Session 4 - C#

* Boxing & unBoxing in Casting

* Boxing Casting From ValueType to Reference Type (implicitly)
* unBoxing Casting From RT to Value Type

ex1

```
object obj;
object obj = new object();
obj = "Ahmed";
// string class
```

RT

```
obj = 'A';
```

RT — char — VT // Casting Boxing

```
obj = 3;
```

RT — int — VT // Casting Boxing

ex

```
int x = 10;
object o1 = x;
```

Take copy of x

Reference Parent → Child VT (int)

// Casting From int (VT) to object (RT)

object obj;
// Declare for Reference of Type object Refering to Default value = Null
// This Refering 'obj' can refer to instance of Type 'object' OR another Type inherting from object

// Parent = Child is a Parent
Boxing Take Child copy Put in object

* Boxing (implicitly) → يعني مش محتاج
Safe Casting هو كده syntax اعمل

[ex] object obj2;

obj2 = 5;

int y = obj2; [XX] unsafe
VT RT

Make Explicit Casting

* Explict Casting (un Boxing)

(*) UT RT unSafe Casting

int y = obj2 ;

لازم تضمن انه شايل int

// Child = Parent ;

Parent is a Child [XX]

عشان كده لازم اضمن للcompiler أني مش هحصل تعارض

// Dog = Animal

مضمنش أن ال Animal اللي شايل هو Dog

محتاج اضمن انه الي شايل فعلا Dog

int y = (int) obj2; (unSafe)

* Boxing , unBoxing behavior is [bad] for [Performance]

يجب تجنبها

* Nullable Types

① ↳ Nullable Type - of Value Type
struct & enum

* null is not vaild Value For Variables of Type ⇒ VT. Datatype

EX

* int Age;
  Age = 22;
  Age = Null; error not vaild

EX For Nullable Type
int? Age; // Age is Nullable int
Age = Null; Valid

* int? ⇒ allow integer Values + Null

EX1
double? Salary; // Nullable Double
Salary = null; // Valid

* Value Types Nullable
is ValueType & allow 'Null' as Value.

* Casting with Nullable

ex
```
int X = 5;
// X : int IS Allow intValues only
int? y = x;
// y Nullable → (intValues) + Null
```

int? y = x; Implicit Safe

~~int~~ ~~y~~

X = y; // unSafe Casting

(int) only (int, Null) // EXPLICIT

ex
```
X = (int) y; // unSafe
```

To Convert ~~From~~ ~~Nullable T~~ ValueType TO NullableType

Syntax
```
Nullable<int> Age;
≡ int? Age;
```

// Protective Code

```
IF (X.HasValue)
{ y = (int) x; } → y = x.Value;
else                  or
{ y = 0; }
```

(X)

or another way

```
IF(X != null) or IF(X is not null)
    y = (int) X;
else
    y = 0;
```

or another way using Ternary condition

```
y = X.HashValue? X.Value : 0;
```

or By using Null Coalescing operator ??

```
y = X ?? 0; // Syntax Sugar
```

---

② Nullable ~~Types~~ Types of RT

*null is default value for Reference Type variables

```
String Message = null;
  └ class (RT)
```

```
String? Message = "Hi";
```

* ليه بنستخدم RT مع Nullable بالرغم من أن RT
↳ has null as default value
↓ ex1 for explain
class Employee
{class String? FName { }
Public
class String LName{ }
}
* الاتنين بيقبلوا null لكن في حالة String مش Nullable هو مش هيعمل check بس هيعمل warning عشان مش أكتر

String? FName; → هيعمل دايما مش هيحتاج initialize
String Lname; = "" ↳ في حالة هيحتاج أو بيعمل initial

Compiler أكثر فرق في حالة
Check
Make warning with RT without Nullable RT

(*)

ex

```
String? notnull = "Hi";
String? Nullable = default;
                   "null"
notnull = Nullable; //null (!)
                    Forgiveness
  notnull = Nullable;
or
without ? with notnull
  notnull = Nullable!;
```

(*)

*null Propagation operator

```
double X = default;
           keyword
default of double
```

```
int [] Numbers = default!; //Null
For (int i = 0 ; i < Numbers.Len; i++)
{ ممكن حاجة هيضرب
  Console.WriteLine(Numbers[i]);}
```

هيلاقي دي ب False مش هيضربه عشان

```
For (int i=0; (Numbers != Null) && (i< numbers.length) ; i++)
```

