



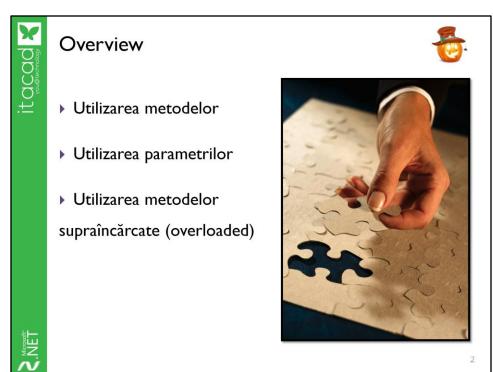


Modulul 4

Metode și parametri – C#



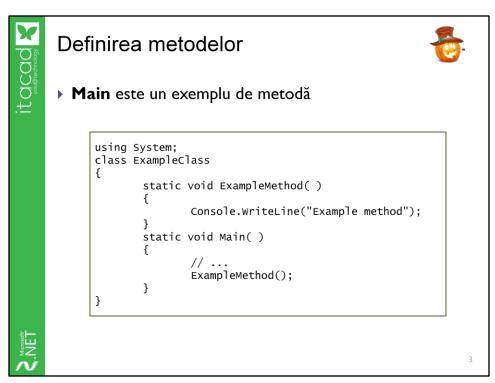




- ➤ Pentru a putea fi mai uşor de menţinut şi depanat, o aplicaţie este împărţită pe mai multe unităţi funcţionale. Un alt avantaj al acestei divizări este reutilizarea codului aceeaşi unitate de cod apare de mai multe ori în aplicaţie.
- > Aceste unități de cod, funcționale, se numesc metode. O metodă este un membru al unei clase care face o anumită acțiune sau care returnează o anumită valoare.
- > După completarea acestui modul studentul va fi capabil să:
 - Creeze metode statice care primesc parametri și returnează o anumită valoare
 - Transmită parametri în diverse moduri
 - Declare și să folosească proceduri supraîncărcate







- ➤ O metodă constă în mai multe instrucțiuni grupate sub un anumit nume. Ea poate fi asemănată cu o funcție, cu o procedură sau cu o subrutină.
- Codul din prezentare conţine trei metode:
 - Main() este punctul de intrare în orice aplicație
 - WriteLine() face parte din .NET Framework fiind o metodă statică din clasa System.Console
 - ExampleMethod() este o metodă din interiorul clasei ExampleClass şi apelează metoda WriteLine()
- ➤ În **C#**, toate metodele aparțin unei clase, ceea ce contrastează cu celelalte limbaje precum *C*, *C++*, *Visual Basic*, care permit funcții globale
- > Crearea unei metode
 - Numele metodei
 - Nu trebuie să aibă același nume cu cel al unei variabile, constante sau alt element din acea clasă
 - Poate fi orice identificator permis în C# și este case-sensitive
 - Lista de parametri
 - Numele metodei este urmat de o listă de parametri între paranteze. Parantezele trebuie întotdeauna să existe, chiar dacă metoda nu are niciun parametru
 - Corpul metodei
 - După paranteze urmează corpul metodei, care se află între acolade static void MethodName() { method body }







Apelul unei metode



- ▶ Apelul unei metode din aceeași clasă
 - Numele metodei + lista de parametri în paranteză
- ▶ Apelul unei metode dintr-o clasă diferită
 - Indicarea clasei din care face parte metoda
 - Metoda trebuie să fie publică

Microsoft

4

> Apelul unei metode

- Se realizează specificând numele metodei şi lista de parametri între paranteze rotunde. Parantezele apar chiar dacă nu există niciun parametru: MethodName()
- În exemplul următor se execută instrucţiunile din *Main()*. Mai întâi se va afişa mesajul "The program is starting". Urmează apelul metodei *ExampleMethod()* din aceeaşi clasă, care printează "Hello, World". La sfârşit, controlul este dat instrucţiunii următoare apelului metodei, care va afişa mesajul "The program is ending".





