Clase.Obiecte.Metode

CLASE SI OBIECTE -introducere

- O clasă descrie unul sau mai multe obiecte care pot fi precizate printr-un set uniform de date şi cod (funcţionalitate);
- Orice obiect are memoria sa proprie unde se păstrează valorile tuturor datelor sale;
- Clasa defineşte caracteristicile si comportarea obiectelor. Reprezinta un set de planuri care precizeaza cum poate fi construit un obiect;
- Orice obiect are un tip; un obiect este o *instanţă a unei clase*

Un obiect este caracterizat de:

- nume – un identificator;
- contine date de un anume tip si metode (servicii, operaţii) – funcţii care accesează datele obiectului.

Clase - proiectare

Definiţia unei clase presupune:

a) combinarea datelor pe care clasa le contine cu operaţiile (codul) care prelucreaza aceste date;

b) ascunderea informaţiei;

C# poate defini mai multe categorii particulare de date si membrii care contin cod, printre care: variabile, constante, proprietati, metode, constructori, destructori, indexari, enumerari, evenimente, delegari.

De regula, modul de structurare a datelor nu este cunoscut: datele sunt declarate intr-o secţiune “privată” a clasei. Accesul la date, pentru consultare, modificare, etc. se face prin intermediul metodelor clasei si proprietatilor clasei; acestea sunt declarate intr-o secţiune “publică” a clasei**.** Pot exista date si metode publice sau private, decizia este a programatorului. Datele si codul pot fi ascunse de unde si conceptul de incapsulare (incapsulare=combinare+ascundere).

Avantaje

- **Combinarea datelor:**
  - defineşte clar ce structuri de date sunt manevrate şicare sunt operaţiile legale asupra lor;
  - programul capătă modularitate;
  - scade riscul alterării datelor din exterior;
  - facilitează ascunderea informaţiei;
- **Ascunderea informaţiei:**
  - programe mai sigure şi mai fiabile;
  - eliberează clasele utilizator de grija manevrării datelor;
  - previne apariţia erorilor;
- **Încapsulare:**

  Combinare + Ascunderea informaţiei = Protejarea datelor
  - previne apariţia erorilor prin limitarea accesului la date;
  - asigură portabilitatea programelor;
  - facilitează utilizarea excepţiilor;

Clase

- O clasă încapsulează date, metode si alti membrii care acţionează asupra acelor date.
- Sintatic, o clasă se declară astfel:

```
class identificator
{
corpul clasei;
}
```

unde *identificator* reprezinta numele clasei, iar *corpul clasei* contine datele si codul din interiorul clasei. In cazul in care, corpul clasei contine numai variabile si metode atunci acesta are forma:

```
{acces tip var1;
 //…
 acces tip varN;
 acces tip-rez Metoda1(parametri)
   {corpul Metodei1;}
 //…
  acces tip-rez MetodaN(parametri)
    {corpul metodeiN}
  }
```

Exemplul 1. In exemplul de mai jos este definita o clasa si sunt create doua obiecte instanta ale acestei clase.

```
using System;
class Point
{
    public double x;
    public double y;
}
 class Segmdr
{
    public static void Main()
    {
        Point punct1 = new Point();
        Point punct2 = new Point();
        double dist;
        punct1.x = 3;
        punct1.y = 4;
        punct2.x = 5;
        punct2.y = 3;
        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);
    }
}
```

Rezultat:

Distanta dintre punctele (3,4) si (5,3) este: 2,24

Observatii: -Definitia unei clase creaza un nou tip de date. In cazul de mai sus acest nou tip de date se numeste *Point*. Am utilizat acest tip de date pentru a declara doua obiecte.

-Dupa executia instructiunii Point punct1 = new Point(); punct1 va fi o instanta a clasei Point. Orice obiect de tip Point va avea copii proprii ale variabilelor instanta x si y.

-Pentru a referi variabilele instanta se utilizeaza operatorul punct (.). Acesta leaga numele unui obiect de un membru al sau. Instructiunea punct1.x = 3; atribuie variabilei x a instantei punct1 valoarea 3.