→ Short Circuit

لو مش هيعمل check للتانية لو الأولى false

or (X)

```
For (int i=0; Numbers?.length; i++)
// Number != null ? Numbers.length : null
```

→ checking

* قبل ما أكتب دايما الحاجة دي ممكن تشيل null اعمل check أو Nullable عشان ميضربش exception

but Previous example have overhead if array have elements

كل مرة هيتأكد من element في array هيعمل check

لو array ب null و دي الـ overhead محتاجه

To Solve this

```
if ( array != null )
{
    For (int i=0; i< array.length; i++)
    { CW( array[i]); }
```

int [] Numbers = default;

int len = Numbers.length;
//un safe Code May Throw exception (Null Reference Exception)
المعنى reference مش بيشاور على حاجة

improving

int? len = Numbers?.length;

or Checks دي ومعرف اعمل null Propagation

int len = Numbers?.length ?? 0;

~~Propagation~~ ل

null Crossing operator

ex if left == null → return 0
else return

Employee emp = new Employee();
ممكن يكون بيشاور reference على null

if (employee != null)
{
if (employee

}

OR

Console.WriteLine(employee ?. Depart?

* Functions Modularity
  - not return → Void
  - return → returnType
* Calling → Name, Arguments
→ Functions using to encapsulat block of code (not repeate code) usability, Mauntainability
  - improving
* Function Must achieve Single Responsibilty
* Function → ① signature
  ② behavior → Prototype =
  - signature
    - → return Type
    - → Name
    - → Parameters
  - behavior → body

* Must determine function

Class Member ← → object Mem

* object Member to Call Fun
Must Create object then Called
based on object
non Static

* Class Member (class ← Static)

Ex1

signature

```
Static Void PrintShapes()
{
    For (int i=0; i<=10; i++)
    {
        cw($"*_*");
    }
}
```

behavior

// For calling
in Main of Class

Program.PrintshaPes();
nameofclass. nameof Fun();
لو بنادي فانكشن موجودة جوها
ال class مش محتاجة اكتب اسمه
لكن لو هي في class تاني يبقى محتاجه
اكتب اسم ال class

* Fun with Parameter

```
static Void Print (int count)
{
    //body
}
//Calling in Main of class

Print ( 1 ); //count = 1
```

* If you have More Than one Param

```
Print(Patern, Count);
```

// ادخلهم بالترتيب

or لو مش عايزة ادخلهم بالترتيب

```
Print (Count: 1, Patern: "$-$")
```

لازم اكتبهم

* You can Make default Value for parmeter of Fun

* لو عندى اكتر من Paramter مثلاً واحد ملوش Default Value ← لازم يتكتب فى الاول

Print (int X, int Count = 1);

→ Console.WriteLine("Hi \n Ahmed \t Nasr");

\n → بتنزل سطر جديد وتكتب

\t → = 4 Space

\\ → \ itself

\r →

@ →
@" \n \r " \ بتلغى دول

\\t → SKiP worked of \t

@

String FolderPath = @"\D:\Downl\";

لو مكتوش حاجة تحتها \D لو اقصد \ نفسها ← // = أو احيب @ قبل ال string

* Paramter of Function

① Parameter as Value Type

```
static void Swap(int x, int y)
{
    int Tmp = x;
    x = y;
    y = Tmp;
}
```

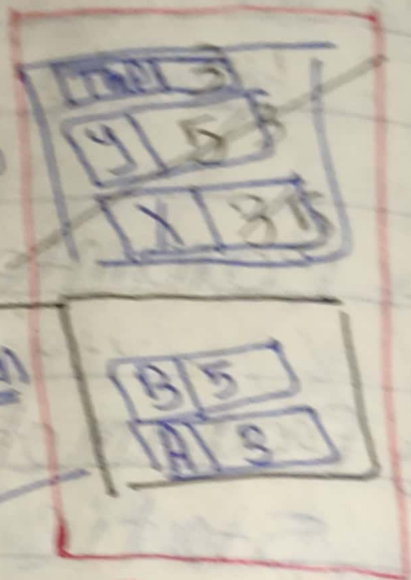
```
// Calling in Main
int A = 3, B = 5;
Console.WriteLine($"A = {A}");
  "       "        "  ($"B = {B}");
Swap(A, B);
   // Passing by Value
X → A = 3
```

① → Passing by value [InPut Parameter]