> Apelul metodelor din interiorul altor clase

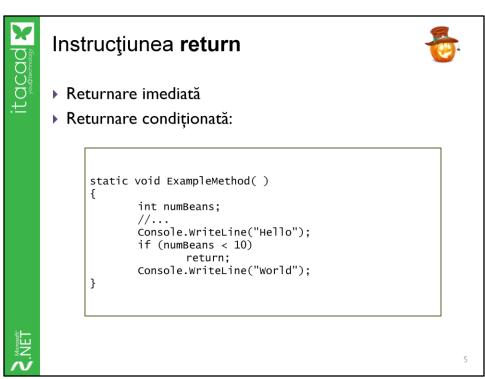
- Pentru a putea apela o metodă dintr-o altă clasă trebuie:
 - Specificată clasa care conţine metoda ClassName.MethodName();
 - Declararea metodei în interiorul clasei de care aparține trebuie realizată cu *public*
- Următorul exemplu arată cum este apelată metoda TestMethod() din clasa A în interiorul lui Main() din clasa B

```
using System;
class A
{
          public static void TestMethod()
          {
                Console.WriteLine("This is TestMethod in class A");
        }
} class B
{
          static void Main()
          {
                      A.TestMethod();
          }
}
```

- Dacă, în exemplul de mai sus, s-ar înlătura numele clasei care precedă **TestMethod()**, compilatorul ar încerca să găsească metoda în clasa **B**. Deoarece nu există nicio metodă cu acest nume în clasa **B**, se va afişa mesajul următor: "The name 'TestMethod' does not exist in the class or namespace 'B'."
- Dacă metoda *TestMethod()* nu ar fi declarată publică în clasa *A,* aceasta ar fi considerată privată, implicit. În acest caz, la compilare se va primi mesajul: "'A.TestMethod()' is inaccessible due to its protection level."
- *public* și *private* specifică gradul de accesibilitate asupra metodei din afara clasei







- ➤ Intrucţiunea *return* poate fi folosită pentru întoarcerea imediată la programul care a apelat metoda. Dacă aceasta nu există, întoarcerea va avea loc după executarea ultimei instrucţiuni din metodă
- ➤ În următorul exemplu se va afișa "Hello", după care se va întoarce la programul apelant.

- > Utilizarea nu este utilă în acest caz, pentru că nu se va afișa niciodată continuarea. În situația în care compilatorul de C# are alarmele activate la nivelul 2 sau mai sus, acesta va printa mesajul: "Unreachable code detected."
- ➤ De obicei, instrucțiunea *return* se execută atunci când o condiție este îndeplinită, verificarea făcându-se cu instrucțiunea *if* sau *switch*

```
static void ExampleMethod()
{
    int numBeans;
    //...
    Console.WriteLine("Hello");
    if (numBeans < 10)
        return;
    Console.WriteLine("World");
}</pre>
```







Metode ce întorc valori



- Metoda va avea un tip diferit de void
- Instrucțiunea return va returna o expresie
- ▶ Metodele **non-void** returnează o valoare

```
static int TwoPlusTwo() {
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}

public static void Main()
{
    int x;
    x = TwoPlusTwo();
    Console.WriteLine(x);
}
```

Instrucţiunea return poate fi folosită şi pentru a întoarce o valoare. Pentru aceasta trebuie:

- Ca declararea metodei să conţină tipul valorii ce va fi returnată
- Să fie adăugată o instrucțiune *return* în corpul metodei
- Să fie specificată valoarea care trebuie returnată în instrucțiunea return
- ➤ Pentru a declara o metodă cu un tip, trebuie înlocuit cuvântul cheie **void** cu tipul corespunzător.
- ➤ Instrucțiunea *return* va da controlul imediat programului apelant și va returna o valoare rezultată în urma evaluării expresiei ce urmează după *return*.
- Exemplul din slide prezintă o metodă care returnează o valoare de tip *int*: *TwoPlusTwo()*. Rezultatul este păstrat într-o variabilă locala din *Main()*, *x*, care este afișată după aceea.
- Metodele *non-void* trebuie să întoarcă o valoare, ceea ce înseamnă că trebuie să existe cel puţin o instrucţiune *return* în corpul metodei. Dacă la compilare se detectează că nicio instrucţiune *return* nu există, se va primi mesajul de eroare: "Not all code paths return a value."
- ➤ Acelaşi mesaj de eroare este întors și dacă se detectează că nu toate căile din program conțin o instrucțiune return:

Instrucţiunea *return* returnează o singură valoare. Dacă este necesară returnarea mai multor valori se vor adăuga noi parametri precedaţi de unul dintre cuvintele cheie *ref* sau *out*. Mai multe detalii despre *ref* şi *out* se află în slide-urile următoare.







