - concluzii

- ▶ Copiii verifică de multe ori când nu e gata
- ▶ Nu toți copiii vor același produs
- ▶ Se produce dezordine la brutarie de fiecare dată când copiii vin sa verifice
- ▶ Scenariul este greu de implementat în această formă dacă există mai multe brutării și grupuri de copii



itacad  
you@technology

Microsoft  
.NET

## Soluția 2

- ▶ Schimbă fluxul astfel încât brutăria este cea care inițiază procesul și nu copiii
- ▶ Implementarea soluției se face prin utilizarea de delegați și de evenimente
- ▶ Delegații au rolul de a înscrie un copil pentru a putea fi contactat în momentul în care sunt produse disponibile
  - ▶ Acest proces poartă numele de “handling”
- ▶ Evenimentele gestionează cele 2 tipuri de produse împreună cu copiii înscriși
  - ▶ Vor exista 2 evenimente, fiecare având o stare proprie

19

 itacad  
you@technology  
  
Microsoft®  
.NET

## Class Copil

```

class Copil
{
    ...

    public int nrPaini, nrCovrigi;

    public void CovrigiGata()
    { CumparaCovrigi(nrCovrigi); }

    public void PaineGata()
    { CumparaPaine(nrPaini); }
}
        
```

20

Clasa Copil conține 2 metode, pe care le vom folosi pentru a anunța că produsele sunt gata, în funcție de dorință.

Dacă un copil vrea să știe când sunt covrigii gata, vom folosi metoda **CovrigiGata()**. De asemenea, dacă vrea să afle când e pâinea gata, folosim metoda **PaineGata()**.

```

class Copil
{
    private Brutarie brutarieSursa;
    public int nrPaini, nrCovrigi;

    public Copil(Brutarie specificaBrutarie)
    { brutarieSursa = specificaBrutarie; }

    public bool verificaBrutarie() { }
    public void CumparaPaine(int cantitate) { }
    public void CumparaCovrigi(int cantitate) { }

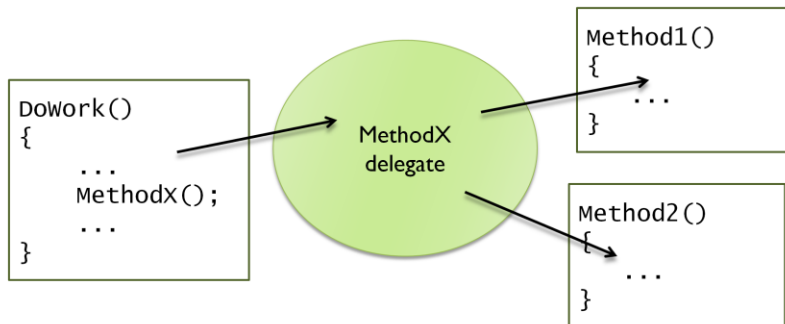
    public void CovrigiGata()
    { CumparaCovrigi(nrCovrigi); }

    public void PaineGata()
    { CumparaPaine(nrPaini); }
}
    
```



## Delegați

- ▶ Un delegat permite apelul indirect al unei metode
  - ▶ Sunt variabile care referă metode
  - ▶ Toate metodele apelate de către același delegat trebuie să aibe aceiași parametri și aceeași valoare întoarsă



21

Un delegat conține o referință către o metodă și nu numele metodei. Folosind delegați, se pot invoca metode fără a ști numele acestora. Apelarea delegatului va executa de fapt metoda referențiată.

Un delegat este similar unei interfețe. Acesta va specifica un contract între cel ce apelează și implementare. Delegatul asociază un nume cu specificarea unei metode. O implementare a metodei poate fi atașată acestui nume, și metoda se poate apela folosind numele delegatului. Principala limitare este că toate metodele implementatoare trebuie să aibe aceiași parametri și aceeași valoare întoarsă.

Pentru a folosi un delegat acesta trebuie întâi definit și apoi instanțiat.


itacad  
you@technology

Microsoft  
.NET

## Crearea Delegatilor

- Tipul unui delegat specifică valoarea întoarsă și parametrii pentru metodele care pot fi referențiate

```
delegate void TrimiteSMS( );
```

Tipul întors de metoda pe care o referențiază delegatul

Parametrii primiți de metoda pe care o referențiază delegatul

- Instanțierea delegaților

```
Copil cp = new Copil( );  
TrimiteSMS sms = new TrimiteSMS(cp.PaineGata);
```

22

### Definirea delegaților

Un delegat specifică o valoare întoarsă și parametrii pentru metodele pe care le referențiază. Următoarea sintaxă poate fi folosită pentru a defini un delegat:

```
public delegate void TrimiteSMS( );
```


Observați că sintaxa pentru definirea delegaților este asemănătoare celei pentru definirea metodelor.