:- Parameter هيمثل Fun تاع Fram of Swap

هيبعت نسخة من B, A عشان كده الValues هيتم انتقال من الـ متغير الـ Main Values

stack Fram of Fun ex of Main



Stack

have local Variable & any Parameter of Main as Fun

Main not have Parameter

* بعد Swap خلاص بيمسح Frame عشان كده مش هيحصل Swaping

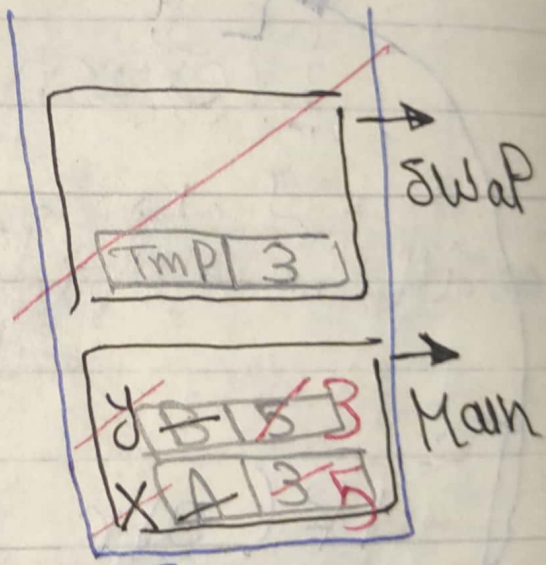
② → Passing by Ref [Parameters inPut outPut]
↳ Reference

```
static void swap(ref int x, ref int y)
{ //body }

// Calling
swap(ref A, ref B);
```

هو هنا باعتبار الـ A و B تفسهم مش نسخة منهم

ref B , ref A لما بابعت
X, Y بيعيد تسمية B, A
مش باعتة نسخة فبعدل
تفسهم B , A

بتشاور
Y → B
X → A



لما بخلص swap بعمل
Frame ال
بتاعها
فبرجع الـ addresses القديمة
A, B
بدل
X, Y

(2) Parameters as Reference Type

(*)

ex (1) Passing by value

```
Static int SumArr(int [] arr)
{
  int Sum = 0;
  For(int i=0; i<arr?.length; i++)
  {
    Sum += arr[i];
  }
  return Sum;
}
```

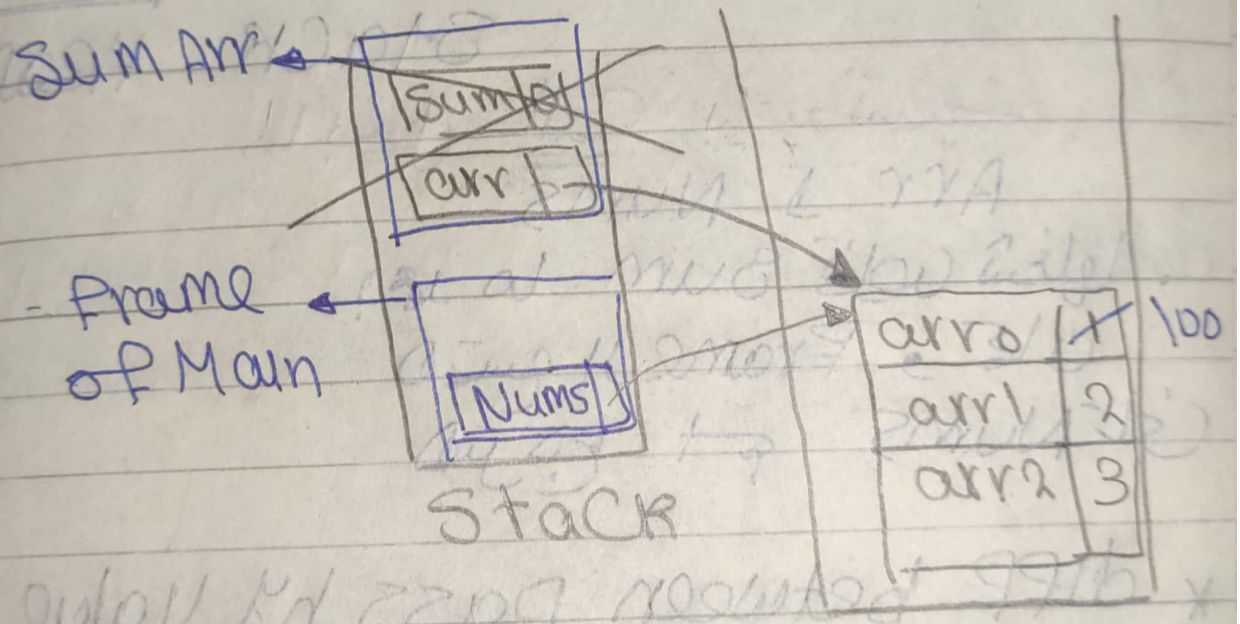
لكن ف overhead كل مرة هياخد element هيعمل check ← notnull