Utilizarea variabilelor locale



- Variabile locale
 - ▶ Create la începutul metodei
 - Private pentru metodă
 - Distruse la ieșirea din metodă
- Variabile partajate
 - Variabilele de clasă pot fi partajate
- Conflicte de nume
 - Variabila locală are același nume cu o variabilă de clasă

Microsoft

Fiecare metodă are un set de variabile locale pe care le folosește în interiorul acesteia. Aceste variabile nu sunt vizibile din exteriorul metodei.

➤ Variabile locale

■ Variabilele locale pot fi incluse în corpul metodei ca în exemplul ce urmează:

```
static void MethodWithLocals()
{
    int x = 1; // Variable with initial value
    ulong y;
    string z;
    ...
}
```

- Variabilele care sunt declarate într-o metodă sunt complet separate de cele din altă metodă chiar dacă au acelaşi nume.
- Memoria necesară pentru fiecare variabilă este alocată în momentul în care metoda este apelată şi este eliberată la sfârşitul execuţiei acesteia. De aceea, valorile variabilelor nu sunt păstrate de la o apelare la alta.





≻Variabile partajate

```
    Considerăm următorul cod care numără de câte ori o metodă este apelată

       class CallCounter_Bad
       {
               static void Init()
                      int nCount = 0;
               }
               static void CountCalls()
                      int nCount;
                       ++nCount;
                      Console.WriteLine("Method called {0} time(s)", nCount);
               static void Main()
               {
                      Init();
                      CountCalls();
                      CountCalls();
               }
       }
       Acest program nu poate funcționa dintr-un motiv important: variabila
       nCount din Init() nu este aceeași cu variabila nCount din CountCalls().
       Nu contează de câte ori este apelată metoda CountCalls(), variabila
       nCount va fi distrusă de fiecare dată când se termină execuția.
       Varianta corectă a programului este:
       class CallCounter_Good
       {
               static int nCount;
               static void Init()
               {
                      nCount = 0;
               static void CountCalls()
                      ++nCount;
                      Console.Write("Method called " + nCount + " time(s).");
               static void Main()
                      Init();
                      CountCalls();
                      CountCalls();
             }
       }
```





■De data aceasta, variabila *nCount* este declarată la nivelul clasei, nu la nivelul metodei. În acest fel va fi vizibilă în toate metodele ce aparţin clasei

≻Conflicte de nume

- În C# poţi numi o variabilă locală la fel ca o variabilă la nivel de clasă, dar acest fapt poate duce la consecințe neașteptate.
- În următorul exemplu *numitems* este declarată ca o variabilă la nivelul clasei *ScopeDemo*, cât și ca variabilă locală pentru metoda *Method1()*. Cele două variabile sunt complet diferite

```
class ScopeDemo
{
    static int numItems = 0;
    static void Method1()
    {
        int numItems = 42;
        ...
    }
    static void Method2()
    {
        numItems = 61;
    }
}
```

■ Deoarece compilatorul nu va trimite niciun mesaj de alertare atunci când variabilele locale și cele de clasă au același nume, ați putea folosi o convenție de nume pentru a deosebi cele două tipuri de variabile.







Declararea și apelul parametrilor



- Declararea parametrilor
 - ▶ Se plasează între paranteze după numele metodei
 - Se declară numele şi tipul parametrilor
- ▶ Apelul metodelor cu parametri
 - Se înlocuiește numele parametrilor cu valori

```
static void MethodWithParameters(int n, string y)
{ ... }

MethodWithParameters(2, "Hello, world");
```

➤ Parametrii permit dezvoltatorului să transmită informaţii în şi din interiorul metodei. La declararea unei metode se pot specifica parametrii acesteia, care urmează după numele metodei, între paranteze rotunde. În exemplele anterioare, nici o metodă nu a avut parametri.

