

Laborator 4

Instrucțiunea do – while

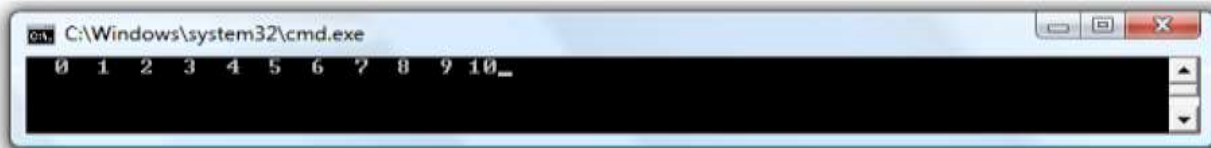
Instrucțiunea **do – while** are sintaxa:

```
do
    Instructiuni;
while (conditie)
```

Se execută **Instructiuni** după care se verifică **conditie**. Dacă aceasta este adevărată, ciclul se reia, altfel ciclul se termină.

Exemplul 32: Asemănător cu exercițiul 28, să se afișeze numerele întregi pozitive ≤ 10

```
using System;
namespace Exemplul_32
{
    class Program
    {
        static void Main(string[] args)
        {
            int n = 0;
            do
            {
                Console.Write("{0,3}", n);
                n++;
            }
            while (n <= 10) ;
            Console.ReadLine();
        }
    }
}
```



Exemplul 33: Să se afișeze numerele cu proprietatea de a fi palindroame, până la o valoare citită de la tastatură. De asemenea, să se afișeze și numărul lor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exemplul_33
{
    class Program
    {
        static void Main(string[] args)
        {
            int x, n, k = 0;
            do
            {
                Console.Write("Dati un numar natural : ");
                n = Convert.ToInt32(Console.ReadLine());
                if (n <= 0)
                    Console.WriteLine("Eroare la citire!");
            } while (n <= 0);
            Console.Write("Numerele palindroame mai mici strict decat {0} sunt :\n", n);
            x = 1;
        }
    }
}
```

```

do
{
    if (palindrom(x) == 1)
    {
        Console.Write(" {0,3} ", x);
        k++;
    }
    x++;
} while (x < n);
Console.WriteLine();
if (k == 0) Console.WriteLine("Nu exista numere!");
else Console.WriteLine("Sunt {0} numere palindroame!", k);
}
static uint palindrom(int x)
{
    int y = 0, z = x;
    do
    {
        y = y * 10 + z % 10;
        z /= 10;
    } while (z != 0);
    if (y == x) return 1;
    else return 0;
}
}
}

```

```

C:\Windows\system32\cmd.exe
Dati un numar natural : 1234
Numerele palindroame mai mici strict decat 1234 sunt :
 1  2  3  4  5  6  7  8  9  11  22  33  44  55  66  77
88  99 101 111 121 131 141 151 161 171 181 191 202 212 222 232
242 252 262 272 282 292 303 313 323 333 343 353 363 373 383 393
404 414 424 434 444 454 464 474 484 494 505 515 525 535 545 555
565 575 585 595 606 616 626 636 646 656 666 676 686 696 707 717
727 737 747 757 767 777 787 797 808 818 828 838 848 858 868 878
888 898 909 919 929 939 949 959 969 979 989 999 1001 1111 1221
Sunt 111 numere palindroame!
Press any key to continue . . .

```

Instrucțiunea for

Instrucțiunea **for** are sintaxa:

```

for(initializareCiclu; conditieFinal; reinitializareCiclu)
    Instructiune

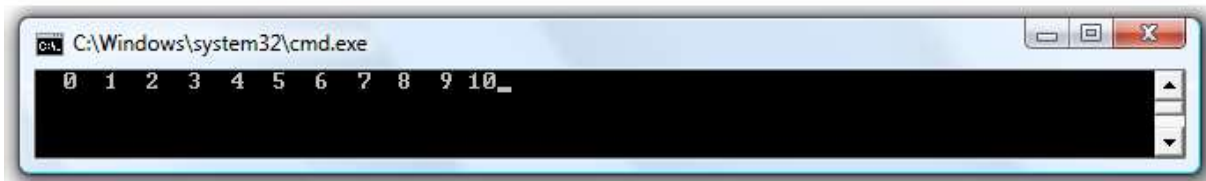
```

Exemplul 34: Ne propunem, să afișăm numerele pozitive ≤ 10