-Nu este necesara utilizarea claselor in acelasi fisier. In acest caz se compileaza programul utilizand linia de comanda:

csc numeprog1.cs numeprog2.cs …. numeprogN.cs

unde numeprogr.1cs, numeprog2.cs …. numeprogN.cs reprezinta numele fisierelor care contin clasele necesare compilarii programului. Evident doar unul dintre acestea contine metoda Main().

Metode

Definitie. Metodele sunt subroutine care prelucreaza datele definite in cadrul clasei si pot oferi accesul la acele date.

De regula interactiunea dintre o clasa si celelalte entitati dintr-un program se face prin intermediul metodelor clasei.

Forma generala a unei metode este:

```
acces tip-rez Nume(lista parametri)
{
        //corpul metodei
}
```

unde

acces=modificator de acces care stabileste care dintre celelalte parti ale programului pot apela metoda (este optonal). Daca lipseste, metoda este privata clasei in care a fost declarata. (Modificatorul de acces poate fi: public, protected, private, internal);

tip-rez= tipul rezultatului pe care metoda il intoarce (Ex: void, int, double…);

Nume=numele metodei, diferit de cuvintele cheie;

lista parametri =secventa de perechi separate prin virgula, in care primul element este un tip, iar al doilea un identificator.

Observatii: Exista doua moduri de a reveni dintr-o metoda

-intalnirea acoladei care marcheaza sfarsitul metodei (daca este de tip void);

-executia unei instructiuni return.

Daca metoda returneaza o valoare obtinuta in urma unui calcul sau ca reusita sau esec a unei operatii atunci se utilizeaza urmatoarea instructiune

```
return val;
```

Exemplul 2. In exemplul de mai jos este modificat programul din exemplul 1. Programul contine inca o clasa (clasa Line), iar codul care calcula distanta dintre doua puncte este organizat sub forma unei metode din clasa Line, numita Lung() .

```
using System;
class Point
{
    public double x;
    public double y;
}
class Line
{
    public Point punct1 = new Point();
    public Point punct2 = new Point();
    public double Lung()
    {
        double l;
        l= Math.Sqrt((punct1.x - punct2.x) *
      (punct1.x - punct2.x) + (punct1.y -
      punct2.y) * (punct1.y - punct2.y));
        return l;
    }
}
```

```
class Segmdr
{
    public static void Main()
    {
        Line seg = new Line();
        double dist;
        seg.punct1.x = 3;
        seg.punct1.y = 4;
        seg.punct2.x = 5;
        seg.punct2.y = 3;
        dist=seg.Lung();
          Console.WriteLine("Distanta dintre
      punctele ({0},{1}) si ({2},{3}) este:
      {4:#.##}", seg.punct1.x, seg.punct1.y,
      seg.punct2.x, seg.punct2.y, dist);
    }
}
```

Rezultat:

Distanta dintre punctele (3,4) si (5,3) este: 2,24

Metode.Utilizarea parametrilor

Parametru efectiv (argument)= valoare transmisa unei metode cand aceasta este invocata.

Parametru formal = variabila in corpul metodei care primeste valoarea parametrului efectiv.

Observatii:- parametrii formali se declara in interiorul parantezelor, dupa numele metodei;

-parametrii formali au domeniul de valabilitate corpul metodei;

-daca sunt folositi mai multi parametrii atunci fiecare isi specifica tipul. Acesta din urma poate fi diferit de tipul celorlalti.

Programul alaturat (Exemplul 3) testeaza daca primul intreg, notat prin d, este divizor al celui de-al doilea intreg. Ambii parametrii sunt introdusi de la tastatura.