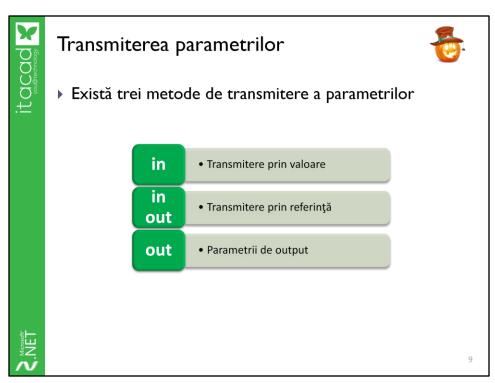
> Declararea parametrilor

- Parametrii sunt declaraţi între paranteze rotunde după numele metodei. Pentru fiecare parametru se specifică numele acestuia şi tipul. Parametrii sunt separaţi prin virgulă.
- În exemplul din slide, metoda *MethodWithParameters()* are doi parametri separaţi prin virgulă:
 - n de tip int
 - y de tip *string*
- > Apelul metodelor cu parametri
 - În momentul în care o metodă este apelată, parametrii sunt înlocuiţi de valorile acestora.
 - În exemplul următor avem două apelări ale metodei. În fiecare caz valorile parametrilor n şi y sunt determinate şi sunt înlocuite atunci când începe execuția metodei MethodWithParameters().

```
MethodWithParameters(2, "Hello, world");
int p = 7;
string s = "Test message";
MethodWithParameters(p, s);
```







➤ Parametrii pot fi transmişi în diferite feluri:

■ Prin valoare

• Aceştia mai sunt numiţi parametri de intrare pentru că sunt transmişi în interiorul metodei, dar niciodată invers.

■ Prin referință

• Mai sunt numiţi parametri *in/out,* pentru ca sunt transmişi metodei, apoi restransmişi în exterior.

■ Prin output

• Denumiți și parametri de ieșire, pentru că informația poate fi transmisă doar în exterior și nu invers.







Transmitere prin valoare



10

- Mecanismul implicit de transmitere a parametrilor
 - Valoarea parametrului este copiată
 - Variabila poate fi modificată în interiorul metodei
 - Modificările nu au niciun efect în exterior

➤ De cele mai multe ori, într-o aplicație, parametrii sunt de intrare. De aceea, dacă nu se specifică nicio caracteristică, parametrul este considerat de intrare.

➤ Parametrul transmis prin valoare este definit prin specificarea tipului şi numelui variabilei. Când metoda este apelată, un nou spaţiu de memorie este alocat pentru parametri şi sunt copiate acolo valorile acestora.

➤ Valoarea care este transmisă metodei trebuie să fie de acelaşi tip cu parametrii sau de un tip care este compatibil pentru conversie. În interiorul metodei se poate scrie cod care modifică parametrii, însă schimbările nu se vor observă din exterior. Valoarea parametrului, la sfârşitul metodei, va fi la fel ca la început.

Următorul exemplu ilustrează cele spuse mai sus:







Transmiterea prin referință



- ▶ Ce este un parametru referință?
 - O referință către o adresă de memorie
- Utilizarea unui parametru referință
 - Cuvântul cheie **ref** apare în definirea și apelul metodei
 - > Schimbările din interiorul metodei sunt vizibile și în exterior
 - Parametrul **trebuie** inițializat înainte de apelul metodei

Microsoft NET

11

> Ce este un parametru referință?

■ Un parametru referință este o referință către o zonă de memorie. Spre deosebire de un parametru valoare, acesta nu rezervă o nouă zonă de memorie pentru a păstra valoarea.

