O privire detaliata asupra claselor si metodelor

Specificatorii de acces din C#

Controlul accesului la membrii unei clase se realizeaza prin utilizarea urmatorilor specificatori de acces:

- *public* (membrii publici pot fi accesati liber de codul din afara clasei);

-private (membrii privati sunt accesibili numai metodelor definite in aceeasi clasa);

-protected (membrii protejati pot fi accesati de metodele definite in cadrul aceleiasi clase sau de metodele definite in cadrul claselor care mostenesc clasa data);

-internal (specificatorul internal este utilizat pentru a declara membrii care sunt cunoscuti in toate fisierele dintr-un asamblaj, insa nu in afara asamblajului).

-protected internal (membrii sunt vizibili atat in clasele care mostenesc clasa in care sunt definiti acesti membrii cat si in cadrul grupului de fisiere care formeaza asamblajul).

Observatie: nu exista specificatori de tipul public internal sau private internal.

Exemplul 2. Utilizarea specificatorilor de acces

```
using System;
class Persoana
{
    protected string nume;
    protected string prenume;
     public Persoana(string nm, string pnm)
     {
     nume = nm;
     prenume = pnm;
     }
 }
     class Angajat : Persoana
     {
     private int anulAngajarii;
     public Angajat(string nm, string pnm, int anang )
       : base(nm, pnm)
     {
     anulAngajarii = anang;
     }
     public void
       AfiseazaNumeleIntregSiAnulAngajarii()
     {
     Console.WriteLine("Angajat: {0} {1} {2}", nume,
       prenume, anulAngajarii);
     }
     }
```

```
class NameApp
     {
     public static void Main()
     {
     Angajat el = new Angajat("Popescu", "Ion",
       1983);
     el.AfiseazaNumeleIntregSiAnulAngajarii();
     }
     }
```

Rezultat:

Angajat: Popescu Ion 1983

Modul de transfer al parametrilor catre metode

Ca parametri pentru metode pot fi utilizati tipurile valorice (int, double, etc.), insa pot fi transmise si obiecte.

Exemplul 3. Transmiterea obiectelor ca parametri pentru metode

```
using System;
class Dreptunghi
{
    int l, L;
    int arie;
        public Dreptunghi(int i, int j)
    {
        l = i;
        L = j;
        arie = l * L;
    }
    public bool Congruent(Dreptunghi obiect)
    {
        if ((obiect.l == l) & (obiect.L == L))
            return true;
        else
            return false;
    }
    public bool Echivalent(Dreptunghi obiect)
    {
        if (obiect.arie == arie)
            return true;
        else
            return false;
    }
```

```
class Transmob
   {
      public static void Main()
      {
         Dreptunghi obiect1 = new Dreptunghi(2, 3);
         Dreptunghi obiect2 = new Dreptunghi(2, 3);
         Dreptunghi obiect3 = new Dreptunghi(1, 6);
         Console.WriteLine("Afirmatia: <<obiectul1
      este congruent cu obiectul2>> este {0}",
      obiect1.Congruent(obiect2));
         Console.WriteLine("Afirmatia << obiectul1
      este congruent cu obiectul3>> este  {0}",
      obiect1.Congruent(obiect3));
         Console.WriteLine("Afirmatia <<obiectul1 este
      echivalent cu obiectul3>> este {0}",
      obiect1.Echivalent(obiect3));
      }
   }
}
```

Rezultat:

Afirmatia: <<obiectul1 este congruent cu obiectul2>> este True

Afirmatia << obiectul1 este congruent cu obiectul3>> este False

"Afirmatia <<obiectul1 este echivalent cu obiectul3>> este True

In C# exista doua modalitati de transmitere al parametrilor catre metode si anume: *transferul prin valoare* si *transferul prin referinta*. In mod implicit, tipurile valorice sunt transferate metodelor prin valoare, in timp ce obiectele sunt transferate prin referinta.