Exemplul 3. a si b sunt paramerii formali, in timp ce d si m sunt parametri efectivi.

```
using System;
class Divizor
{
   public bool EsteDivizor(int a, int b)
   {
      if ((b % a) == 0)
         return true;
      else
         return false;
   }
}
class MyDiv
{
   public static void Main()
   {
      int d, m;
      Console.WriteLine("Programul testeaza daca d este
      divizor al lui m");
      Console.Write("Introduceti nr intreg d=");
      d = Convert.ToInt32(Console.ReadLine());
      Console.Write("\n Introduceti numarul intreg m=");
      m = int.Parse(Console.ReadLine());
      Divizor x=new Divizor();
      x.EsteDivizor(d, m);
      if (x.EsteDivizor(d, m))
        Console.WriteLine("{0} este divizor al lui {1}", d, m);
      else
        Console.WriteLine("{0} nu este divizor al lui {1}", d,
      m);
    }
}
```

Rezultat:

In functie de intregii introdusi de la tastatura, obtinem doua tipuri de mesaje.

Constructori

Variabilele instanta pot fi initializate manual (in Exemplul 1 variabilele x si y pentru ambele obiecte). Aceasta practica poate genera erori la initializarea unui camp. O solutie in acest sens o reprezinta utilizarea constructorilor.

Un constructor initializeaza un obiect atunci cand este creat. Au acelasi nume cu clasa din care fac parte. Nu au tip explicit.

Forma generala:

```
acces nume-clasa(parametri)
{
//codul constructorului
}
```

Se utilizeaza pentru atribuirea valorilor initiale pentru variabilele instanta definite in cadrul clasei sau pentru efectuarea altor operatii initiale necesare la crearea unui obiect complet initializat.

C# pune la dispozitie un constructor implicit care initializeaza toate variabilele membru cu zero (pt. tipuri valorice) respectiv cu null (pt. tipuri referinta). (Astfel, in Exemplul 1, imediat dupa crearea obiectelor punct1 si punct2, variabilele x si y ale ambelor obiecte au valorile 0. Apoi acestea sunt reinitializate manual). Daca este definit propriul constructor, cel implicit nu va fi utilizat.

Exemplul 4. Este modificat programul prezentat in exemplul 1 prin includerea unui constructor al clasei Point. La crearea obiectelor punct1 si punct2 se utilizeaza acest constructor care initializeaza variabilele x si y.

```
using System;
class Point
{
    public double x;
    public double y;
    public Point(double a, double b)
    {
        x = a;
        y = b;
     }
}
class Segmdr
{
    public static void Main()
    {
        Point punct1 = new Point(3,4);
        Point punct2 = new Point(5,3);
        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);
    }
}
```

Rezultat:

Distanta dintre punctele (3,4) si (5,3) este: 2,24

Colectarea spatiului neutilizat

Utilizand operatorul new se aloca dinamic memorie libera pentru memorarea obiectelor create. Intrucat memoria nu este infinita, operatorul new poate sa esueze daca memoria este epuizata. Se pune astfel problema recuperarii memoriei din obiecte care nu mai sunt utilizate.

In cazul limbajului C++ se utilizeaza operatorul *delete* pentru a elibera memoria.

Limbajul C# pune la dispozitie o metoda diferita si anume colectarea automata a spatiului neutilizat.

Sistemul de colectare automata recupereaza spatiul ocupat de obiectele care nu mai sunt necesare programului. Daca nu mai exista nici o referinta la un obiect atunci se presupune ca acel obiect nu mai este necesar, iar memoria ocupata de el poate fi eliberata.

Colectarea automata se declanseaza destul de rar pe parcursul executiei programului. Nu se va declansa doar pentru ca exista obiecte care nu mai sunt folosite. Colectarea automata se declanseaza atunci cand se indeplinesc urmatoarele doua conditii: exista o mare necesitate de memorie si exista obiecte care pot fi reciclate.

Colectarea necesita un timp mare. Astfel aceasta se va declansa doar daca este imperios necesar. Nu se poate determina cu exactitate cand are loc colectarea spatiului neutilizat

Destructori

Destructorii au forma generala:

~nume-clasa()
{
// codul destructorului
}

Destructorul este deci declarat similar cu un constructor, fiind insa precedat de caracterul ~(tilda).