■ Declararea unui parametru referință

Pentru a declara un parametru referință trebuie specificat, în plus, pe lângă tip și nume, cuvântul cheie **ref**

```
static void ShowReference(ref int nld, ref long nCount) {

// ...
}
```

Utilizarea mai multor tipuri de parametrii

■ Cuvântul cheie *ref* se aplică doar parametrului care urmează după el, nu întregii liste de parametri. Fiecare parametru, referință conține *ref* în definiția lui. În exemplul următor, *nld* este transmis prin referință, iar *longVar* prin valoare:

> Potrivirea tip-valoare

La apelarea metodei parametrii referință sunt precedați de cuvântul cheie **ref**. Valoarea variabilei la apelare trebuie să corespundă tipului variabilei din definiția metodei și nu poate fi o constantă sau o expresie.





```
int x = 10;
long q = 20;
ShowReference(ref x, ref q);
```

■ Dacă la apelare nu se specifică *ref*, atunci se va primi următoarea eroare de compilare: "Cannot convert from 'int' to 'ref int.'"

> Modificarea unui parametru referință

■ Dacă se modifică parametrul referință în interiorul metodei, schimbarea va avea loc la nivel global, adică și în exteriorul metodei, pentru că se modifică aceeași zonă de memorie.

> Asignarea parametrului referință înainte de apelare

- Parametrul referință trebuie să fie inițializat cu o valoare înainte de a fi folosit. În exemplul de mai sus acesta este inițializat în *Main()* cu valoarea 6.
- Dacă în metoda *Main()* ar fi fost următorul cod, compilatorul ar fi afișat eroarea: "Use of unassigned local variable 'k."

```
int k;
AddOne(ref k);
Console.WriteLine(k);
```







Parametrii de output



- ▶ Ce sunt parametrii de ieşire?
 - ▶ Valorile sunt transmise spre exterior
- Utilizarea parametrilor de ieşire
 - La fel ca ref, doar că valorile nu sunt transmise în metodă
 - Se folosește cuvântul cheie out

Microsoft

Ce sunt parametrii de output?

- Parametrii de ieşire sunt la fel ca parametrii referință, doar că ei pot fi transmişi doar într-un singur sens: din metodă spre exterior. Ei reprezintă nişte referințe către zone de memorie care păstrează o anumită informație. Față de parametrii valoare, aceștia nu trebuie neapărat inițializați înainte de apelul metodei.
- Sunt foarte utili când se doreşte transmiterea unui grup de valori în exterior sau când nu se doreşte iniţializarea parametrilor dinainte.

> Utilizarea parametrilor de output

- Pentru a declara un parametru de ieşire se foloseşte cuvântul cheie *out* plasat înaintea tipului, la fel ca în exemplul din slide.
- Pentru fiecare parametru de *output* se specifică separat acest lucru: *out* va preceda pe fiecare în parte în declararea metodei. La fel se întâmplă şi când este apelată metoda.
- În interiorul metodei parametrul nu va fi tratat preferențial, ci ca orice altă variabilă.







Liste variabile de parametrii



13

- Cuvântul cheie params
- ▶ Se declară ca un vector la sfârșit
- Intotdeauna transmitere prin valoare

Declararea metodelor cu număr de parametrii variabil

- Pentru a specifica existența unui număr variabil de parametrii se folosește cuvântul cheie *params*.
- Pentru fiecare metodă nu există mai mult de un singur parametru *params*.
- Întotdeauna parametrul *params* se declară după ceilalți parametri, la sfârșit.
- Lista de parametri de lungime variabilă este un vector unidimensional.

> Transmiterea valorilor

- Când se apelează o metodă cu număr variabil de parametri, aceștia pot fi specificați în două moduri:
 - Fiecare valoare în parte separată prin virgulă
 - Ca un vector de valori
- Următorul exemplu evidențiază ambele tehnici:

```
static void Main()
{
          long x;
          x = AddList(63, 21, 84); // List
          x = AddList(new long[]{ 63, 21, 84 }); // Array
}
```

- Indiferent care dintre modalități este folosită, în metodă se va considera că parametrii au fost transmişi într-un vector. Se poate folosi proprietatea *Length* a unei vector pentru a afla câți parametri au fost transmişi.
- Parametrii declaraţi cu *params* sunt transmişi prin valoare, ceea ce înseamnă că orice modificare în interiorul metodei nu va fi vizibilă în exterior.







