Extension Methods

هي function زي أي function تانيه ولكن ليها شروط و syntax مختلف شويه.

ومن اسمها بنستخدمها عشان نعمل extension لكلاس تاني من غير ما نعدل في الكلاس نفسه.

Syntax:

public static class int\_extensions

{

public static bool is\_even (this int num)

{

return num%2==0;

}

public static bool is\_odd (this int num)

{

return !is\_even(num);

}

}

Call syntax:

static void Main(string[] args)

{

int s=0;

Console.WriteLine(s.is\_even());

Console.WriteLine(s.is\_odd());

}

في المثال اللي فوق ده كنت عاوز اضيف ميزه لكلاس int عشان اعرف هل الرقم زوجي ولا فردي فعملت كلاس جديد سميته int\_extensions ونلاحظ هنا انه مش تابع لكلاس int ولا حتى بيورث منه حاجه وضفت فيه 2 methods الأولى بتتأكد اذا كان زوجي ولا لا والتانيه بتتأكد اذا كان فردي او لا.

نلاحظ في الsyntax بتاع كل method ان الparameter قبله كلمة this وده السبب في ان لما عملت call للmethod عملته باستخدام الobject s اللي من نوع int مش باسم الكلاس وهنا بيجي دور this لانها بتعود على الobject اللي بيستخدم الmethod

نلاحظ كمان ان عشان اعمل extension Method لازم الكلاس والmethod يكونو static

Nullable & Null operators

في أوقات كتير بيكون عندي متغير (value type)او reference type ولسبب ما مش محتاج اخصصله قيمه في الوقت الحالي في الحالة دي بتيجي وظيفة null كreserved word in C# بتعبر عن اللا قيمة  
ولكن منقدرش نستخدمها الا مع الnullable types سواء كانت value او reference

ايه هي الnullable types ؟   
هي أي حاجه من الممكن ان ميكونش ليها قيمة واللي بيحدد ده هو المبرمج وقت تعريف متغير مثلا فبيضيف علامة استفهام بعد نوع المتغير او بيستعين بكلاس <Nullable> و......الخ   
ودي بعض الامثله لتعريف متغير nullable قيمته بnull:

int? num1 = null;

Nullable<int> num2 = null;

Nullable<int> num3 = new Nullable<int>();

Nullable<int> num4 = default(int?);

A diagram of a function

Description automatically generated with medium confidenceالجدير بالذكر ان لما نعرف nullable type ده بيضيف 2 read only properties للobject اللي عرفناه الأولى اسمها HasValue وظيفتها ترجع قيمه boolean هتكون true اذا فيه value و false لو مفيش والتانيه بترجع الvalue نفسها زي ما متوضح في الصورة:

وزي ما فيه nullable type هيبقى على النقيد من ذلك فيه non nullable type وهو الشائع والأكثر استخداما بين المبرمجين زي تعريف متغير int قيمته ب 5 مثلا وبما انه non nullable يعني لو اسندنا له قيمة null مش هينفع زي المثال ده:

int num1 = null;

وأخيرا عشان نضمن سلامة الكود واحنا بنتعامل مع الnullable types دي ومنستخدمش methods معتمده على قيمة وهي في الأساس ممكن تكون مش موجوده(null) لازم نتأكد الأول هل قيمة المتغير ده مثلا وده عن طريق شرط بسيط زي المثال ده:

char[] word = null;

int? num\_of\_chars = 5;

if (word != null)

num\_of\_chars = word.Length;

Console.WriteLine($"number of characters in the world: {num\_of\_chars}");

ولكن C# اختصرت الشرط ده في ما يسمى ب the null conditional operator فهيختصر الشرط اللي فوق ده كالتالي:

num\_of\_chars = word?.Length;

Console.WriteLine($"number of characters in the world: {num\_of\_chars}");