Pentru adaugarea unui destructor la o clasa, il includeti ca si pe orice alt membru al clasei. Acesta va fi apelat inainte ca un obiect din acea clasa sa fie reciclat. In interiorul destructorului, se vor specifica actiunile care trebuie executate inainte ca obiectul sa fie distrus.

Destructorul se apeleaza imediat inaintea colectarii automate. Acesta nu va fi apelat cand se iese din domeniul de valabilitate al obiectului. Lucrul acesta difera de C++, unde destructorii sunt apelati cand domeniul de valabilitate se incheie.Nu se poate determina cu exactitate cand se va executa un destructor.

Mai mult, este posibil ca programul dumneavoastra sa se termine inaintea inceperii operatiei de colectare automata, caz in care destructorul nu va fi apelat deloc.

Exemplul 5. Utilizarea destructorilor

```
using System;
class Destructor
{
    int x;
    public Destructor(int i)
    {
        x=i;
    }
    ~Destructor()
    {
        Console.WriteLine("Distrugem "+x);
    }

    public void generator(int i)
    {
        Destructor o = new Destructor(i);
    }

}
class DestrDemo
{
    public static void Main()
    {
        int num;
        Destructor ob=new Destructor(0);
        for (num = 1; num < 10000; num++)
            ob.generator(num);
    }
}
```

Tablouri

Definitie. Un tablou reprezinta o colectie de variabile de acelasi tip, referite prin intermediul unui nume comun.

Observatii:

- La fel ca si in alte limbaje de programare, in C# tablourile pot avea mai multe dimensiuni;
- Tablourile se pot utiliza ca si tablourile din alte limbaje de programare, insa spre deosebire de acestea, in C# tablourile sunt implementate ca obiecte.
- Un avantaj obtinut prin implementarea tablourilor ca obiecte este acela ca spatiul ocupat de obiectele neutilizate poate fi colectat automat.

Declararea unui tablou unidimensional:

tip [ ] nume_tablou = new tip[dimensiune];

unde tip reprezinta tipul de baza al tabloului, nume_tablou reprezinta un identificator care contine referinta la zona de memorie alocata de new, iar dimensiune reprezinta numarul de elemente pe care tabloul le memoreaza. (Ex: int [ ] n=new int[6]; prima parte a instructiunii declara variabila de referinta n la tablou. In partea a doua se utilizeaza operatorul new pentru a aloca dinamic (la executia programului) memorie pentru tablou si pentru a atribui variabilei n o referinta la zona de memorie alocata).

Tabloul fiind un obiect, putem separa declaratia de mai sus in doua parti:

tip [ ] nume_tablou;
nume_tablou= new tip[dimensiune];

Accesarea unui element din tablou:

Un element individual din tablou poate fi accesat utilizand un index. Indexul descrie pozitia unui element din tablou. La fel ca in limbajul C, in C# toate tablourile au indexul primului element egal cu zero. In cazul exemplului de mai sus, primul element este n[0], al doilea este n[1],… cel de-al saselea este n[5].

Exemplul 1. In programul de mai jos sunt determinate cel mai mic element si cel mai mare element ale unui tablou ale carui elemente de tip int sunt initializate manual.

```
using System;
class MinMax
{
    public static void Main()
    {
        int[] n = new int[6];
        int min, max;
        n[0] = 3298;
        n[1] = 8;
        n[2] = -98;
        n[3] = 48;
        n[4] = -298;
        n[5] = -28;
        min = max = n[0];
        for (int i = 1; i < 6; i++)
        {
            if (n[i] < min)
            {
                min = n[i];
            }
            if (n[i] > max)
            {
                max = n[i];
            }
        }
        Console.WriteLine("min={0}, max={1}", min, max);
    }
}
```

Rezultat:

min=-298, max=3298

Initializarea unui tablou:

In programul anterior, valorile elementelor individuale au fost incarcate manual. O modalitate mai simpla este aceea de a realiza aceasta operatie direct de la creare.

Forma generala pentru initializarea unui tablou este:

```
tip [  ] nume_tablou={val1, val2, val3,… valN};
```

unde elementelor nume_tablou[0], nume_tablou[1],… nume_tablou[N-1] ale tabloului le-au fost atribuite valorile val1, val2,…si respectiv valN.