Concluzii transmitere parametri



Mecanisme

- ▶ Transmiterea prin valoare e cea mai întâlnită
- Intoarcerea cu **return** e utilizată pentru o singură valoare
- ▶ Pentru vector, liste etc. se folosește ref sau out
- Folosiți ref doar dacă datele sunt transmise și in și out

▶ Eficiență

Transmiterea prin referintă este cea mai eficientă deoarece nu se copiaza datele ci se indica unde sunt acestea.



14

➤ Fiind foarte multe posibilități de a alege parametrii, un dezvoltator trebuie sa ţină cont de următoarele criterii:

➤ Mecanisme

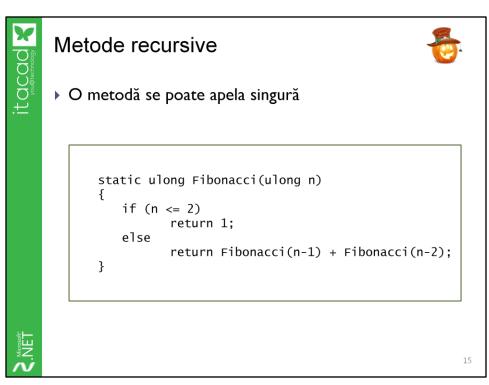
- Parametrii transmişi prin valoare oferă o formă limitată de protecţie, ce constă în faptul că orice schimbare asupra acestora, în metodă, nu va fi transmisă în afară. Aceasta sugerează că, în afară de situaţia în care se doreşte transmiterea informaţiei în exterior, parametrii valoare sunt cei recomandaţi.
- Dacă trebuie transmisă informație din metodă spre programul apelant se poate utiliza instrucțiunea *return*. Aceasta este ușor de folosit numai că nu poate întoarce decât o singură valoare. Pentru mai multe valori utilizați *ref* (comunicare în ambele direcții) sau *out* (transfer de informație din metodă)

Eficienţă

- Pentru tipurile predefinite este mai eficient să fie transmise prin valoare.
- Nu există un set de reguli exact pentru a implementa un program eficient şi, de multe ori, trebuie considerate, pentru o aplicaţie, mai degrabă corectitudinea, stabilitatea şi robusteţea înaintea eficienţei.







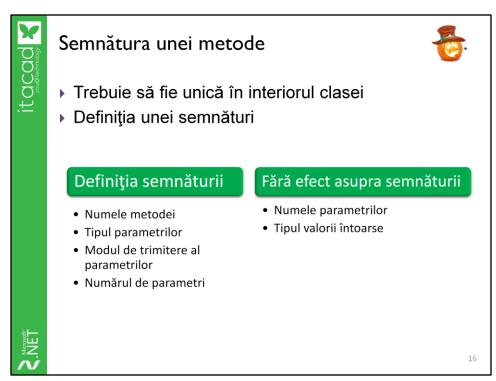
- ➤ O metodă se poate apela pe ea însăşi, această tehnică numindu-se *recursivitate*. Recursivitatea este foarte utilă mai ales în manipularea unor tipuri de date mai complexe, precum listele sau arborii
- Metodele în C# pot fi recursive mutual: A apelează B, iar B apelează din nou A
- > Un exemplu de recursivitate:
 - Şirul lui Fibonacci
 - Primii doi termeni au valoarea 1
 - ullet De la termenul 3 în sus, fiecare se calculează ca suma dintre ultimii doi termeni. Exemplu: termenul 3 este calculat ca suma dintre termenul 1 și 2
 - Implementarea metodei Fibonacci este următoarea:

```
static ulong Fibonacci(ulong n)
{
     if (n <= 2)
         return 1;
     else
        return Fibonacci(n-1) + Fibonacci(n-2);
}</pre>
```

ullet Orice metodă recursivă trebuie să aibă o condiție de terminare, în exemplu condiția fiind n <= 2







> Semnătura unei metode este folosită de compilator pentru a face diferența între metodele aceleiași clase. Nu pot exista două metode cu aceeași semnătură, aparţinând aceleiași clase.