تاني null operator وهو the null coalescing operator باختصار هو شرط اذا المتغير اللي بنستخدمه فيه قيمه هنستخدمها واذا مكانش فيه قيمه يعني قيمته ب null هنستخدم القيمة الdefault اللي بعد علامتين الاسفهام زي المثال اللي تحت واللي هتكون قيمة num2 فيه في اول مره ب -1 لان قيمة num1 ب null وتاني مره ب 10.

int? num1 = null;

int num2 = num1 ?? -1;

num1 = 10;

num2 = num1 ?? -1;

Exception Handling

هو الجزء المسؤول عن معالجة الأخطاء اللي ممكن تظهر وقت ال debuggingبتاع البرنامج

في حالتين للموضوع ده:

الأولى ان المبرمج هو اللي عاوز يرمي exception لسبب ما وفي الحاله دي بيستخدم throw كالتالي:

throw new DivideByZeroException();

والتانيه ان الexception حصل بسبب خطأ في البرنامج وهنا بنستخدم try ثم catch ثم finally كالتالي:

try

{

int x = 1;

int y =0;

x = x/y;

}

catch (Exception ex)

{

Console.WriteLine("Divide By Zero exception ):");

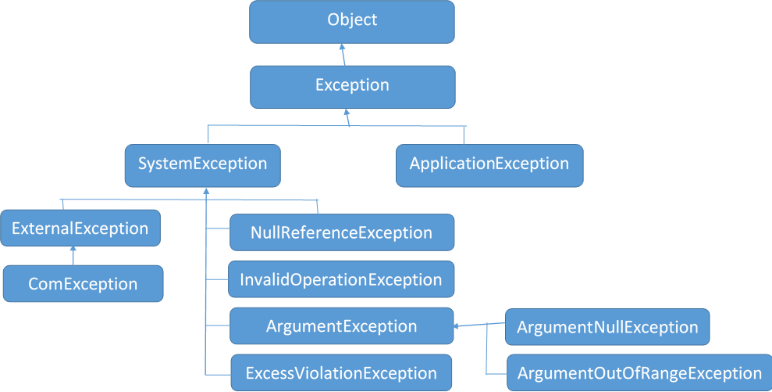
}

finally {

Console.WriteLine("End");

}

في المثال اللي فوق حصل exception بسبب عملية القسمه على 0 اللي رياضيا تعتبر من المحرمات💀 ولهاذا السبب حطينا الكود اللي بيعمل الexception ده في try block عشان اذا حصل نقدر نتعامل مع الexception ده في الcatch block طيب ايه فايدة finally في الحاله دي ؟ عشان نفهم وظيفة finally لازم نعرف أنواع الexceptions اللي بتدعمها c# واللي موضحه في الصوره اللي تحت:



في أنواع كتير للexceptions منها اللي جاهز تقدر تستخدمه وهو ما يندرج تحت الsystem exception class ومنها اللي تقدر تنشئه وفي الحاله دي بيعتبر application exception بشرط انه يورث من ال class Exception وسواء كان application او system exception فكلهم بيندرجو تحت الsuper class “Exception” ايه الفايده من ده ؟ لنفترض انت عامل 2 catch للمثال اللي فوق كالتالي:

catch(DivideByZeroException ex)

{

Console.WriteLine("Divide By Zero exception ):");

}

catch (Exception ex)

{

Console.WriteLine("general Exception handler");

}

في المثال اللي فوق ده لو حصل قسمه على الصفر الcatch block الأولى(DivideByZeroException) هي اللي هتشتغل وهي اللي هتعالج الexception اللي حصل لكن لو بدلنا أماكن الcatch blocks اللي تحت فوق واللي فوق تحت زي كده مثلا :

catch (Exception ex)

{

Console.WriteLine("general Exception handler");

}

catch(DivideByZeroException ex)

{

Console.WriteLine("Divide By Zero exception ):");

}

catch(NullReferenceException ex)

{

Console.WriteLine("Null Reference Exception ):");