```

using System;
namespace Exemplul_34
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int n = 0; n <= 10; n++)
            {
                Console.Write("{0,3}", n);
            }
            Console.ReadLine();
        }
    }
}

```

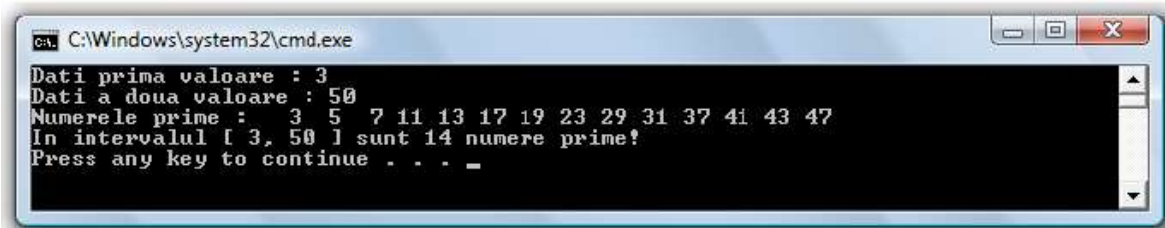


Exemplul 35: Să se determine numerele prime, precum și numărul lor, cuprinse între două valori întregi citite de la tastatură.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exemplul_35
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b, x, k = 0; // k va determina cate numere prime sunt in
            interval
            do
            {
                Console.Write("Dati prima valoare : ");
                a = Convert.ToInt32(Console.ReadLine());
            } while (a <= 0);
            do
            {
                Console.Write("Dati a doua valoare : ");
                b = Convert.ToInt32(Console.ReadLine());
            } while (b <= a);
```

```
                Console.Write("Numerele prime : ");
                for (x = a; x <= b; x++)
                {
                    if (prim(x) == 1)
                    {
                        Console.Write("{0, 3}", x);
                        k++;
                    }
                }
                Console.WriteLine();
                if (k == 0)
                    Console.WriteLine("In intervalul [ {0}, {1} ] nu sunt numere
prime!", a, b);
                else
                    Console.WriteLine("In intervalul [ {0}, {1} ] sunt {2} numere
prime!", a, b, k);
            }
            static int prim(int x)
            {
                if (x == 1) return 0;
                if (x % 2 == 0 && x != 2) return 0;
                for (int d = 3; d * d <= x; d += 2)
                    if (x % d == 0) return 0;
                return 1;
            }
        }
    }
}
```