Pentru a vedea care este diferenta dintre aceste doua modalitati sa consideram mai intai urmatoarele secvente de cod:

a) int a; int b; a=10; b=a;

b) Dreptunghi ob1; Dreptunghi ob2; ob1=new Dreptunghi(2,7); ob2=ob1;

In cazul a) se declara mai intai doua variabile a si b. Prima dintre ele se initializeaza cu valoarea 10 dupa care si prima variabila primeste aceeasi valoare. Asadar au fost create doua variabile, ambele au valoarea 10.

In cazul b) se declara doua variabile ob1 si ob2. Apoi, utilizand operatorul new, se creaza o imagine fizica a unui obiect care este referit prin intermediul variabilei ob1. Ultima instructiune face ca si ob2 sa refere acelasi obiect. Asadar a fost creat un singur obiect referit prin intermediul a doua variabile ob1 si ob2.

Aceasta diferenta face ca cele doua modalitati de transfer (prin valoare si respectiv prin referinta) ale parametrilor sa aiba nuante diferite. Astfel:

-*transferul prin valoare*: Metoda copiaza valoarea parametrului efectiv in parametrul formal al subrutinei. Modificarile aduse parametrului subrutinei nu vor modifica valoarea parametrului efectiv.

-*transferul prin referinta*: Se transmite parametrului formal o referinta a parametrului efectiv si nu valoarea acestuia. In interiorul subrutinei, referinta este utilizata pentru accesul la parametrul efectiv. Asadar modificarile parametrului formal vor afecta si parametrul efectiv.

Exemplul 4. Transferul prin valoare si transferul prin referinta

```
using System;
class Test
{
    public int a, b;

    public Test(int i, int j)
    {
        a=i;
        b=j;
    }
}
class DemoReferintaValoare
{
    public static void NoChange(int i, int j)
    {
        i = i + j;        j = -j;
    }
    public static void Change(Test obiect)
    {
        obiect.a = obiect.a + obiect.b;
        obiect.b = -obiect.b;
    }
```

```
public static void Main()
    {
        Test ob = new Test(10,20);
        Console.WriteLine("a={0} si b={1}
      inainte de apelul metodei
      NoChange",ob.a,ob.b);

        NoChange(ob.a,ob.b);

        Console.WriteLine("a={0} si b={1}
      dupa apelul metodei NoChange", ob.a,
      ob.b);
        Console.WriteLine("a={0} si b={1}
      inainte de apelul metodei Change",
      ob.a, ob.b);

        Change(ob);

        Console.WriteLine("a={0} si b={1}
      dupa apelul metodei Change", ob.a,
      ob.b);
     }
}
```

Rezultat:

a=10 si b=20 inainte de apelul metodei NoChange
a=10 si b=20 dupa apelul metodei NoChange
a=10 si b=20 inainte de apelul metodei Change
a=30 si b=-20 dupa apelul metodei Change

Utilizarea modificatorilor ref si out

In mod implicit tipurile valorice sunt transferate catre metode prin valoare. Insa acest comportament poate fi modificat cu ajutorul cuvintelor cheie ref si out.

Modificatorul de parametrii ref:

-se utilizeaza atunci cand se doreste ca o anumita metoda sa modifice parametrii efectivi;

-forteaza in C# transferul prin referinta in detrimentul transferului prin valoare;

-apare atat in declaratia cat si in apelul metodei. Parametrul transferat cu ref trebuie sa aiba o valoare atribuita inainte de apel. Asadar, nu se poate apela o metoda care initializeaza un parametru avand modificatorul ref.

Exemplul 5. Utilizarea parametrului ref

```
using System;
class DemoShimb
{
    public static void Schimb(ref int i, ref int j)
    {
        int t;
        t = i;
        i = j;
        j = t;
    }
    public static void Main()
    {
        int x=10, y=20;
        Console.WriteLine("x={0} si y={1} inainte de apelul metodei Schimb",x,y);
        Schimb(ref x,ref y);
        Console.WriteLine("x={0} si y={1} dupa apelul metodei Schimb", x, y);
    }
}
```

Rezultat:

x=10 si y=20 inainte de apelul metodei Schimb

x=20 si y=10 dupa apelul metodei Schimb

Modificatorul de parametrii out:

-se utilizeaza atunci cand se doreste intoarcerea de catre metoda a mai multor valori mediului apelant. O instructiune return intoarece o singura valoare. Daca se doreste intoarcerea spre exemplu a doua valori atunci problema se rezolva cu ajutorul modificatorului out.

