### Modulul 4. Excepțiile și prevenirea lor în C#

#### Ce este o eroare și o excepție? Importanța prevenirii lor.

Apariția erorilor în programare este considerată un eveniment obișnuit, ele arată programatorilor că programul nu a fost scris/executat corect și oferă posibilitatea de a identifica proveniența erorilor înainte de a crea o versiune finală a aplicației.

După categoria lor erorilor pot fi clasificate în 3 tipuri de bază:

1. **De sintaxă** - reprezintă erorile cauzate de abaterea de la standardele limbajului de programare.

*Exemple:*

Console,WriteLine(“a”); //, în loc de .

Console.WriteLine(“b”) //omis ; la sfârșitul rândului

Console.WriteLine(“b”): //: în loc de ;

string private cuvint =”alo”; //nivelul de accesibilitate private declarat după tipul variabilei

**Erorile de sintaxă vor fi indicate direct în editor și nu vă vor permite să rulați programul până când nu vor fi înlăturate.**

1. **De logică** - reprezintă erorile cauzate de logica incorectă elaborată de programator, ceea ce semnifică că programul va funcționa corect, nu vor exista mesaje de eroare în editor, *dar rezultatul obținut va fi diferit de cel așteptat.*

*Exemplu:*

class EroriLogice

{

static void Main()

{

int inceput = 10;

int sfirsit = 1;

int raspuns = 0;

for (int i = inceput; i < sfirsit; i++)

{

raspuns += i;

}

Console.WriteLine(raspuns);

}

}

**Erorile logice pot fi cele mai complicat de identificat din motiv că ele nu generează mesaje de eroare în editor și nu stopează execuția programului. Ca să identificați o eroare de logică trebuie în mod logic să traversați execuția programului și să identificați originea.**

***Recomandare: Folosiți mesajele la consolă pentru a vedea rezultatul fiecărei etape de execuția a programului. De exemplu dacă aveți un program care trebuie să execute X pași, verificați rezultatul la fiecare pas, astfel veți identifica problema mai rapid.***

1. **Erori în timpul execuției sau EXCEPȚII** - sunt erorile care intervin în timp ce programul funcționează, dar nu sunt legate de sintaxă, ceea ce le poate face cele mai periculoase pentru programul vostru pentru că pot genera erori care vor opri execuția programului sau îl vor face să meargă incorect.

*Exemplu:*

class Exceptii

{

static void Main()

{

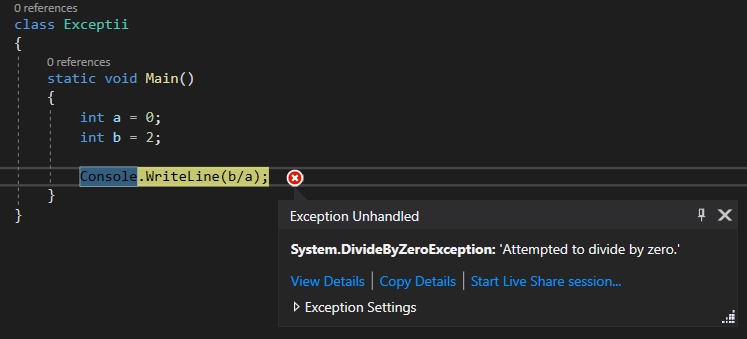
int a = 0;

int b = 2;

Console.WriteLine(b/a);

}

}



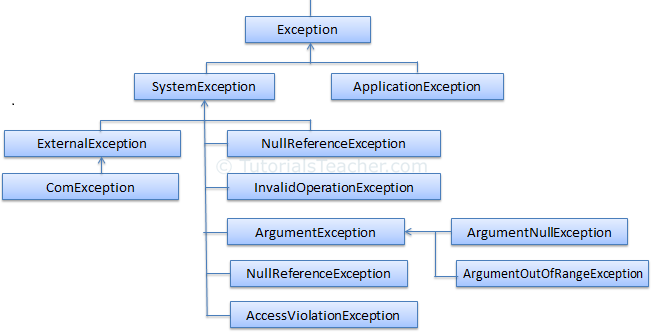
Puteți să încercați să preveniți excepțiile testând aplicația în foarte multe scenarii. În caz că aveți o aplicație masivă testați fiecare funcționalitate în parte înainte să le combinați.