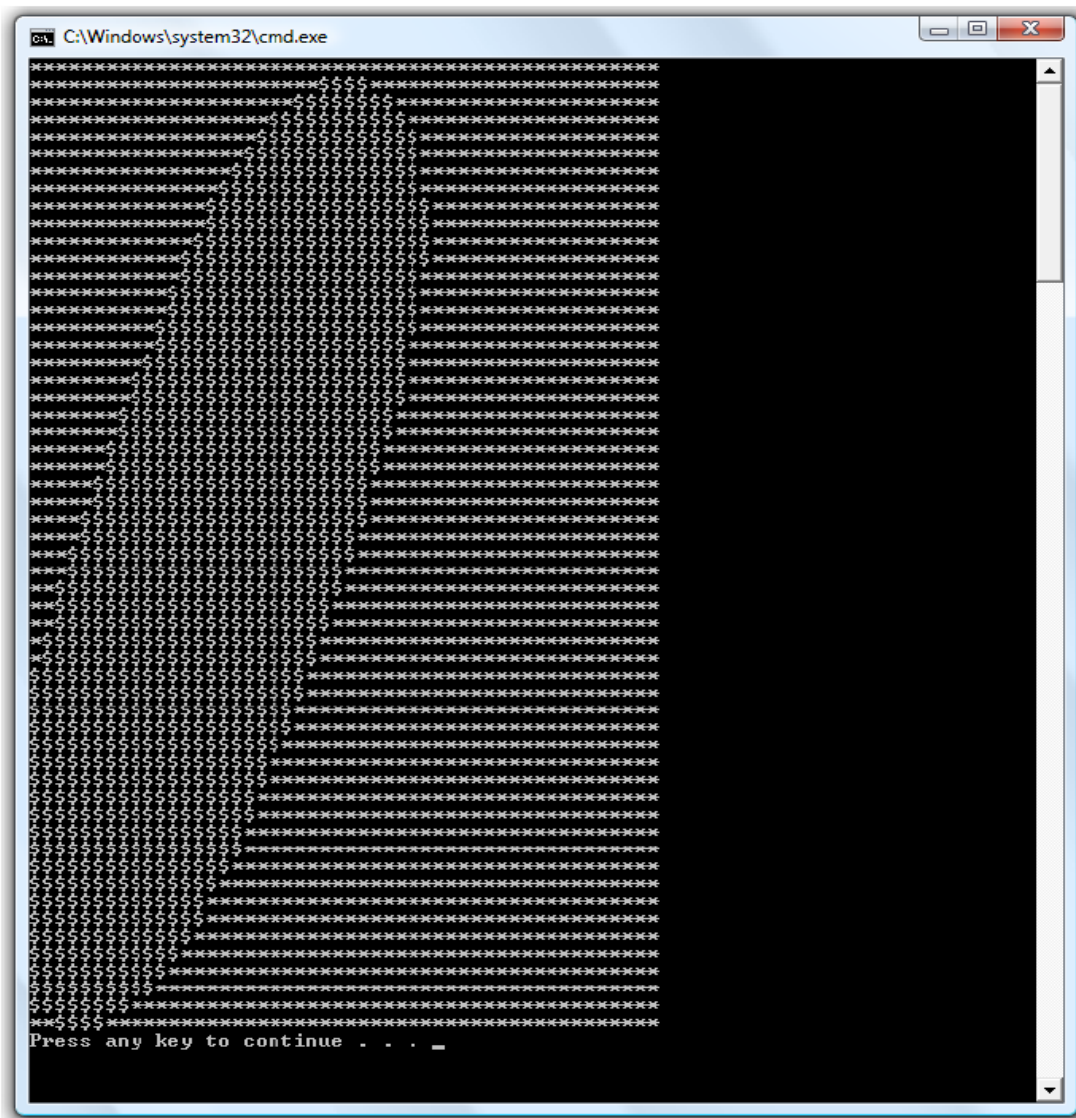
```
C:\Windows\system32\cmd.exe
Dati prima valoare : 3
Dati a doua valoare : 50
Numerele prime : 3 5 7 11 13 17 19 23 29 31 37 41 43 47
In intervalul [ 3, 50 ] sunt 14 numere prime!
Press any key to continue . . . _
```

Exemplul 36: Un exemplu de for pe numere reale.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exemplul_36
{
    class Program
    {
        static void Main(string[] args)
        {
            double rc, ic;
            double x, y, z;
            int n;
            for (ic = 1.4; ic >= -1.4; ic -= 0.05)
            {
                for (rc = -0.7; rc <= 1.80; rc += 0.05)
                {
```

```
                    n = 0;
                    x = ic * ic + rc * rc;
                    y = 2 * ic - 4 * rc;
                    z = x * x + y * y;
                    while (n <= 40 && z < 5)
                    {
                        x = ic * ic + rc * rc - rc;
                        y = 2 * ic - 4 * rc;
                        z = x * x - y * y;
                        n++;
                    }
                    switch (n % 4)
                    {
                        case 0: Console.Write("*"); break;
                        case 1: Console.Write("$"); break;
                        case 2: Console.Write("o"); break;
                        case 3: Console.Write("@"); break;
                    }
                    Console.WriteLine();
                }
            }
        }
    }
}
```



Instrucțiunea foreach

O instrucțiune nouă, pe care o aduce limbajul C#, este **foreach**. Această instrucțiune enumeră elementele dintr-o colecție, executând o instrucțiune pentru fiecare element. Elementul care se extrage este de tip read-only, neputând fi transmis ca parametru și nici aplicat un operator care să-i schimbe valoarea.

Pentru a vedea cum acționează, o vom compara cu instrucțiunea cunoscută **for**.

Considerăm un vector nume format din șiruri de caractere:

```
string[] nume = {"Ana", "Ionel", "Maria"}
```

Afișarea șirului folosind for:

```
for(int i=0; i<nume.Length; i++)  
    Console.Write("{0} ", nume[i]);
```

Același rezultat îl obținem folosind instrucțiunea **foreach**:

```
foreach (string copil in nume)  
    Console.Write("{0} ", copil);
```

Mai dăm încă un exemplu de folosire a lui **foreach**:

```
string s="Curs"+" de"+" informatica";
foreach(char c in s)
    Console.Write(c);
```

Exemplul 37: Să se împartă un șir de caractere în cuvinte. Se va afișa numărul de cuvinte și fiecare cuvânt în parte

```
using System;
namespace Exemplul_37
{
    class Program
    {
        static void Main(string[] args)
        {
            string sir = "Acesta este un sir";
            char[] delimiter = { ' ', ',', '.', ':' };
            Console.WriteLine("Sirul care va fi impartit in cuvinte\n'{0}'", sir);
            string[] cuvint = sir.Split(delimiter);
            Console.WriteLine("Sunt {0} cuvinte in text:", cuvint.Length);
            foreach (string s in cuvint)
            {
                Console.WriteLine(s);
            }
        }
    }
}
```

Instrucțiunea goto

Instrucțiunea **goto** poate fi folosită, în C#, pentru efectuarea unor salturi, în instrucțiunea **switch**

Exemplul 38:

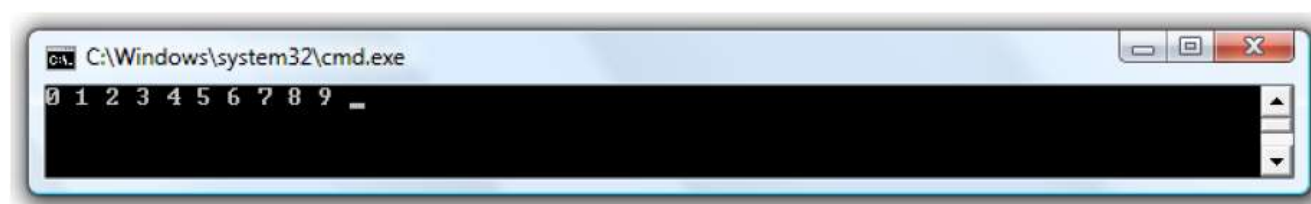
```
switch (a)
{
    case 13:
        x = 0;
        y = 0;
        goto case 20;
    case 15:
        x = 3;
        y = 1;
        goto default;
    case 20:
        x = 5;
        y = 8;
        break;
    default:
        x = 1;
        y = 0;
        break;
}
```


Instrucțiunea continue

Instrucțiunea **continue** permite reluarea iterației celei mai apropiate instrucțiuni **switch**, **while**, **do – while**, **for** sau **foreach**.

Exemplul 39:

```
using System;
namespace Exemlul_39
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 0;
            while (true)
            {
                Console.Write("{0} ", i);
                i++;
                if (i < 10)
                    continue;
                else
                    break;
            }
            Console.ReadLine();
        }
    }
}
```



Instrucțiunile try-catch-finally și throw

Prin excepție se înțelege un obiect care încapsulează informații despre situații anormale. Ea se folosește pentru a semnaliza contextul în care apare o situație specială.

Exemple: erori la deschiderea unor fișiere a căror nume este greșit, împărțire la 0 etc. Aceste erori se pot manipula astfel încât programul să nu se prăbușească.

Când o metodă întâlnește o situație dintre cele menționate mai sus, se va „arunca” o excepție care trebuie sesizată și tratată. Limbajul C# poate arunca ca excepții obiecte de tip **System.Exception** sau derivate ale acestuia. Aruncarea excepțiilor se face cu instrucțiunea **throw**