-apare atat in declaratia cat si in apelul metodei;

-intoarce o valoare dintr-o metoda. Nu este necesar ca variabila utilizata ca parametru out sa fie initializata inainte de apelul metodei. Metoda va da acesteia o valoare. Mai mult, in corpul metodei, un parametru out este considerat ca neinitializat.

Exemplul 6. Utilizarea parametrului out

```
using System;
class Dreptunghi
{
    int l;
    int L;
    public Dreptunghi(int i, int j)
    {
        l = i;
        L = j;
    }
    public int InfoDreptunghi_si_Arie(out
     bool patrat)
    {
        if (l == L)
            patrat = true;
        else
            patrat = false;
        return l * L;
    }
}
```

```
class DemoDreptunghi
{
    public static void Main()
    {
        int aria;
        bool p;
        Dreptunghi ob = new Dreptunghi(5, 10);
        aria = ob.InfoDreptunghi_si_Arie(out p);
        Console.WriteLine("Dreptunghiul este
     patrat: {0}",p);
        Console.WriteLine("Aria figurii
     geometrice considerate este {0}",aria);
    }
}
```

Rezultat:

Dreptunghiul este patrat: False

Aria figurii geometrice considerate este 50

Utilizarea unui numar variabil de parametri

La crearea unei metode, numarul parametrilor transmisi metodei este in general cunoscut. Exista situatii in care acest numar este necunoscut.

In acest caz se utilizeaza modificatorul params pentru a declara un tablou de parametri. Acest tablou poate contine 0, 1 sau mai multi parametri.

In cazul in care o metoda are atat parametri obisnuiti cat si un parametru params acesta din urma trebuie sa fie ultimul din lista de parametri.

In plus, nu poate exista mai mult de un parametru de tipul params.

Exemplul 7.Programul de mai jos calculeaza maximul dintr-o secventa de valori

```
using System;
class Maxim
{
    public int ValMax(params int [  ] nume)
    {
        int m;
        if (nume.Length == 0)
        {
            Console.WriteLine("Eroare: nu
      sunt parametri");
            return 0;
        }
else
        {
            m = nume[0];
            for (int i = 0; i < nume.Length; i++)
            {
                if (nume[i] > m)
                    m = nume[i];
            }
            return m;
        }
    }
}
```

```
class DemoMaxim
{
    public static void Main()
    {
        int max;
        Maxim ob = new Maxim();
        int[ ] tablou = { 0, 3, 5, 7, -23, 44 };
        max=ob.ValMax(tablou);
        Console.WriteLine("Maximul este {0}",max);

        int a = 7, b = 22;
        max = ob.ValMax(a, b);
        Console.WriteLine("Maximul dintre {0} si {1}
      este: {2}", a,b, max);

    }
}
```

Rezultat:
Maximul este 44
Maximul dintre 7 si 22 este: 22

Supraincarcarea metodelor

Una dintre cele mai interesante facilitati oferite de C# o reprezinta supraincarcarea metodelor.

Definitie. Spunem ca metodele sunt supraincarcate atunci cand doua sau mai multe metode din cadrul aceleiasi clase au acelasi nume.

Observatii: -Daca metodele sunt supraincarcate atunci declaratiile parametrilor lor difera. Metodele au un numar diferit de parametri sau acelasi numar de parametrii insa de tipuri diferite. Pe scurt, metodele nu au aceeasi *signatura.* *Signatura* unei metode reprezinta numele unei metode si lista parametrilor acesteia.

-Signatura unei metode nu include tipul rezultatului intors. Daca metodele au acelasi nume, acelasi numar si tip de parametrii insa tipul rezultatului intors este diferit atunci se produce o eroare. Compilatorul nu dispune de suficienta informatie pentru a decide care metoda sa aleaga.