> Definiția unei semnături

- Semnătura unei metode constă în nume, numărul de parametri, tipul parametrilor şi determinanţii acestora (out, ref).
- Următoarele trei metode au semnături diferite, deci pot fi declarate în aceeași clasă:

static int LastErrorCode()
static int LastErrorCode(int n)
static int LastErrorCode(int n, int p)

> Elemente care nu afectează semnătura

■ Valoarea returnată nu intră în definiția semnăturii. Următoarele două metode au aceeași semnătură:

static int LastErrorCode(int n) static string LastErrorCode(int n)

■ De asemenea, semnătura nu include numele parametrilor. Următoarele două metode au aceeași semnătură, chiar dacă numele parametrilor este diferit:

static int LastErrorCode(int n) static int LastErrorCode(int x)







Declararea unei metode supraîncărcate



- ▶ Metode care au același nume în interiorul clasei
 - ▶ Se disting prin lista de parametri

- ➤ Metodele supraîncărcate sunt metode cu acelaşi nume, din cadrul aceleiaşi clase. Compilatorul le deosebeşte prin listele de parametri
- ➤ În exemplul din slide, sunt definite două metode **Add**, iar în funcția **Main()** sunt apelate tot două. Deși ambele metode au același nume, compilatorul face diferența între ele prin listele de parametri.
- ➤ Prima metodă primeşte doi parametri de tip *int*. A doua metodă *Add* are trei parametri tot *int*. Pentru că listele de parametri diferă, compilatorul va şti despre care dintre metode este vorba la apelare.
- ➤ Denumirile nu pot fi aceleaşi pentru o metodă şi o varibila/constantă/enumeraţie. Următorul cod este greşit:







Utilizarea metodelor supraîncărcate



- Se folosesc atunci când:
 - Aveți metode similare care au nevoie de parametri diferiți
 - Vreţi să adăugaţi funcţionalităţi noi la o metodă existentă
- ▶ Evitați să le folosiți pentru că:
 - Sunt greu de menţinut
 - Sunt greu de depanat

Microsoft*

18

➤ Presupunem că avem situaţia în care trebuie create două metode care afişează un mesaj de salut unui utilizator. Numele utilizatorului poate fi cunoscut uneori, alteori nu. O variantă ar fi următoarea:

➤ Codul va merge, dar clasa va avea două metode care fac acelaşi lucru. O variantă mai corectă este folosind metode supraîncărcate:





```
static void Greet(string Name)
                             Console.WriteLine("Hello" + Name);
               static void Main()
                              Greet();
                              Greet("Alex");
               }
Metodele supraîncărcate sunt foarte utile atunci când se dorește adăugarea de noi
funcționalități codului existent. Exemplul următor ilustrează acest lucru:
       class GreetDemo
       {
               enum TimeOfDay { Morning, Afternoon, Evening }
               static void Greet()
               {
                      Console.WriteLine("Hello");
               static void Greet(string Name)
                      Console.WriteLine("Hello" + Name);
               static void Greet(string Name, TimeOfDay td)
                      string Message = "";
                      switch(td)
                              case TimeOfDay.Morning:
                                     Message="Good morning";
                                     break;
                              case TimeOfDay.Afternoon:
                                     Message="Good afternoon";
                                     break;
                              case TimeOfDay.Evening:
                                     Message="Good evening";
                                     break;
                      Console.WriteLine(Message + " " + Name);
               }
               static void Main()
                      Greet();
                      Greet("Alex");
                      Greet("Sandra", TimeOfDay.Morning);
               }
       }
```

➤ Deşi metodele supraîncărcate sunt folositoare în unele cazuri, utilizarea lor frecventă în cadrul aceleiaşi clase poate duce la dificultăţi în menţinerea şi localizarea erorilor.







Sumar



- Utilizarea metodelor
- Utilizarea parametrilor
- Utilizarea metodelor supraîncărcate (overloaded)

Microsoft

19