ممكن ميعملش array يبعتلي null ف الحالة دى عندى مشكلة

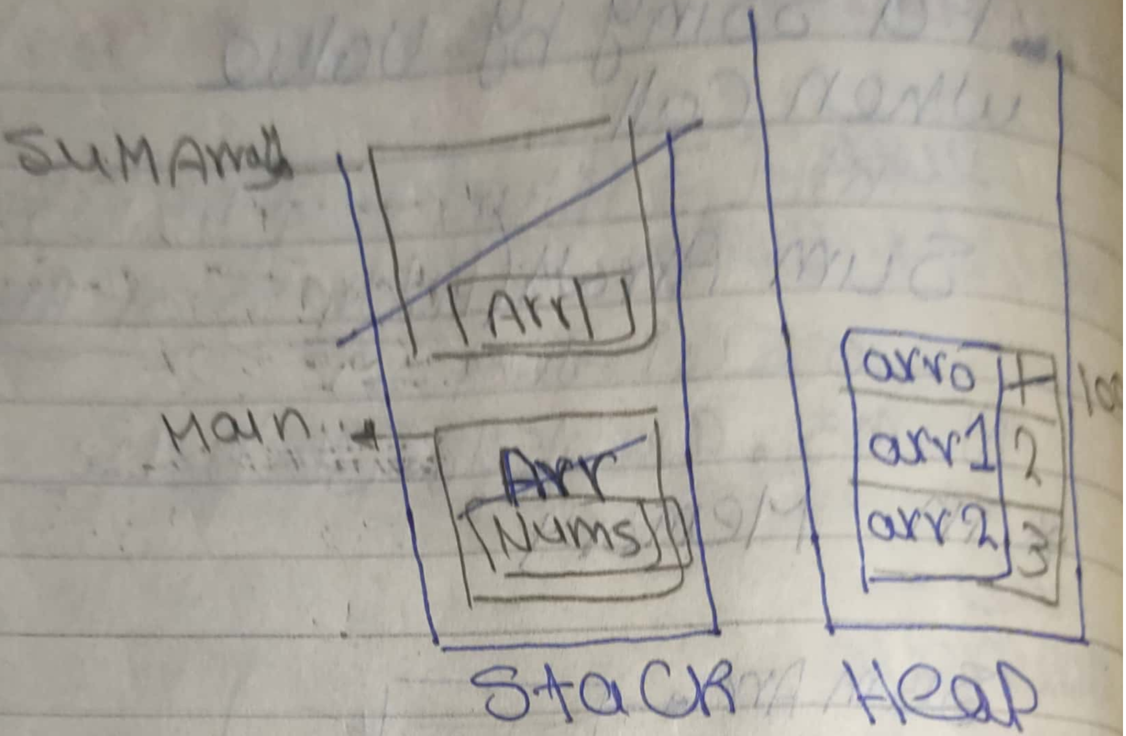
```
if(arr is not null)
{ For (int i=0 ; i<arr.length; i++)
  Sum += arr[i]; }

arr[0] = 100;
```

انا بغير ف array

→ Passing by Value
when Call
int [] Nums = {1,2,3}
Sum Array(Address of Array)
هيشاور على نفس الكان
Nums, arr
* in Memory



② Passing by Ref
Calling → Fun (ref int[] arr)
→ Sum Array(ref Numbers)
Reference of
Array → can access
array ال Sum of Main
Nums → alise name (parameter of Fun)