### Instanțierea delegaților

După definirea unui delegat, acesta trebuie instanțiat și trebuie referențiat către o metodă. Pentru a face acest lucru, trebuie apelat constructorul delegatului și furnizată metoda obiectului ce se dorește a fi apelată. Următorul exemplu demonstrează:

```
public delegate void TrimiteSMS( );
```

```
class Test  
{  
    void Example()  
    {  
        Copil cp = new Copil( );  
        TrimiteSMS sms;  
        sms = new TrimiteSMS(cp.PaineGata);  
        sms();  
    }  
}
```

 itacad  
you@technology  
  
Microsoft  
.NET

## Folosirea delegaților

► Pentru a apela un delegat, se folosește următoarea sintaxă

```
public delegate void TrimiteSMS( );

class Test {
    TrimiteSMS sms;
    Copil cp = new Copil( );
    sms = new TrimiteSMS(cp.PaineGata);
    sms( );
}
```

Nu se specifică implementare

Aici se face apelul

Apelul nu se face aici

Un delegat este o variabilă ce invocă o metodă. Apelul este făcut asemănător apelului unei metode, cu excepția faptului că delegatul va înlocui numele metodei.

### Exemplu

Următorul exemplu definește definiția, crearea și folosirea delegaților pentru exemplul cu brutaria. Se populează un **ArrayList** cu instanțe ale delegaților ce referențiază metodele de anunt al copiilor. Metoda **AnuntaCopii** va itera de data aceasta pe delegați și îi va apela.

```
public delegate void TrimiteSMS( );

public class ContacteazaCopii
{
    public void AdaugaCopil(TrimiteSMS contact)
    {
        listaContacte.Add(contact);
    }
    public void AnuntaCopii( )
    {
        foreach(TrimiteSMS sms in listaContacte)
            sms( );
    }
    private ArrayList listaContacte = new ArrayList( );
}
```

class Exemplu

```
{  
    public static void Main( )  
    {  
        ContacteazaCopii contacteaza = new ContacteazaCopii( );  
        Copil cp1 = new Copil( );  
        contacteaza .AdaugaCopil(new TrimiteSMS(cp1.PaineGata));  
        Copil cp2 = new Copil( );  
        contacteaza .AdaugaCopil(new TrimiteSMS(cp2.CovrigiGata));  
        contacteaza.AnuntaCopii( );  
    }  
}
```



itacademy  
you@technology

## Definirea și folosirea evenimentelor

- ▶ Funcționarea evenimentelor
- ▶ Definirea evenimentelor
- ▶ Trimiterea parametrilor evenimentelor



24

În exemplul cu brutăria, s-a discutat folosirea delegaților pentru a rezolva problema legată de faptul cum sunt anunțați copiii că produsele sunt gata. Totuși, cineva trebuie să verifice dacă pâinea sau covrigii sunt gata. Se poate rezolva această problemă folosind evenimente.

Evenimentele permit unui obiect să notifice alte obiecte în momentul în care apar modificări. Obiectele se pot înregistra ca fiind interesate de un anumit eveniment și vor fi anunțate când acel eveniment.

La finalul acestei lecții, veți putea să:

- definiți evenimente
- tratați evenimente



## Definirea evenimentelor

### ► Definirea unui eveniment

```
public delegate void TrimiteSMS( );
private event TrimiteSMS PaineDisponibila;
```

### ► Înregistrarea pentru un eveniment

```
Copil cp = new Copil( );
...
PaineDisponibila += new TrimiteSMS(cp.PaineGata);
```

### ► Notificarea obiectelor înregistrate

```
public void AnuntaPaine( )
{
    if (PaineDisponibila != null) //verificare abonați
        PaineDisponibila( );
}
```

26

Evenimentele în C# folosesc delegați pentru a apela metodele obiectelor înregistrate. În momentul în care un eveniment este declanșat, este posibil să se apeleze mai mulți delegați. Ordinea în care aceștia sunt invocați nu este însă garantată. Dacă unul dintre delegați aruncă o excepție, este posibilă oprirea procesării evenimentului pe de-a-ntregul, rezultând în neapelarea celorlalți delegați.

### Definirea delegaților

Pentru a defini un eveniment, un publisher în primul rând definește un delegat și îl pune la baza evenimentului. Următorul cod definește un delegat numit **TrimiteSMS** și doua evenimente numite **PaineGata** și **CovrigiGata** ce invocă delegați de tipul **TrimiteSMS** atunci când sunt ridicate.

```
public delegate void TrimiteSMS( );
private event TrimiteSMS PaineDisponibila;
private event TrimiteSMS CovrigiDisponibila;
```

### Înregistrarea pentru un eveniment

Obiectele ce se înregistrează specifică o metodă ce va fi apelată automat în momentul în care evenimentul este ridicat. Acest lucru se realizează adăugând metoda de callback delegatului definit de către eveniment.