Observatii:

- Cand un tablou se initializeaza in acest mod nu este necesara utilizarea operatorului new;
- Desi redundanta, este corecta utilizarea formei

```
tip [  ] nume_tablou=new tip[ ] {val1, val2, val3,… valN};
```

- De asemenea, putem separa instructiunea de mai sus in doua comenzi:

```
tip [  ] nume_tablou;
nume_tablou=new tip[ ] {val1, val2, val3,… valN};
```

- Este GRESITA insa utilizarea instructiunilor:

```
tip [  ] nume_tablou;
nume_tablou= {val1, val2, val3,… valN};
```

- Este corecta si utilizarea formei

```
tip [  ] nume_tablou=new tip[N] {val1, val2, val3,… valN};
```

insa dimensiunea N a tabloului trebuie sa fie egala cu numarul elementelor initializate. Mai mult N trebuie sa fie un literal. Astfel, este corecta initializarea

```
const int arraySize=5;
int [ ] myArray=new int [arraySize] {0, 2, 4, 6, 8};
```

insa urmatoarele sunt gresite:

```
int arraySize=5;
int [ ] myArray=new int [arraySize] {0, 2, 4, 6, 8};
```

```
const int arraySize=5;
int [ ] myArray=new int [arraySize] {0, 2, 4};
```

Verificarea marginilor

Marginile tablourilor sunt strict verificate. Depasirea sfarsitului de tablou sau utilizarea indicilor negativi genereaza erori la executie. Spre exemplu: programul de mai jos, dupa ce variabila i atinge valoarea 50 genereaza o exceptie de tipul IndexOutOfRangeException si programul se termina:

Exemplul 2. Programul genereaza o eroare la executie.

```
using System;
class ArrayErr
{
    public static void Main()
    {
  int[ ] array = new int[50];
        for (int i = 0; i < 100; i++)
          {
            array[i] = i;
             Console.WriteLine("array[{0}]={1}", i,array[i]);
          }
    }
}
```

Tablouri multidimensionale

Forma generala a unui tablou multidimensional este:

tip [,…,] nume_tablou = new tip[dim1, dim2, … dim N];

Spre exemplu, declaratia de mai jos creaza un tablou tridimensional de elemente de tip long cu dimensiunile 6x7x12

long [, ,] n=new long[6,7,12];

La fel ca in cazul unui tablou unidimensional, initializarea tabloului se poate face fie manual (spre exemplu instructiunea n[3,5,10]=76; atribuie elementului avand indexul [3,5,10] valoarea 76), fie la crearea tabloului. In acest caz, initializarea se face prin includerea listei de initializare a fiecarei dimensiuni intr-o pereche proprie de acolade. Daca avem de-a face cu un tablou bidimensional, pentru care primul index variaza de la 0 la M-1, iar cel de-al doilea de la 0 la N-1, atunci initializarea se face astfel

tip [,] nume_tablou={ {$val_0$0, val_0 1,… val_0N-1}, {val_1 0, val_1 1,… val_1 N-1}, …
{val_{M-1} 0, val_{M-1} 1,… val_{M-1} N-1}};

Observatie: Blocurile sunt separate prin virgule, iar acolada finala este urmata de punct si virgula.

Exemplele 3 si 4

```
/*initializarea si afisarea valorilor
 unui tablou bidimensional pentru care
 prima dimensiune este 2 iar cea de-a doua
 dimensiune este 3 */

using System;
class ArrayErr
{
    public static void Main()
    {
        int[ , ] array = new int[ , ] {{1,2,3},
                                       {3,5,6}};
        for (int i = 0; i < 2; i++)
            for(int j=0; j<3; j++)
        {
            Console.WriteLine("array[{0},{1}]={2}", i,j,
       array[i,j]);
        }
    }
}
```