-Prin supraincarcarea metodelor, limbajul C# implemeteaza conceptul de *polimorfism*. Altfel spus, C# implementeaza *paradigma "o singura interfata, mai multe metode*". In limbajele care nu permit supraincarcarea, exista mai multe versiuni ale aceleiasi functii. Spre exemplu, in C, functia abs( ) intoarce valoarea absoluta a unui intreg, labs( ) valoarea absoluta a unui intreg lung, iar fabs( ) valoarea absoluta a unui float. In C# toate metodele care calculeaza valoarea absoluta au acelai nume Abs( ). Este misiunea compilatorului sa aleaga care metoda a lui Abs( ) trebuie sa o foloseasca. Asadar, utilizand conceptul de supraincarcare a metodelor, mai multe metode au fost comprimate in una singura.

-Exista situatii in care tipurile parametrilor formali nu coincid cu tipurile parametrilor efectivi. In acest caz se realizeaza o conversie automata de tip. Versiunea metodei alese este aceea pentru care setul de parametrii formali este cel mai apropiat de setul de parametri efectivi.

Exemplul 8: Supraincarcarea metodelor

```
using System;
class Overload
{
    public void Ovload()
    {
        Console.WriteLine("Nici un parametru");
    }
    public void Ovload(int a)
    {
        Console.WriteLine("Un parametru de tip int:
      a={0}",a);
    }
    public void Ovload(double a)
    {
        Console.WriteLine("Un parametru de tip
      double: a={0}", a);
    }
    public void Ovload(ref double a)
    {
        Console.WriteLine("Un parametru de tip ref
      double: a={0}", a);
    }
    public int Ovload(int a, int b)
    {
        Console.WriteLine("Doi parametrii de tip int:
      a={0} b={1}", a, b);
        return a + b;
    }
    public double Ovload(double a, double b, double c)
    {
        Console.WriteLine("Trei parametrii de tip double:
      a={0}, b={1}, c={2}", a,b,c);
        return a*b*c;
    }
}

class DemoOverload
{
    public static void Main()
    {
        double x = 3.1;
        Overload ob = new Overload();
        ob.Ovload();
        ob.Ovload(2);
        ob.Ovload(2.1);
        ob.Ovload(ref x);
        ob.Ovload(2, 3);
        ob.Ovload(2, 3, 4);
    }
}
```

Rezultat:

Nici un parametru
Un parametru de tip int: a=2
Un parametru de tip double: a=2,1
Un parametru de tip ref double: a=3,1
Doi parametrii de tip int: a=2 b=3
Trei parametrii de tip double: a=2, b=3, c=4

Exemplul 9: Supraincarcarea metodelor

```
using System;
class Overload{
    public double Aria(double r)    {
        double A=4*(Math.Atan(1))*r*r;
        Console.WriteLine("Aria cercului avand raza r={0} este A={1:#.##}", r, A);
        return A;
    }
    public double Aria(double b, double h)  {
        double A;
        A = h * b/2;
        Console.WriteLine("Aria triunghiului avand baza b={0} si inaltimea h={1} este A={2}",b,h,A);
        return A;
    }
    public double Aria(double b, double B, double h)   {
        double A;
        A = (b+B) * h / 2;
        Console.WriteLine("Aria trapezului avand baza mica b={0}, baza mare B={1} si inaltimea h={2} este A={3}",
       b, B, h, A);
        return A;
    }
}
class DemoOverload{
      public static void Main()
    {
        Overload ob = new Overload();
        ob.Aria(3);         ob.Aria(2,3);       ob.Aria(2,3,4);
    }
}
```

Rezultat:

Aria cercului avand raza r=3 este A=28,27
Aria triunghiului avand baza b=2 si inaltimea h=3 este A=3
Aria trapezului avand baza mica b=2, baza mare B=3 si inaltimea h=4 este A=10