**Dar cea mai bună metodă de a ne asigura că nu vor exista excepții este de a crea blocuri pentru procesarea excepțiilor, acesta este scopul lecției date.**

#### Ierarhia excepțiilor

În C# toate tipurile de excepții care pot fi generate derivă din clasa de bază **System.Exception**.

**Ierarhia exceptiilor.** Excepțiile în C# sunt grupate într-o ierarhie, fiecare tip de excepție este utilizat în anumite cazuri, ceea ce ne oferă posibilitatea să procesăm fiecare tip de excepții în mod separat.



<https://www.tutorialsteacher.com/csharp/csharp-exception>

După cum e demonstrat în graficul de mai sus excepțiile se împart în două ramuri generale:

**ApplicationException** sunt excepțiile generate în timpul funcționării aplicației de către utilizator(de exemplu: când avem un câmp input de tip string iar utilizatorul introduce o valoare numerică). Excepțiile de acest tip nu sunt fatale, aplicația va continua să funționeze în mod corect.

**SystemException** sunt excepțiile vor provoca stoparea totală a programului, acestea sunt cauzate de introducerea de către programator a unui tip de date incorect, de utilizarea unei operații imposibile matematic, de lipsa unei referințe necesare pentru funcționare și altele. Ceea ce au în comun e că trebuie prevenite cu orice scop dacă doriți ca programul vostru să funcționeze.

Excepțiile system la rândul lor se împart în:

1. **NullReferenceException** - excepția care apare când încercați să faceți referință la un obiect care nu a fost inițializat.

*Exemplu:*

class NullRef

{

static void Main()

{

int[] array = null;

Console.WriteLine(array.Length);

}

}

1. **Invalid Operation Exception** - excepția care apare când încercați să faceți o operație care este imposibilă în contextul dat.

*Exemplu:*

class InvalidOperation

{

static void Main()

{

int[] numbers = {};

int first = numbers.First();

Console.WriteLine(first);

}

}

1. **ArgumentException** - apare atunci când unul din argumentele transmise unei metode nu este valid.

Această excepție la rândul ei se va clasifica în 2 tipuri:

1. **ArgumentNullException** - argumentul transmis unei metode este nul.

*Exemplu:*

class ArgumentNull

{

static void Main()

{

string exemplu = "Cuvint";

Console.WriteLine(exemplu.IndexOf(null));

}

}

1. **ArgumentOutOfRangeException** - apare atunci când argumentul transmis metodei iese din limitele admisibile ale metodei.

*Exemplu:*

class ArgumentOutOfRange

{

static void Main()

{

int[] array = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int inceput = -1;

int sfirsit = 11;

for (int i = inceput; i < sfirsit; i++)

{

Console.WriteLine(array[i]);

}

}

}

1. **AccessViolationException** - apare când se atentează de a citi sau modifica date care nu sunt accesibile, sau de a accesa o parte din memorie care nu a fost încă alocată.

*Exemplu:*

unsafe class Access

{

static void Main()

{

IntPtr ptr = new IntPtr(123);

Marshal.StructureToPtr(123, ptr, true);

}

}

1. **ExternalException și ComException** - sunt excepțiile care apar când programul încearcă să acceseze DLL-uri externe. Este întâlnită doar în programele care au nevoie să utilizeze fișiere, imagini sau librării externe pentru a realiza o anumită funcție

*Exemplu:*

class Access

{

static void Main()

{

Capture capture = new Capture(); //create a camera capture

Bitmap image = capture.QueryFrame().Bitmap; //take a picture

image.Save((path + "data" + i.ToString() + ".jpg").Replace(" ", "\_"));

}

}

#### Cum prevenim excepțiile

**Pentru a preveni toate excepțiile menționate anterior,** în C# putem folosi 4 cuvinte cheie:

1. ***Try*** *-* în acest **bloc** încercăm dacă poate fi executată operația de care avem nevoie, dacă este detectată o excepție atunci blocul try transmite controlul blocului **catch.**
2. ***Catch*** *-* blocul catch **“prinde” excepțiile** transmise de try și o tratează în modul în care determină programatorul.
3. ***Finally*** *-* instrucțiunile din blocul finally se execută totdeauna indiferent de dacă a fost găsită o excepție sau nu.

Observații importante:

1. Blocul **try** totdeauna se **autocompletează cu catch sau finally**. Nu există scenariu în care un block try poate fi folosit fără catch sau finally și viceversa.
2. Puteți utiliza un număr nelimitat de blocuri try-catch-finally atât timp cât este respectată combinația dintre ele.