Rezultat:

```
array[0,0]=1
array[0,1]=2
array[0,2]=3
array[1,0]=3
array[1,1]=5
array[1,2]=6
```

```
/*initializarea si afisarea valorilor
 unui tablou tridimensional avand
 dimensiunile 3, 4 si respectiv 2 */

using System;
class ArrayErr
{
    public static void Main()
    {
  int[, ,] array ={ {{2,3}, {5,6}, {1,-1}, {0,6}},
                    {{4,5}, {6,-3}, {3,-5}, {8,9}},
      {{-2,33}, {15,16},{11,-10}, {10,60}} };
        for (int i = 0; i < 3; i++)
            for(int j=0; j<4; j++)
                for (int k=0; k<2; k++)
                {
                Console.WriteLine("array[{0},{1},{2}]={3}",
       i,j,k, array[i,j,k]);
                }
    }
}
```

Tablouri in scara

In exemplele anterioare in care au fost create tablouri bidimensionale, au fost create asa numitele *tablouri dreptunghiulare*. Adica toate liniile au avut acelasi numar de elemente.

In cazul in care doriti sa creati un tablou bidimensional in care lungimea fiecarei linii sa fie diferita, puteti utiliza asa numitele *tablouri in scara.*

Definitie. Tablourile in scara sunt tablouri cu elementele tablouri.

Forma generala utilizata pentru declararea unui tablou in scara este:

tip[ ] [ ] nume_tablou=new tip [dim] [ ];

unde dim reprezinta numarul de linii din tablou. Liniile nu au fost inca alocate. Acestea se aloca in mod individual, pentru ca lungimea liniei sa varieze. Forma generala este:

nume_tablou[0]=new tip[dim_0];
nume_tablou[1]=new tip[dim_1];
……
nume_tablou[dim-1]=new tip[dim_{dim-1}];

Exemplu: int [ ] [ ] array=new int [2] [ ]; //aceasta secventa de cod aloca memorie pentru
array[0]=new int[3]; //un tablou avand doua linii. Prima linie contine 3
array[1]=new int[7]; //elemente, iar cea de-a doua 7 elemente.

Majoritatea aplicatiilor nu utilizeaza tablourile in scara. Insa aceste sunt utile in unele situatii, spre exemplu atunci cand este necesara memorarea elementelor unei matrici de dimensiuni foarte mari, insa care are doar putine elemente semnificative.

Exemplul 5. Utilizarea tablourilor in scara

```
using System;
class ArrayErr
{
    public static void Main()
    {
        int[][] pasageri =new int[7][];
        pasageri[0]=new int[4]{20,14,8,9};
        pasageri[1] = new int[4] { 14, 7, 14, 8 };
        pasageri[2] = new int[4] { 17, 12, 9, 19 };
        pasageri[3] = new int[4] { 13, 15, 18, 14 };
        pasageri[4] = new int[4] { 20, 19, 18, 20 };
        pasageri[5] = new int[2] { 10, 5 };
        pasageri[6] = new int[2] { 7, 9 };
           for (int i = 0; i < 5; i++)
            for(int j=0; j<4; j++)
                {
                Console.WriteLine("In ziua {0}, cursa {1} a avut {2} pasageri", i+1,j+1, pasageri[i][j]);
                }
           for (int i = 5; i < 7; i++)
               for (int j = 0; j < 2; j++)
               {
                   Console.WriteLine("In ziua {0}, cursa {1} a avut {2} pasageri", i + 1, j + 1, pasageri[i][j]);
               }
    }
}
```

Rezultat: Programul afiseaza numarul de pasageri pe care l-a avut fiecare cursa.

Fiecarui tablou ii este asociata proprietatea Length.

Aceasta contine numarul de elemente pe care tabloul le poate memora.