Exemplul 10. Supraincarcarea constructorilor

```
using System;                          //Ca orice alte metode, constructorii pot fi si ei supraincarcati
class Point {
    public double x;
    public double y;
    public Point()    {    }
    public Point(double a)    {   x = a;  }
    public Point(double a, double b)    {   x = a;    y = b;   }
}
class Segmdr{
    public static void Main()    {
        Point punct1 = new Point();
        punct1.x = 3;     punct1.y = 5;
        Point punct2 = new Point(10);
        punct2.y = 5;
        Point punct3 = new Point(2,3);
        double dist;
        dist = Math.Sqrt((punct1.x - punct2.x) * (punct1.x - punct2.x) + (punct1.y - punct2.y) * (punct1.y - punct2.y));
        Console.WriteLine("Distanta dintre punctele ({0},{1}) si ({2},{3}) este: {4:#.##}", punct1.x, punct1.y, punct2.x,
      punct2.y, dist);
        dist = Math.Sqrt((punct1.x - punct3.x) * (punct1.x - punct3.x) + (punct1.y - punct3.y) * (punct1.y - punct3.y));
        Console.WriteLine("Distanta dintre punctele ({0},{1}) si ({2},{3}) este: {4:#.##}", punct1.x, punct1.y, punct3.x,
      punct3.y, dist);
        dist = Math.Sqrt((punct2.x - punct3.x) * (punct2.x - punct3.x) + (punct2.y - punct3.y) * (punct2.y - punct3.y));
        Console.WriteLine("Distanta dintre punctele ({0},{1}) si ({2},{3}) este: {4:#.##}", punct2.x, punct2.y, punct3.x,
      punct3.y, dist);
    }
}
```

Rezultat:
Distanta dintre punctele (3,5) si (10,5) este: 7
Distanta dintre punctele (3,5) si (2,3) este: 2,24
Distanta dintre punctele (10,5) si (2,3) este: 8,25

Metoda Main

Pana acum a fost utilizata o singura forma a metodei Main( ). Exista insa mai multe forme ale acesteia datorate supraincarcarii. Astfel:

-Metoda Main( ) poate intoarce o valoare in procesul apelant (in sistemul de operare). In acest caz se utilizeaza urmatoarea forma a metodei Main( ):

```
public static int Main( );
```

De regula, valoarea intoarsa de Main( ) indica daca programul s-a terminat normal sau ca urmare a unei conditii de eroare. Prin conventie, daca valoarea intoarsa este 0 atunci programul s-a terminat normal. Orice alta valoare indica faptul ca s-a produs o eroare.

-Metoda Main( ) poate accepta parametrii. Acesti parametrii se mai numesc parametrii in linia de comanda. Un parametru in linia de comanda reprezinta o informatie care urmeaza imediat dupa numele programului in linia de comanda utilizata la executia acestuia. Pentru a primi acesti parametri se utilizeaza urmatoarele forme ale metodei Main( ):

```
public static void Main(string [ ] args)
public static int Main(string [ ] args)
```

Prima intoarce void iar cea de-a doua o valoare intreaga.

Exemplele 11 si 12. Metoda Main() cu parametri

```
using System;
class Descompunere
{
    public static void Main(string [] args)
    {
        int n;
        if (args.Length<1 )
       {
       Console.WriteLine("Introduceti un numar
       natural de la tastatura");
        n=int.Parse(Console.ReadLine());
        }
        else
        {
        n = int.Parse(args[0]);
        }
        int count = 2;
        Console.Write("{0}=", n);
        while (count <= n)
        {
            while (n % count == 0)
            {
                n = n / count;
                Console.Write("{0} ", count);
            }
            count++;
        }
    }
}
```

Rezultat: Programul descompune in factori primi un intreg introdus de la tastatura. Dupa caz, este utilizata metoda Main() cu sau fara parametri.

```
using System;

class TestDemo
{
    public static void Main(string [] args)
    {


Console.WriteLine("Sunt {0} parametrii ", args.Length);
   if (args.Length > 0)
    {
    Console.WriteLine("Acestia sunt:");
    }
   for (int i = 0; i < args.Length; i++)
            Console.WriteLine(args[i]);
    }
}
```

Rezultat:

Programul returneaza parametrii introdusi in linia de comanda

Cuvantul cheie static

In mod obisnuit, un membru al unei clase trebuie sa fie accesat utilizand o instanta a acelei clase. Spre exemplu, in cel de-al doilea program din paragraful suparaincarcarea metodelor pentru a putea fi folosita metoda Aria( ) in blocul metodei Main( ) a fost creat un obiect ob de tipul clasei Overload.