*Exemplu:*

class TryCatchFinally

{

static void Main()

{

try

{

int x = 5;

//Determina daca e posibila impartirea

int y = 5 / x;

//Daca da, afiseaza rezultatul

Console.WriteLine(y);

}

catch

{

//Executa logica de tratare a exceptiei

Console.WriteLine("Introduceti un numar diferit de 0!");

}

finally

{

//Bloc optional, se executa mereu

Console.WriteLine("Program executat");

}

}

}

1. ***Throw*** *-* cuvântul cheie throw permite programatorului să genereze manual o excepție în timpul execuției programului.

Aceasta permite programatorului să evite unele scenarii nedorite de execuție, și să execute comenzile din catch când unul din aceste scenarii se întâmplă.

*Exemplu:*

class Throw

{

public string ClassName = "Throw";

static void Main()

{

Throw std = null;

try

{

PrintStudentName(std);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.ReadKey();

}

private static void PrintStudentName(Throw std)

{

if (std == null)

throw new NullReferenceException("Obiectul transmis este null!");

Console.WriteLine(std.ClassName);

}

}

#### Metode avansate de prelucrare a excepțiilor

1. **Clasificarea și prelucrarea mai multor excepții.**

În unele cazuri o singură metodă poate genera mai multe tipuri de excepții, și fiecare dintre aceste tipuri trebuie prelucrată diferit pentru că are o cauză diferită.

În acest caz putem folosi un singur bloc try în care încercăm să executăm metoda, în combinație cu mai multe blocuri catch, fiecare din ele destinate pentru un anumit tip de excepții.

*Exemplu:*

class ClasifExceptiilor

{

static void Main()

{

int[] array = {1};

PrelucreazaTablou(array);

}

//Afiseaza elementul cu locul 2 dintr-un tablou

static void PrelucreazaTablou(int[] \_tablou)

{

try

{

Console.WriteLine(\_tablou[1]);

}

catch (NullReferenceException e) //Parametrul transmis este null

{

PrinteazaMesajEroare(e);

}

catch (IndexOutOfRangeException e) //Dimensiunea tabloului primit este prea mica

{

PrinteazaMesajEroare(e);

}

catch (FormatException e) //Parametrul transmis nu este de tip int

{

PrinteazaMesajEroare(e);

}

catch (Exception e) //Alta exceptie neprevazuta

{

PrinteazaMesajEroare(e);

}

}

static void PrinteazaMesajEroare(Exception \_e)

{

Console.WriteLine(\_e.Message);

}

}

1. **Blocuri try integrate**

Există unele cazuri în care este util să folosim un bloc try integrat în altul. De exemplu în cazul în care căutați o excepție de un anumit tip, dar programul generează o altă excepție la care nu v-ați așteptat, anume în această situație blocul try extern se folosește în rolul de “plasă de siguranță adițională”.

*Exemplu:*

class TryIntegrat

{

static void Main()

{

int[] array = null;

try

{

try

{

Console.WriteLine(array[1]);

}

catch (IndexOutOfRangeException e) //Dimensiunea tabloului primit este prea mica

{

Console.WriteLine(e.Message);

}

}

catch (Exception e)

{

Console.WriteLine("Exceptie neprevazuta:" + e.Message);

}

}

}

1. **Generarea excepțiilor repetată**

În anumite cazuri putem avea nevoie de a accesa informația despre o excepție dintr-o metodă anumită în metoda Main. În acest caz putem folosi **generarea repetată** a excepțiilor.

Generarea repetată reprezintă procesul de “aruncare” a unei excepții dintr-o metodă secundară în cea Main prin utilizarea cuvântului cheie throw. În acest mod noi putem accesa toată informația despre o excepție în ramura de bază a programului

*Exemplu:*

class ExceptieRepetata

{

static void Main()

{

try

{

Impartire(5, 0);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

static void Impartire(int x, int y)

{

try

{

int result = x / y;

}

catch

{

throw;

}

}

}

#### Utilizarea construcțiilor checked și unchecked

În mod normal în C# tipurile primitive(int, float, decimal etc.) au o anumită limită prestabilită. Când această limită este depășită în timpul execuției programului, programatorul poate decide dacă acestă depășire va genera excepție sau nu cu ajutorul blocurilor checked/unchecked.