Exemplul 6. Utilizarea proprietatii Length

```
using System;
class LenghtDemo
{
 public static void Main()
{
 int[] list = { 2, 4, 6, 4, 3, 8, 6, 9 };
 int[][] pasageri =new int[7][];
 pasageri[0]=new int[4]{20,14,8,9};
 pasageri[1] = new int[4] { 14, 7, 14, 8 };
 pasageri[2] = new int[4] { 17, 12, 9, 19 };
 pasageri[3] = new int[4] { 13, 15, 18, 14 };
 pasageri[4] = new int[4] { 20, 19, 18, 20 };
 pasageri[5] = new int[2] { 10, 5 };
 pasageri[6] = new int[2] { 7, 9 };
 Console.WriteLine("Lungimea tabloului list
este {0}", list.Length );
 Console.WriteLine("Lungimea tabloului
pasageri este {0}", pasageri.Length);
 Console.WriteLine("Lungimea tabloului
pasageri[0] este {0}", pasageri[0].Length);
 Console.WriteLine("Lungimea tabloului
pasageri[5] este {0}", pasageri[5].Length);
    Console.Write("Tabloul list:\t");
 for (int i = 0; i < list.Length ; i++)
    Console.Write(list[i] + " ");
 Console.WriteLine();
 }
}
```

Rezultat:

```
Lungimea tabloului list este 8
Lungimea tabloului pasageri este 7
Lungimea tabloului pasageri[0] este 4
Lungimea tabloului pasageri[5] este 2
Tabloul list:2 4 6 4 3 8 6 9
```

Bucla *foreach*

Bucla *foreach* se utilizeaza pentru ciclarea prin elementele unei colectii. O colectie reprezinta un grup de obiecte. C# defineste mai multe tipuri de colectii, unul dintre acestea o reprezinta tablourile.

Forma generala a buclei foreach este:

```
foreach(tip nume_var in colectie)
{instructiuni; }
```

unde tip nume_var specifica tipul si numele unei variabile de iterare care va primi la fiecare iteratie a lui foreach valoarea unui element din colectie. Colectia prin care se face ciclarea este specificata de colectie.

Observatie: Variabila de iterare trebuie sa fie de acelasi tip sau de un tip compatibil cu tipul de baza al colectiei.

Exemplul 7. Utilizarea buclei foreach pentru ciclarea intr-un tablou unidimensional.

```
using System
class ForeachDemo
{
    public static void Main()
    {
        int sum=0;
        int [] n =new int[20];

        for (int i = 0; i < n.Length; i++)
            n[i] = i*i;

        foreach (int x in n)
        {
            Console.WriteLine("valoarea este: "
     + x);
            sum +=x;
        }

        Console.WriteLine("Suma este {0}",
      sum );

    }
}
```

Rezultat:

```
valoarea este: 0
valoarea este: 1
valoarea este: 4
valoarea este: 9
valoarea este: 16
valoarea este: 25
valoarea este: 36
valoarea este: 49
valoarea este: 64
valoarea este: 81
valoarea este: 100
valoarea este: 121
valoarea este: 144
valoarea este: 169
valoarea este: 196
valoarea este: 225
valoarea este: 256
valoarea este: 289
valoarea este: 324
valoarea este: 361
Suma este 2470
```

Observatie: Principala diferenta intre instructiunile for si foreach este accea ca foreach ofera acces doar la citirea elementelor unei colectii. Spre exemplu, instructiunea:

```
int [ ] n = new int[3];
for (int i = 0; i < n.Length; i++)
      n[i] = 5*5;
```

nu poate fi inlocutita cu:

```
 foreach (int x in n)
      x = 5*5;
```

sau orice alta comanda care incearca sa atribuie valori elementelor colectiei.

Stringuri

Stringuri

Unul dintre cele mai importante tipuri de date este tipul string. Tipul string defineste si implementeaza sirurile de caractere. Spre deosebire de alte limbaje de programare unde stringurile sunt reprezentate ca tablouri de caractere, in C# stringurile sunt obiecte. Asadar, tipul string este un *tip de referinta*.