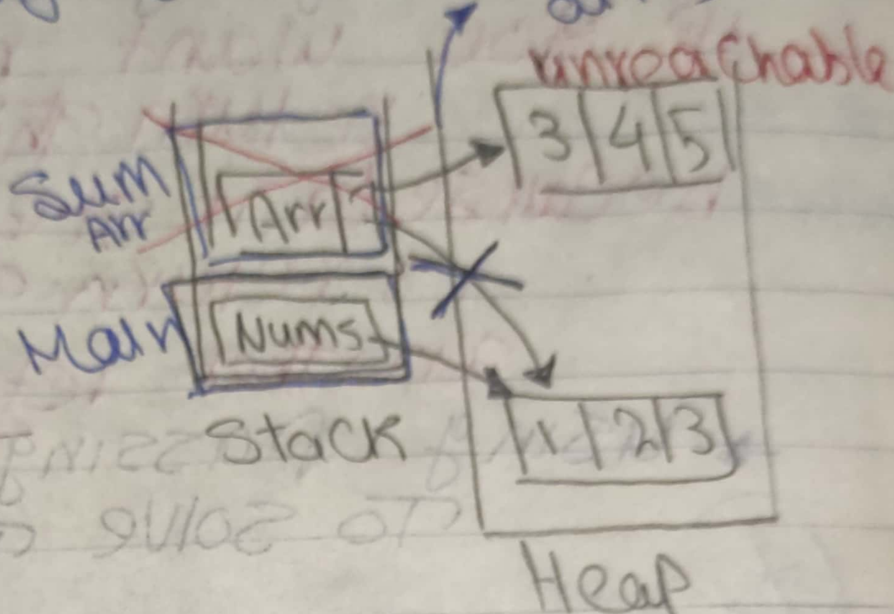


لما بعمل call الـ بيعيد تسمية
Arr لـ Nums
بعد ما Sum تخلص شغلها
هتمسح الـ Frame ، كمان
هترجع تم الـ Nums

* diff between pass by value Vs
Pass by Ref with RT Reference Type

```
static int Sumarr (int[] arr)
{
    ~~int []~~ arr = new int [] {3,4,5};   ← ref
    // body
}
```

(input only) ← Passing by Value



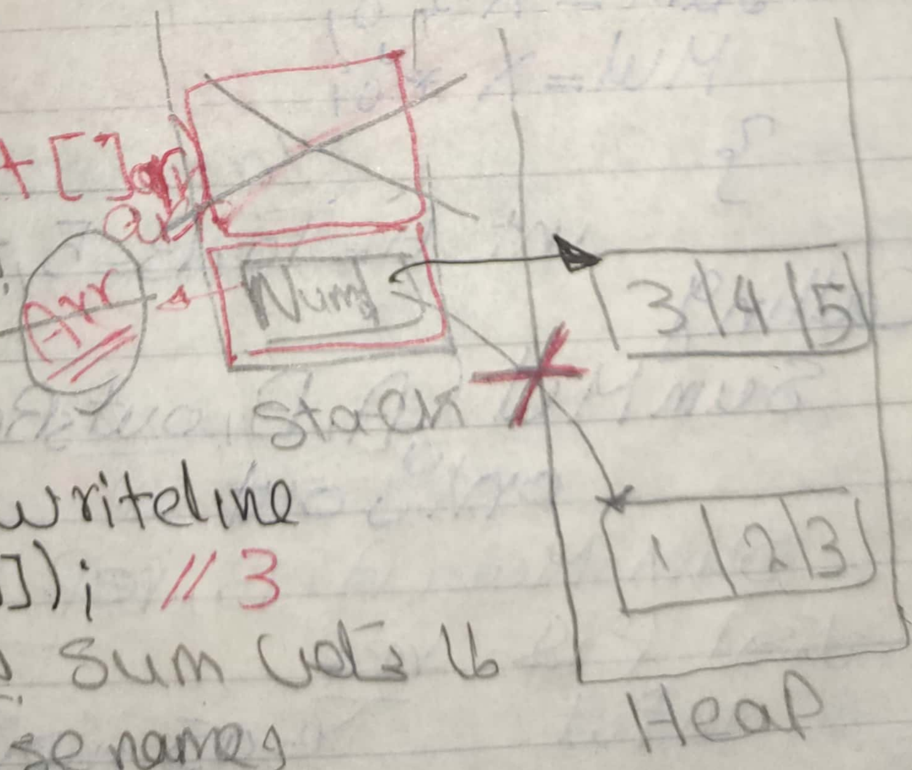
Console.WriteLine(Num[0]); //1

Passing by Ref (input, output)

Call SumArr(ref Nums)

in Fun

(ref int[] arr)



Console.WriteLine
(Num[0]); //3

Frame بيمسح Sum لما يخلص

Arr ← alise names

access

③ Passing by out (out Parameter)

