

## Laborator 13

### Tratarea excepțiilor în C#

**Definiție:** O **excepție** este un obiect care încapsulează informații despre o situație anormală.

Excepția se folosește pentru a semnaliza contextul în care apare acea situație deosebită

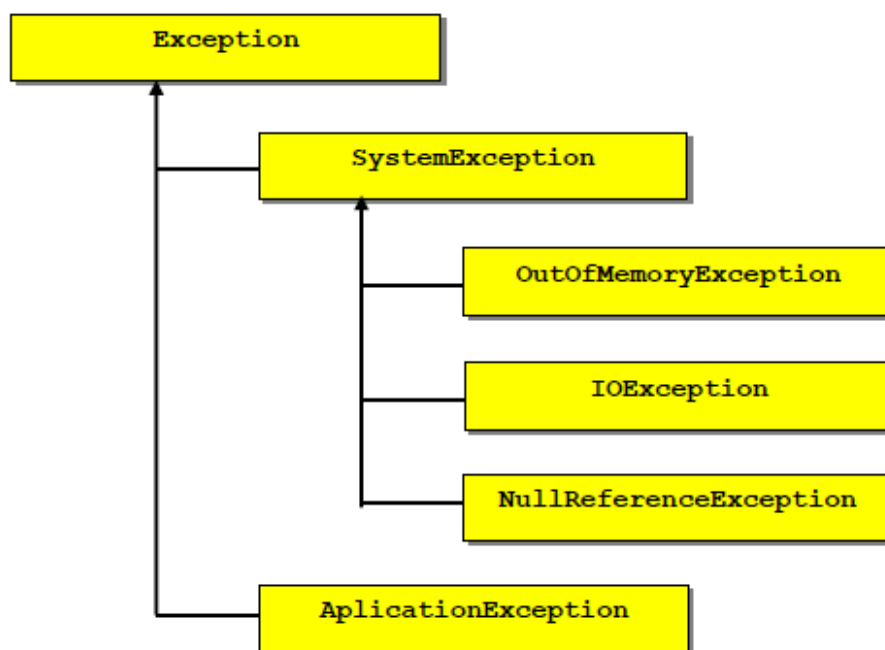
**Observație:** Nu trebuie confundat termenul de excepție cu cel de eroare sau „bug”.

Excepțiile nu sunt concepute pentru prevenirea bug-urilor. Chiar dacă programatorul elimină toate bug-urile din programul său pot apărea erori pe care el nu le poate preveni:

- încercare de deschidere a unui fișier inexistent
- împărțiri la zero

În cazul în care o metodă întâlnește o astfel de excepție, atunci respectiva excepție va trebui **„prinsă”** în vederea tratării (rezolvării) ei.

În C# se pot **arunca** ca excepții obiecte de tip **System.Exception** sau derivate ale lui. Pe lângă ierarhia de excepții pe care limbajul C# o are inclusă, programatorul își poate crea propriile sale tipuri excepție.



Ierarhia excepțiilor

Dintre metodele și proprietățile clasei **Exception** amintim:

Metodele și proprietățile clasei Exception	Explicații
<pre>public Exception( ) public Exception (string) public Exception (string, Exception)</pre>	sunt constructori
<pre>public virtual string HelpLink {get; set;}</pre>	obține sau setează o legătură către fișierul Help asociat excepției, sau către o adresă Web
<pre>public Exception InnerException {get;}</pre>	returnează excepția care este încorporată în excepția curentă
<pre>public virtual string Message {get;}</pre>	obține un mesaj care descrie excepția curentă
<pre>public virtual string Source {get; set;}</pre>	obține sau setează numele aplicației sau al obiectului care a cauzat eroarea
<pre>public virtual string StackTrace {get;}</pre>	obține o reprezentare de tip string a apelurilor de metode care au dus la apariția excepției
<pre>public MethodBase TargetSite {get;}</pre>	obține metoda care a aruncat excepția curentă

C# definește câteva excepții standard derivate din System.Exception. Acestea sunt generate când se produc erori la execuția programului. Dintre acestea amintim:

Excepția	Explicații
ArrayTypeMismatchException	Incompatibilitate între tipul valorii memorate și tipul tabloului
DivideByZeroException	Încercare de împărțire la zero
IndexOutOfRangeException	Indexul tabloului depășește marginile definite
InvalidCastException	Operatorul cast incorect la execuție
OutOfMemoryException	Datorită memoriei insuficiente apelul lui new eșuează
OverflowException	Depășire aritmetică
StackOverflowException	Depășirea capacității (definite) stivei

**Observație:** Este posibilă definirea de către programator a propriilor clase de excepții. Acestea vor fi derivate din **ApplicationException**.

## Aruncarea și prinderea excepțiilor

### Blocurile try și catch

POO oferă o soluție pentru gestionarea erorilor: folosirea blocurilor **try** și **catch**. În scrierea codului, programatorul va separa acele instrucțiuni care sunt sigure (adică nu pot fi generatoare de excepții), de cele care sunt susceptibile să conducă la erori. Partea de program care poate genera excepții o vom plasa într-un bloc **try**, iar partea corespunzătoare tratării excepției, într-un bloc **catch**.

În cazul în care blocul try generează o excepție, Runtime întrerupe execuția și caută un bloc **catch** apropiat care, în funcție de tipul său să poată trata respectiva eroare. În cazul în care este găsit respectivul bloc **catch** programul continuă cu instrucțiunile din corpul **catch**. În cazul în care nu se găsește nici un **catch** corespunzător, execuția programului este întreruptă.