Exista insa posibilitatea ca un membru al unei clase sa fie utilizat fara a depinde de vreo instanta. In acest caz, membrul respectiv este declarat ca static. Atat metodele cat si variabilele pot fi declarate statice.

Variabilele declarate ca static sunt variabile globale. La declararea unei instante a clasei, variabilele static nu sunt copiate. Ele sunt partajate de toate instantele clasei. Variabilele static sunt initializate la incarcarea clasei in memorie. Daca nu se specifica in mod explicit o valoare de initializare atunci o variabila static se initializeaza cu zero daca este tip numeric, cu null in cazul tipurilor de referinta si respectiv false pt tipul bool.

Diferenta dintre o metoda statica si o metoda obisnuita este ca metoda statica poate fi apelata fiind prefixata de numele casei din care face parte, fara crearea unei instante a clasei.

Exemplul 13. Initializarea si utilizarea variabilelor statice

```
using System;
class StaticD
{
    public static int a;
    public int b;
}
class StaticDemo
{
    public static void Main()
    {
        Console.WriteLine("Valoarea initiala a variabilei StaticD.a este {0}", StaticD.a);
        StaticD obj = new StaticD();
        obj.b = 10;
        //obj.a=20;  //fiind globala, variabila a poate fi initializata in acest mod
        StaticD.a = 20; //in schimb poate fi initializata manual sau printr-un constructor sau vreo metoda
        Console.WriteLine("Valoarea variabilei Static.a este {0}. Valoarea variabilei obj.b este {1}",
      StaticD.a, obj.b);
    }
}
```

Rezultat:

Valoarea initiala a variabilei StaticD.a este 0

Valoarea variabilei Static.a este 20. Valoarea variabilei obj.b este 10

Exemplul 14. Utilizarea metodelor statice

```
using System;
class Overload
{
    public static double Aria(double r)    {
        double A = 4 * (Math.Atan(1)) * r * r;
        Console.WriteLine("Aria cercului avand raza r={0} este A={1:#.##}", r, A);
        return A;
    }
    public static double Aria(double b, double h)   {
        double A;
        A = h * b / 2;
        Console.WriteLine("Aria triunghiului avand baza b={0} si inaltimea h={1} este A={2}", b, h, A);
        return A;
    }
    public static double Aria(double b, double B, double h)   {
        double A;
        A = (b + B) * h / 2;
        Console.WriteLine("Aria trapezului avand baza mica b={0}, baza mare B={1} si inaltimea h={2} este A={3}",
      b, B, h, A);
        return A;
    }
}
class DemoOverload
{
    public static void Main()    {
        Overload.Aria(3); Overload.Aria(2, 3); Overload.Aria(2, 3, 4);
    }
}
```

Rezultat:

Aria cercului avand raza r=3 este A=28,27
Aria triunghiului avand baza b=2 si inaltimea h=3 este A=3
Aria trapezului avand baza mica b=2, baza mare B=3 si inaltimea h=4 este A=10

Supraincarcarea operatorilor

Definitie. Procesul de redefinire a unui operator in contextul dat de o anumita clasa nou creata poarta numele de *supraincarcarea operatorilor*.

Observatii: -Supraincarcarea operatorilor este utila in numeroase aplicatii. Acest proces permite integrarea unei clase definite de utilizator in mediul de programare. Spre exemplu, daca o clasa defineste o lista de elemente atunci se poate utiliza operatorul + pentru adaugarea unui nou element in acea lista;

-La suparaincarcarea unui operator, semnificatia initiala a acestuia se conserva. Supraincarcarea reprezinta adaugarea unei noi operatii specifice unei anumite clase. Asadar, in cazul exemplului de mai sus, supraincarcarea lui + pentru adaugarea elementelor intr-o lista nu schimba semnificatia sa (care este de adunare) in contextul numerelor intregi;

- Pentru supraincarcarea unui operator se utilizeaza cuvantul cheie operator pentru a defini o metoda operator.