```
throw new System.Exception();
```

Prinderea și tratarea excepțiilor se face folosind un bloc **catch**. Pot exista mai multe blocuri **catch**, fiecare dintre ele prinde și tratează o excepție.

Pentru a garanta că un anumit cod se va executa indiferent dacă totul decurge normal sau apare o excepție, acest cod se va pune în blocul **finally** care se va executa în orice situație.

Exemplul 40:

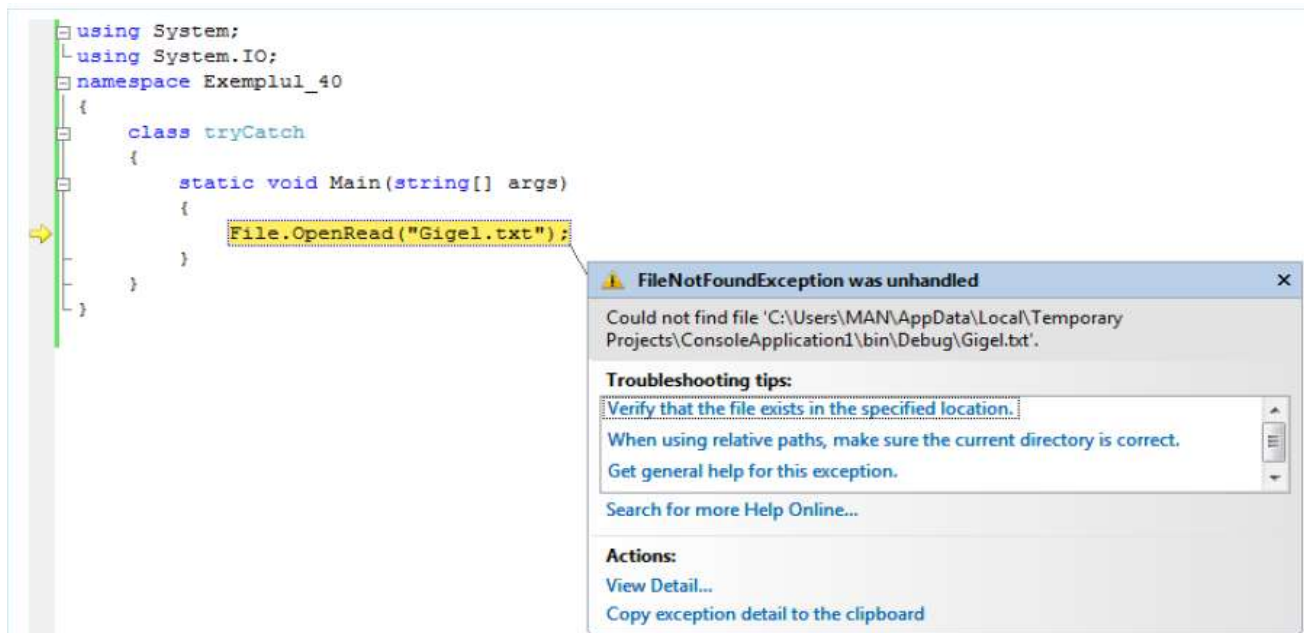
Presupunem că dorim să citim fișierul „Gigel.txt”

```

using System;
using System.IO;
namespace Exemplul_40
{
    class tryCatch
    {
        static void Main(string[] args)
        {
            File.OpenRead("Gigel.txt");
        }
    }
}

```

Încercând să compilăm obținem:



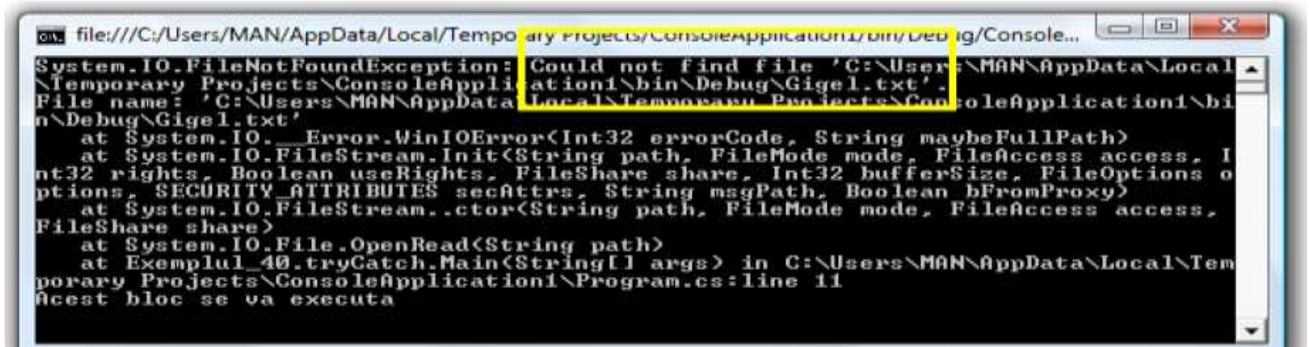
Pentru a remedia această eroare, vom prinde excepția, punând într-un bloc **try** linia care a furnizat-o.

Putem vizualiza mesajul produs de excepția întâlnită:

```

using System;
using System.IO;
namespace Exemplul_40
{
    class tryCatch
    {
        static void Main(string[] args)
        {
            try
            {
                File.OpenRead("Gigel.txt");
            }
            catch (FileNotFoundException a)
            {
                Console.WriteLine(a);
            }
            finally
            {
                Console.WriteLine("Acest bloc se va executa");
                Console.ReadLine();
            }
        }
    }
}

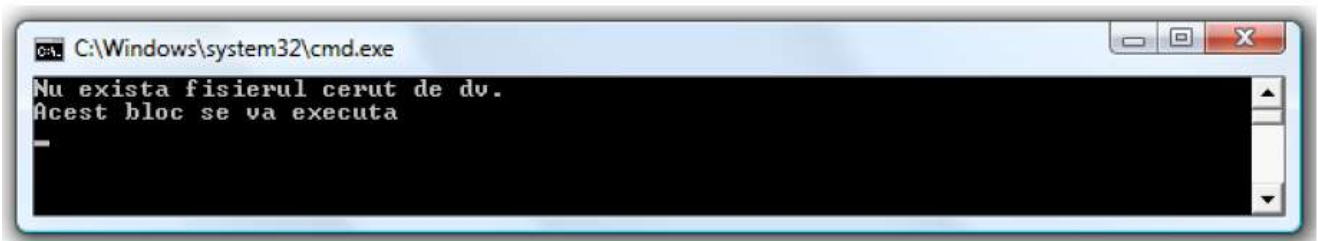
```

```
file:///C:/Users/MAN/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug/Console...
System.IO.FileNotFoundException: Could not find file 'C:\Users\MAN\AppData\Local\Temporary Projects\ConsoleApplication1\bin\Debug\Gigel.txt'.
File name: 'C:\Users\MAN\AppData\Local\Temporary Projects\ConsoleApplication1\bin\Debug\Gigel.txt'
   at System.IO._Error.WinIOError(Int32 errorCode, String maybeFullPath)
   at System.IO.FileStream.Init(String path, FileMode mode, FileAccess access, Int32 rights, Boolean useRights, FileShare share, Int32 bufferSize, FileOptions options, SECURITY_ATTRIBUTES secAttrs, String msgPath, Boolean bFromProxy)
   at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare share)
   at System.IO.File.OpenRead(String path)
   at Exemplul_40.tryCatch.Main(String[] args) in C:\Users\MAN\AppData\Local\Temporary Projects\ConsoleApplication1\Program.cs:line 11
Acest bloc se va executa
```

Bineînțeles că în blocul catch putem să scriem ce cod dorim, de exemplu:

```
using System;
using System.IO;
namespace Exemplul_40
{
    class tryCatch
    {
        static void Main(string[] args)
        {
            try
            {
                File.OpenRead("Gigel.txt");
            }
            catch (FileNotFoundException a)
            {
                Console.WriteLine("Nu exista fisierul cerut de dv.");
            }
            finally
            {
                Console.WriteLine("Acest bloc se va executa");
                Console.ReadLine();
            }
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
Nu exista fisierul cerut de dv.
Acest bloc se va executa
_
```

Alteori putem simula prin program o stare de eroare, „aruncând” o excepție (instrucțiunea **throw**) sau putem profita de mecanismul de tratare a erorilor pentru a implementa un mecanism de validare a datelor prin generarea unei excepții proprii pe care, de asemenea, o „aruncăm” în momentul neîndeplinirii unor condiții puse asupra datelor.

Clasa **System.Exception** și derivate ale acesteia servesc la tratarea adecvată și diversificată a excepțiilor.