**Blocul checked -** presupune că atunci când depășirea valorii maxime sau minime se întâmplă o excepție va fi generată.

**Blocul unchecked -** presupune lucrul invers, când valoarea maximă sau minimă este depășită programul nu afișează excepție.

**Observație:** Dacă nu indicăm aceste două blocuri C# funcționează în mod unchecked.

*Exemplu:*

class Check

{

static void Main()

{

checked

{

int val = int.MaxValue;

Console.WriteLine(val + 2);

}

}

}

#### Filtre pentru excepții

Filtrele pentru excepții sunt o adiție relativ nouă în C#. Introduse în C# 6.0 în 2015, acestea permit verificarea erorii primite cu un anumit filtru preselectat de programator, dacă verificarea returnează true va fi executată logica din blocul catch, dacă nu - nu.

*Exemplu:*

class ExcFilter

{

static void Main()

{

int x = 1;

int y = 0;

try

{

int result = x / y;

}

catch (DivideByZeroException) when (y == 0 && x == 0)

{

Console.WriteLine("Ambii parametri sunt egali cu 0!");

}

catch (DivideByZeroException) when (y == 0)

{

Console.WriteLine("y este egal cu 0!");

}

catch (DivideByZeroException ex)

{

Console.WriteLine(ex.Message);

}

}

}

**Programe pentru elaborare:**

**Program 1.**

1. Scrieți un program în care veți avea variabila salariu.
2. Creați o metodă MiscorareSalariu care va primi un parametru numeric și va scădea din variabila salariu parametrul primit.
3. Dacă variabila salariu va fi mai mică ca 0, folosiți throw pentru a genera o eroare cu mesajul “Salariul nu poate fi negativ!”.

**Rezolvare:**

class Angajat

{

private static int salariu = 10000;

static void Main()

{

MiscorareSalariu(12000);

}

public static void MiscorareSalariu(int \_valoare)

{

int rezultat = salariu - \_valoare;

try

{

if (rezultat <= 0)

throw new Exception("Salariu nu poate fi negativ");

else

Console.WriteLine("Salariu Nou:{0}", rezultat);

}

catch

{

rezultat = salariu;

Console.WriteLine("Schimbarea nu a fost acceptata. Salariu a ramas {0}", rezultat);

}

}

}

**Program 2.**

1. Scrieți un program în care veți declara un array de string-uri.
2. Creați o metodă procesare string care va avea parametru un tablou unidimensional de stringuri.
3. Printați la consolă litera 5 din fiecare cuvânt.
4. Folosiți try catch pentru evitarea obținerii unui parametru null, dacă fiecare cuvânt are 5 litere, dacă tabloul are cel puțin un element.
5. Printați mesaje erorilor respective.

**Opțional**

1. În caz că tabloul este null - înlocuiți-l cu unul corespunzător.
2. În caz că tabloul are mai puțin de un element, înlocuiți-l cu unul corespunzător cu minim un element string.
3. În caz că un cuvânt are mai puțin de 5 litere substituiți-l cu unul de lungime potrivită și continuați.
4. Continuați execuția programului după ce ați făcut înlocuirile corespunzătoare și executați condiția inițială propusă (afișați litera 5 din fiecare cuvânt din tablou).

**Program 3.**

1. Scrieți un program în care veți declara câteva variabile în metoda Main(). (Alegeți singuri condiția și tipul variabilei)
2. Creați o metodă care va primi ca parametri variabilele care le-ați declarat în Main().
3. Creați cel puțin două excepții în metoda care ați creat-o(nu în Main).
4. Folosiți throw pentru a arunca excepția în Main.
5. Procesați excepți cu ajutorul try-catch în metoda Main() și afișați mesajul excepției corespunzător.

**Opțional**

1. Ajustați în Main() variabilele care crează excepția și continuați execuția programului pentru a finisa cu succes scopul inițial.

**Program 4.**

1. Scrieți un program în care veți crea mai multe excepții.
2. Folosiți try-catch(Exception e) și printați tipul excepției cu ajutorul filtrelor.

**Program 5.**

1. Scrieți un program în care veți crea mai multe excepții.
2. Folosiți try-catch(Exception e) și printați tipul excepției cu ajutorul filtrelor.

**Programe opționale**

**Program 6.**

1. Scrieți un program în care veți afișa toate tipurile de excepții din clasa de bază Exception.