Formele generale pentru operatorii unari si respectiv opearatorii binari sunt:

```
public static tip-rez operator op(tip-param operand)
{ //operatii;}
 public static tip-rez operator op(tip-param1 operand, tip-param2 operand)
{ //operatii;}
```

unde: -operatorul supraincarcat +, -, etc. va substitui op;

-tip-rez reprezinta tipul valorii intoarse de operatia op. Valoarea intoarsa poate fi de orice tip, insa adeseori este de acelasi tip cu clasa pentru care are loc supraincarcarea operatorului;

-in cazul operatorului unar, operand trebuie sa fie de acelasi tip cu clasa pentru care se redefineste operatorul. In cazul operatorului binar, cel putin unul din operanzi (operand1 sau operand2) trebuie sa fie de acelasi tip cu clasa pentru care se redefineste operatorul.

-parametrii operatorilor nu pot utiliza modificatorii ref si out.

Exemplul 15. Supraincarcarea operatorilor binari

```
using System;
class Rlaen
{
   public int n;
   double[ ] vector;

   public Rlaen(int dimensiune)
   {
      n = dimensiune;
      vector=new double[n];
      for (int i = 0; i < vector.Length; i++)
         vector[i] = 0;
   }
   public Rlaen(int dimspatiu, params double[ ] v)
   {
      n = dimspatiu;
      vector = new double[n];
      for (int i = 0; i < vector.Length; i++)
         vector[i] = v[i];
   }
   public static Rlaen operator +(Rlaen op1, Rlaen op2)
   {
      if (op1.n == op2.n)
      {
         Rlaen rezultat = new Rlaen(op1.n);
         for (int i = 0; i < op1.vector.Length; i++)
         {
            rezultat.vector[i] = op1.vector[i] + op2.vector[i];
         }
         return rezultat;
      }
      else
      {
         Console.WriteLine("Cei doi operanzi nu au aceeasi
      dimensiune. Operatia nu se poate realiza");
         Rlaen rezultat = new Rlaen(op1.n);
         return rezultat;
      }
   }
   public static Rlaen operator -(Rlaen op1, Rlaen op2)
   {

      if (op1.n == op2.n)
      {
         Rlaen rezultat = new Rlaen(op1.n);
         for (int i = 0; i < op1.vector.Length; i++)
         {  rezultat.vector[i] = op1.vector[i] - op2.vector[i];  }
         return rezultat;
      }
      else
      {
         Console.WriteLine("Cei doi operanzi nu au aceeasi dimensiune.
      Operatia nu se poate realiza");
         Rlaen rezultat = new Rlaen(op1.n);
         return rezultat;
      }
   }
   public void show()
   {
      Console.Write("(");
      for (int i=0; i<vector.Length-1; i++)
      Console.Write("{0:#.##},",vector[i]);
      Console.Write("{0:#.##})", vector[vector.Length-1]);
      Console.WriteLine();
   }
}
class RlaenDemo
{
   public static void Main()
   {
      Rlaen a = new Rlaen(4, 1, 2, 3, 4);
      Rlaen b = new Rlaen(4, 4, 3,2,1);
      Rlaen c = new Rlaen(4);
      Console.Write("a=");     a.show();
      Console.Write("b=");     b.show();
      c = a + b;   Console.Write("a+b=");   c.show();
      c = a - b;   Console.Write("a-b=");  c.show();
   }
}
```

In exemplul anterior, au fost supraincarcati operatorii binari + si – pentru o clasa care reprezinta din punct de vedere matematic spatiul R^n. Cei doi operanzi au acelasi tip.

Exista posibilitatea de a supraincarca operatori pentru care tipurile operanzilor sunt diferite. Spre exemplu, in clasa Rlaen din exemplul anterior putem considera metodele * (alaturate) pentru care unul din operanzi este de tip double, iar celalalt este un obiect din clasa Rlaen. Din punct de vedere matematic, am introdus cele doua operatii care organizeaza spatiul R^n ca un spatiu vectorial peste R, operatia interna de adunare si operatia externa de inmultire cu scalari.