}

هنا اول catch هيستقبل أي exception هيقابله والباقي يعتبر من غير فايده ليه ده بيحصل ؟ لانه ال supper class لباقي ال exception classes فاي exception هيعدي عليه هيعتبر نفسه جزء منه فهيتنفذ فيه مش في غيره عشان كده لازم نرتب الcatch بحسبب ترتيبها في شجرة العيله لل exception classes بحيث الأبناء يبقو فوق والآباء يبقو تحت

And finally يجي دور finally واستخدامها لنفترض ان ولا catch قدر يعالج الexception اللي حصل ده هنا يجي دور finally اللي بتعتبر الbase case فبتنفذ الكود الأخير في أي حالة exception حصلت سواء الcatch عالجتها او لا .

Asynchronous programming

المسار الطبيعي لاي برنامج مش بيستخدم ال asynchronous programmingانه بيبدأ ينفذ الكود اللي في الmain امر امر واذا مثلا قابل call ل function بيروح ينفذها وبيوقف تنفيذه في الmain لحد ما يكمل الfunction ويرجع يكمل تنفيذ في الmain من تاني لكن الأسلوب ده ليه عيوبه لان لو عندنا 2 functions مش معتمدين على بعض يعني مفيش واحده مضطره تستنى التانيه عشان تشتغل ففي الحاله دي احنا ضيعنا فرصة اننا نخليهم يشتغلو مع بض وهنا يجي دور ال asynchronous programming في انه بيوفرلنا الفرصة دي.  
C# بتوفر الthreads كحل للمشكله اللي فوق ولكنه مش افضل وسيله في وجود الabstractoin منه وهو الtask كلهم بيأدو نفس الغرض باختلاف الsyntax ودي مقارنه بسيطه بينهم:



Syntax for both of them:

Thread thread = new Thread(() => { Console.WriteLine("hi 1"); });

Task task = new Task(() => { Console.WriteLine("hi 2"); });

thread.Start();

task.Start();

output for the above code:

A black background with white text

Description automatically generatedA black background with white text

Description automatically generated

لو جربنا الكود اللي فوق اكتر من مره هنلاحظ ان الoutput ممكن يختلف كل مره وده لان الtread والtask بيشتغلو في نفس الوقت واللي بيخلص الأول بيظهر الناتج قبل التاني .

لنفترض ان الtask شغال على function ولكن الfunction دي بتاخد وقت عشان تتنفذ وانا عاوز البرنامج يكمل ولما الfunction تخلص تبقى تطبع الناتج مثلا الحالة دي ممكن نطبقها باكتر من طريقه كالتالي:

Using GetAwaiter() and OnCompleted():

static void Main(string[] args)

{

Task<string> task = Task.Run(getname);

var awaiter = task.GetAwaiter();

awaiter.OnCompleted(() =>

{

Console.WriteLine(awaiter.GetResult());

});

}

static string getname()

{

var name = "ahmed";

return name!;

}

Using ContinueWith():

Task<string> task = Task.Run(getname);

task.ContinueWith((x) => Console.WriteLine(x.Result));

Console.WriteLine("wating for the result ...");

على النقيض من ذكل في بعض الأحيان بنبقى بننفذ الtask بشكل Asynchronously ولكن بنحتاج نستنى نتيجه معينه عشان نقدر نكمل وهنا تيجي أهمية await و async.

Example:

static async Task Main(string[] args)

{

await ClcAgeAsync();

Console.WriteLine("wating for the result ...");

}

static async Task ClcAgeAsync()

{

int age=0;

await Task.Run(() => age = System.DateTime.Now.Year - GetBirthDateYear());

Console.WriteLine("age is:"+age);

}

نلاحظ اننا لازم نستخدم async قبل اسم الmethod واذا مش بترجع حاجه هنكتب جنبها Task واذا بترجع حاجه هنكتب Task<return data type>اما بالنسبة لawait فبتيجي مع الfunction او الtask اللي مش عاوزين نتخطاه الا لما يكمل اللي بيعمله .

الfunction اللي اسمها ClcAgeAsync هنا بتسمى بasynchronous function.