Observatii: -un string poate fi construit prin utilizarea unui literal, spre exemplu:

```
string str="acesta este un sting";
```

sau prin pornind de la un tablou de tip char, spre exemplu:

```
char [ ] charray={'a', 'b', 'w', 'r'};
string str=new string(charray);
```

- clasa string contine mai multe metode care opereaza asupra stringurilor. Spre exemplu:

static string Copy(string str); intoarce o copie a stringului str;

int CompareTo(string str) intoarce o valoare negativa daca stringul care invoca metoda este mai mic decat str, pozitiva daca este mai mare si nula daca stringurile sunt egale;

int indexOf(string str) cauta in stringul apelant subsirul str. Intoarce indicele primei aparitii, sau -1 in caz de esec;

int LastindexOf(string str) cauta in stringul apelant subsirul str. Intoarce indicele ultimei aparitii, sau -1 in caz de esec;

string Substring(int startindex, int len) intoarce un substring al stringului apelant. Parametrul startindex precizeaza indicele de inceput, iar len lungimea subsirului extras;

string [ ] Split(char [ ] separator, int count) intoarce un tablou de tip string avand dimensiunea mai mica sau egala cu count. Tabloul este obtinut prin impartierea stringului apelant in subsiruri. Aceasta operatie se realizeaza la intalnirea separatorului specificat de separator.

-tipul string contine proprietatea Length;

-pentru determinarea valorii unui caracter dintr-un string se utilizeaza un index. Exemplu:

```
string str=“sirdecaractere”;
Console.WriteLine(str[3]);
```

Metoda WriteLine va afisa litera d;

-pentru a verifica daca doua stringuri sunt egale se poate utiliza operatorul ==. In cazul in care operatorul == este aplicat asupra referintelor la obiecte, acesta stabileste daca cele doua referinte refera acelasi obiect. In cazul stringurilor, chiar daca acestea sunt obiecte, operatorul == compara efectiv daca cele doua stringuri sunt egale. Acelasi lucru se intampla si cu operatorul != care compara continutul a doua obiecte de tip string.

-pentru a concatena doua stringuri se utilizeaza operatorul +;

-stringurile nu se pot modifica. Dupa creare, un obiect de tip string nu poate fi modificat. Daca este nevoie de un string care este o variatie a unui string deja existent atunci trebuie creat unul nou care sa contina modificarile dorite.

Exemplul 1. Metode si operatii cu stringuri

```
using System;
class StringDemo
{  public static void Main()
   {   string str1="Acesta este un string";
       string str2=string.Copy(str1);
       string str3 = "Acesta este un alt string";
       for (int i = 0; i < str1.Length; i++)
           Console.Write(str1[i]);
       Console.WriteLine();
       if (str1 == str2)
           Console.WriteLine("str1=str2");
       else
           Console.WriteLine("str1 !=str2");
       if (str1 == str3)
           Console.WriteLine("str1=str3");
       else
           Console.WriteLine("str1 !=str3");
       int result = str1.CompareTo(str3);
           if(result ==0)
               Console.WriteLine("str1 este egal cu
     str3");
           else
               Console.WriteLine("str1 si str3 sunt
     diferite");
        string s = str3.Substring(12, 13);
        Console.WriteLine(s);
   }
}
```

Rezultat:
Acesta este un string
str1=str2
str1 !=str3
str1 si str3 sunt diferite
un alt string

```
using System;
class MinMax
{
   public static void Main()
   {
       string S = "Acesta este un string";
       string subsirS;
       subsirS = S.Substring(0, 8);
       Console.WriteLine(subsirS);
   }
}
```

Rezultat:
Acesta e

```
using System;
class MinMax
{
   public static void Main()
   {
       string S = "Acesta este un string";
       string subsirS = "est";
       int i,j;
       i = S.IndexOf(subsirS);
       j = S.LastIndexOf(subsirS);
       Console.WriteLine("Primul index este {0}",i);
       Console.WriteLine("Ultimul index este {0}", j);
   }
}
```

Rezultat:
Primul index este 2
Ultimul index este 7

Exemplu:

```
using System;
class Citire
{
    public static void Main()
    {
        string [ ] myStringArray;
        char[ ] charArray = { '-' };

        string myString="This-is-my-string";
        myStringArray=myString.Split(charArray,2);
        foreach (string s in myStringArray)
        {
            Console.WriteLine(s);
        }
    }
}
```

Rezultat:

This

is-my-string