If you want return More than return this not allowed because last step in Fun is return That another return not exist

→ using Passing by out To solve this

*

```
Static Void SumMul
(int X, int y, out int Sum,
                out int Mul)
{
  Sum = X + y;
  Mul = X * y;
}

int A=10, B=5, SResult, MResult;
//Calling
SumMul(A, B, out SResult, out MResult);
```

only out

SumMul(A, B, ref SResult, ref MResult);

ref يجب ارسالها input

as input output

* اعرف Call بتاعك Vars

Sum Mul (int A, int B, out int SRes,
out int MRes);

Assign لكن في الحالة دي لازم اعمل
في الـ Fun هيعمش يرجع حاجة
فاضي

* لكن مع بتاع اعمل rep
initialize
الاول ...

int A=5;
(rep out A)

rep
as input
then output

* Keyword Params
* Fun Take only one Params
as Type of array as Parameter;

لو عندي مجموعة مثلا رقم
Then ① Parameter without intialVal
② // with intialVal

Then ③ Params as Paramter

(*)

// int [] Nums = {1,2,3,4,5,6,7};

x y array

int Sum = SumArr (1,2,3,4,5,6,7

لازم Params لانها array اعرف كتابة من

Console.WriteLine

("{0}+{1}+{3}={2} {4}", 10, 20, 35,

"Hamada");

(*)

without initial

with initial

Static int SumArr (int X, int y=5,

any in With array Params int [] arr

#

{

int Sum=0

if (arr is not null)

For (int i=0; i<arr.Length; i++)

Sum += arr [i];

return Sum;

}

* How Can Write Desensive or Protective Code?

① Exception → this unexpected code

يعني أنا بـ Throw exception لو أنا مش متوقعة
او معرفش الكود ممكن يضرب فين لكنا
أنا مطلوب مني handle الـ code
في كل الحالات اللي متوقع تحصل و
أفترض دفاع عندي هو Try, catch

(*)

EX/ For Code not handled

```
Static void Dosomecode()
{
    int x, y, z;
    x = int.Parse(Console.Readline());
    y = int.parse(Console.Readline());
    z = x / y;
    int [] arr = {1,2,3}
    arr [100] = 22;
}
```

Pervious Code ← مثال بـ

① لو الـ user دخل Type غير int
x ^ y

② لو الـ user دخل الـ y=0
مينفعش اقسم على صفر

③ 1001 دخلت الـ length بتاع array
وكمان كان لازم اتأكد ان
array is not null

Developer Must handle all Cases of your Code
Then
using Try | Catch

↓

Catch يجب دخلها
او
Catch _ Finally
او
Finally

وده علشان لو حصل exception في اي حالة
غير الـ handle هيفرق الكود وكمان
بيكمل تنفيذ الكود عادي
مش هيقف

Catch → using For if have any exception throw this excep
↓
in Try

Finally → using to Deallocate or Free or Delete or Release or Close un Managed Resources
Managed كل حاجة تقريبا
by CLR
لكنا في حالة
أنت الي فتحت او عملت بإيدك مثلا
Connection With DB
انت لازم تقفلها عشان كده
Finally نستخدم

EXI For Handled Perviouse EXI

(*)

```
Static Void ProtectiveCode()
{ int x, y, z;
  bool Flag;
  do { Console.WriteLine("Please Enter x");
    Flag = int.TryParse(Console.Read, out x);
  } While (!Flag)
do { Console.WriteLine("Please Enter y");
  Flag = int.TryParse(Console.Read, out y)
} While (!Flag || y == 0)
z = x/y
int [] arr = {1,2,3};
IF (arr?.length >= 100)
  { Arr[100] = 22; }
```

(*)

Then

```
Try
{
  ProtectiveCode();
}

Catch(EXCEPTION ex)
 {
   ex.
  Console.Writeline(ex.Message);
 } // ex المفروض
Finally
 {
   Console.WriteLine("Finally");
 }
```

لك في حالة حدوث ex في مسدج

لك عشان اتأكد انه دخل ونفذ Finally

Exception Parent class
Inhert from Exception

① System Exception Child for Parent For These
① Format Exception
② indexoutof Range Exception
③ Null Reference "
④ inVaild oPeration "
⑤ Arithmetic "

5.① Divided By Zero Exception
5.② oVerFlow "

② APPli Cation EXCeption