Având în vedere că într-un corp **try** pot să apară excepții diferite, în program pot exista mai multe blocuri corespunzătoare **catch**.

Exemplul 91:

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

namespace Exceptiile
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.Write("Introduceți un număr ");
                int i = int.Parse(Console.ReadLine());
                Console.Write("Introduceți încă un număr ");
                int j = int.Parse(Console.ReadLine());
                int x = i / j;
            }
            catch (OverflowException e)
            {
                Console.WriteLine("Numarul nu este intreg"); // (1)
                //Console.WriteLine(e); // (2)
            }

            /*
            catch (DivideByZeroException e)
            {
                //Console.WriteLine(e); // (3)
                Console.WriteLine("Exceptia DivideByZero"); // (4)
            }
            */
            Console.WriteLine("Programul ruleaza in continuare");// (5)
        }
    }
}
```

Să analizăm puțin programul de mai sus:

- Dacă liniile (2) și (3) nu sunt comentate, în urma execuției programului, respectivele linii afișează informații despre excepțiile apărute.
- Liniile (1) și (4) au fost puse pentru a personaliza informațiile referitoare la excepțiile apărute.
- Linia (5) a fost pusă în program pentru a demonstra rularea fără probleme, în cazul în care blocurile **catch** există. Încercați să comentați unul dintre blocurile **catch**, introduceți date care să producă excepția pe care blocul comentat ar trata-o și veți observa întreruperea, cu mesaj de eroare a rulării programului.

**Observație:** Pentru a intercepta orice excepții, indiferent de tipul lor se va folosi **catch** fără parametru. Prin aceasta se va crea o rutină care va intercepta și trata toate excepțiile.

## Instrucțiunea **throw**

Programatorul poate să-și compună modalități proprii de aruncare a erorilor folosind instrucțiunea **throw**:

```
throw new NumeExceptie(exceptie);
```

unde:

**NumeExceptie** trebuie să fie numele unei clase apropiate de excepția avută în vedere

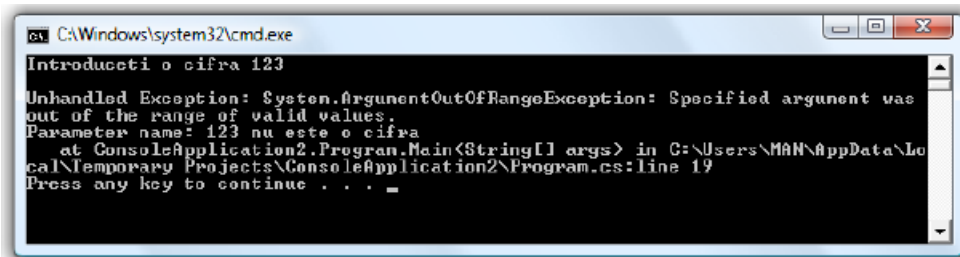
**exceptie** – este un mesaj care apare în cazul în care apare excepția, iar aceasta nu este prinsă cu **catch**

Exemplul 92:

```
class Program
{
    static void Main(string[] args)
    {
        try                                //(1)
        {                                  //(2)
            Console.Write("Introduceti o cifra ");
            int i = int.Parse(Console.ReadLine());
            if (i < 0 || i > 9)
            {
                string exceptie = i + " nu este o cifra"; //(0)
                throw new ArgumentOutOfRangeException(exceptie);
            }
        }                                //(3)
        catch (ArgumentOutOfRangeException)           //(4)
        {                                              //(5)
            Console.WriteLine("Nu este cifra");        //(6)
        }                                              //(7)
        Console.WriteLine("Programul ruleaza in continuare");
    }
}
```

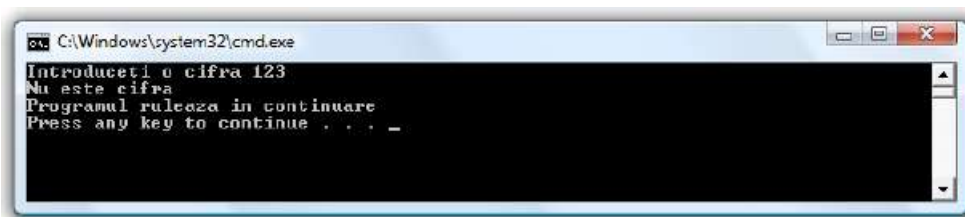
Să analizăm programul de mai sus:

- Dacă comentăm liniile (1), (2), (3), (4), (5), (6), (7) și la rularea programului introducem un număr în loc de o cifră, programul se oprește din execuție, iar ca mesaj apare \_irul definit de utilizator în linia (0)



- Dacă vom comenta doar liniile aferente blocului catch (4), (5), (6), (7), apare un mesaj de eroare privind faptul că se așteaptă un bloc catch sau finally

- Dacă nici una dintre liniile programului nu este comentată, la rulare, chiar dacă introduce un număr în loc de o cifră vom obține:



## Blocul finally

Limbajul C# permite ca la ieșirea dintr-un bloc **try** să fie executate obligatoriu, în cazul în care programatorul dorește acest lucru, anumite instrucțiuni. Pentru acest lucru, respectivele instrucțiuni vor fi plasate într-un bloc **finally**.

Blocul **finally** este util fie pentru a evita scrierea unor instrucțiuni de mai multe ori, fie pentru a elibera resursele după părăsirea excepției.