Un aspect interesant este acela ca o operatie redefinita, poate la randul ei sa fie suparaincarcata. Este cazul operatorului * care este definit in doua moduri.Din punct de vedere matematic oricare dintre cele doua metode conduce la acelasi rezultat. Bineinteles aceasta nu este o regula.

Toate metodele utilizate in exemplul anterior au creat in blocul lor un obiect instanta a clasei Rlaen.

```
public static Rlaen operator *(double l, Rlaen op2)
   {
         Rlaen rezultat = new Rlaen(op2.n);
         for (int i = 0; i < op2.vector.Length; i++)
         {
            rezultat.vector[i] = l* op2.vector[i];
         }
         return rezultat;



   }
   public static Rlaen operator *(Rlaen op2, double l)
   {
      Rlaen rezultat = new Rlaen(op2.n);
      for (int i = 0; i < op2.vector.Length; i++)
      {
         rezultat.vector[i] = l * op2.vector[i];
      }
      return rezultat;
   }
```

Supraincarcarea operatorilor unari

In cazul operatorilor unari, singurul operand trebuie sa fie acelasi tip cu clasa pentru care se redefineste operatorul, in cazul nostru un obiect instanta a clasei Rlaen.

Alaturat sunt exemplificate trei metode. Prima dintre ele nu modifica operandul, intrucat in blocul metodei este creat un nou obiect care este intors ca rezultat. In schimb, celelalte doua metode modifica parametrul op1.

```
public static Rlaen operator -(Rlaen op1)
{

    Rlaen rezultat = new Rlaen(op1.n);
    for (int i = 0; i < op1.vector.Length; i++)
    { rezultat.vector[i] = -op1.vector[i]; }
    return rezultat;
}

public static Rlaen operator ++(Rlaen op1)
{

    for (int i = 0; i < op1.vector.Length; i++)
    { op1.vector[i] = op1.vector[i]+1; }
    return op1;
}

public static Rlaen operator --(Rlaen op1)
{

    for (int i = 0; i < op1.vector.Length; i++)
    { op1.vector[i] = op1.vector[i] - 1; }
    return op1;
}
```

Supraincarcarea operatorilor relationali

Operatorii relationali, cum ar fi ==, <, <=, pot de asemenea fi supraincarcati. De regula, un operator relational supraincarcat va intoarce o valoare true sau false, desi nu este neaparat. Insa daca alegeti sa intoarceti alt tip se pot crea confuzii.

Exista o restrictie importanta referioare la redefinirea operatorilor relationali si anume aceea ca trebuie redefiniti in perechi. Adica, daca este redefinit operatorul < atunci trebuie redefinit si operatorul >. Perechile sunt urmatoarele: (==, !=); (<,>); (<=, >=).

Alaturat sunt redefiniti operatorii == si != pectru clasa Rlaen.

Exista cativa operatori (altii decat cei relationali) care nu pot fi supraincarcati, si anume: &&, ||, [ ], ( ), new, is, sizeof, ?,. =, +=, -=.

Cuvintele cheie true sau false pot fi utilizate ca operatori unari in scopul redefinirii.

```
public static bool operator ==(Rlaen op1, Rlaen op2)
{    if (op1.n == op2.n)
     {    bool[ ] compar = new bool[op1.vector.Length];
          for (int i = 0; i < op1.vector.Length; i++)
          {    compar[i]=(op1.vector[i]== op2.vector[i]);
               if (!compar[i])
               {  return false;   break; }
          }

        return true;
     }
     else
     {   Console.WriteLine("Cei doi operanzi nu au aceeasi
     dimensiune. Operatia nu se poate realiza");
        return false; }
}
```

```
public static bool operator !=(Rlaen op1, Rlaen op2)
{   if (op1.n == op2.n)
    {   bool[] compar = new bool[op1.vector.Length];
        for (int i = 0; i < op1.vector.Length; i++)
        {   compar[i] = (op1.vector[i] != op2.vector[i]);
            if (compar[i])
            { return true;   break;  }
        }
        return false;
    }
    else
    {   Console.WriteLine("Cei doi operanzi nu au aceeasi
    dimensiune. Operatia nu se poate realiza");
        return false;   }
}
```