De exemplu, în scenariul cu brutăria, puteți crea două instanțe de copil și amândouă să se înregistreze pentru evenimentul **PaineGata**:

```
Copil cp1 = new Copil( );
Copil cp2 = new Copil( );
...
PaineDisponibila += new TrimiteSMS(cp1.PaineGata);
PaineDisponibila += new TrimiteSMS(cp2.PaineGata);
CovrigiDisponibili += new TrimiteSMS(cp2.CovrigiGata);
```

### Notificarea obiectelor înregistrate

Pentru a notifica obiectele înregistrate, trebuie ridicat evenimentul. Sintaxa este aceeași ca la apelarea unui delegat. În exemplul nostru, metoda

**AnuntaPaineDisponibila** a componentei ce se ocupa de monitorizarea brutariei nu mai trebuie să facă iterația prin delegați:

```
public void AnuntaPaineDisponibila( )
{
    if (PaineDisponibila != null)
        PaineDisponibila( );
}
```

Execuția unui eveniment în acest fel va căuta toți delegații si îi va invoca. În acest exemplu, toți copiii ce au fost înregistrați la acest eveniment vor fi anuntați. Observați că întâi se verifică dacă evenimentul are măcar un delegat înregistrat. Fără această verificare, s-ar obține excepție la rulare.




itacad  
you@technology

Microsoft  
.NET

## Trimiterea de parametri evenimentelor

- ▶ Parametri pentru evenimente sunt trimiși ca EventArgs
  - ▶ Trebuie definită o clasă derivată din EventArgs pentru a acționa ca un container pentru parametrii evenimentului
- ▶ Aceeași metodă de înregistrare poate fi apelată de mai multe evenimente
  - ▶ Întotdeauna trimiteți obiectul publisher ca prim parametru al metodei

27

### Reguli pentru trimiterea parametrilor evenimentelor

Pentru a trimite parametri unei metode înregistrate, înglobați parametrii într-o singură clasă ce oferă o metodă ce permite obținerea lor. Derivați această clasă din **System.EventArgs**

De exemplu, în cazul scenariului brutăriei, metodele de anunțare a copiilor au nevoie de prețul produselor pentru a determina câți bani trebuie să își ia copiii. Pentru a rezolva această problemă, puteți crea următoarea clasă ce trimite prețul produselor.

```
public class DetaliiPaine: EventArgs
{
    public readonly double pret;

    public DetaliiPaine(double pret)
    {
        this.pret = pret;
    }
}
```

Clasa **DetaliiPaine** conține un parametru de tip întreg. Constructorul stochează pretul și după care acesta poate fi citit atunci când este nevoie.

### Obiectul ce a trimis evenimentul

Un obiect se poate înregistra la mai multe evenimente, aceștia putând să fie lansați de publisher-i diferiți. De asemenea, se poate folosi aceeași metodă pentru callback pentru toate evenimentele. De aceea, de obicei evenimentul va conține informații legat de publisher-ul ce l-a lansat. Prin convenție, acesta este primul parametru trimis metodei înregistrate și este de obicei numit sender. Codul următor prezintă o versiune nouă a metodelor **PaineGata** și **CovrigiGata**, modificându-le pentru a aștepta senderul ca prim parametru și pretul ca al doilea.

```
public class Copil
{
    ...

    public void PaineGata(object sursa, DetaliiPaine detalii)
    {
        // Preia prețul
        double currentPrice = detalii.pret;

        ...
    }
    public void CovrigiGata(object sursa, DetaliiCovrigi detalii)
    {
        // Preia prețul
        double currentPrice = detalii.pret;

        ...
    }
}
```

După modificarea metodelor din clasa Copil, trebuie să modificăm și modul în care vom ridica eventul.

Înainte:

```
public void AnuntaPaineDisponibila( )
{
    if (PaineDisponibila != null)
        PaineDisponibila( );
}
```

După

```
public void AnuntaPaineDisponibila( )
{
    if (PaineDisponibila != null)
    {
        DetaliiPaine detPaine = new DetaliiPaine(1.2);
        PaineDisponibila( this, detPaine);
    }
}
```

Se poate observa că se trimite ca parametri obiectul sursă (this) și un obiect de tipul DetaliiPaine care conține pretul produsului.



itacad  
you@technology

Microsoft  
.NET

## Overview

- ▶ Introducere a operatorilor
- ▶ Supraîncărcarea operatorilor
- ▶ Crearea și folosirea delegaților
- ▶ Definirea și utilizarea evenimentelor



28

- Se pot supraîncărca următorii operatori:  $+=$ ,  $-=$ ,  $/=$  și  $\%=?$
- În ce caz ar trebui ca un operator de cast să fie explicit?
- Ce este un delegat?
- Cum se face înscrierea la un eveniment?