C# - Apuntes diversos

Page • 1 enlace entrante • Tag

Conceptos diversos

- **base (keyword) → Acceso a la clase base (sus métodos)**
- Campo estático → Variable accesible a través de la clase en la que fue definido (NO a través de las instancias)
- **Clase derivada → Hereda todos los elementos de la clase base menos los constructores y los finalizadores**
- Const → Valores inmutables, se conocen en tiempo de compilación
- System.Environment.SpecialFolder → Enum que contiene los directorios especiales de cada sistema operativo
- **Finalizador → Se ejecuta cuando pasa el garbage collector por la instancia. Utilizado para liberar objetos que usan recursos no administrados.**
- **Herencia → Permite crear clases que reutilizan, extienden, y modifican el comportamiento de otras clases**
- Indizador → Definición de cómo aplicar el operador a los objetos de una clase
- **internal (keyword) → Solo es público dentro del ensamblado, no afuera**
- Miembros estáticos → Miembros que pertenecen a la propia clase (o tipo) en lugar de a un objeto determinado (instancia)
- **nameof(obj) —>** devuelve el nombre de una variable, clase u método
- **Objeto polimórfico → Objeto que en tiempo de ejecución puede adoptar cualquier forma de un tipo no idéntico a su tipo declarado**
- **Principio Open-Close → Una entidad de software debe quedarse abierta para su extensión, pero cerrada para su modificación**
- Propiedad → Integra los conceptos de campo y método. Internamente se codifican con los bloques de código
- **protected (keyword) → Permite que las variables se puedan ver abajo en la jerarquía**
- readonly → Tienen una función similar a las constantes pero sin sus restricciones (variables de sólo lectura). Solo pueden asignarse en su

declaración y dentro de un constructor. Se asignan en tiempo de ejecución como cualquier variable

- **struct →** Normalmente, los tipos de estructura se usan para diseñar tipos de pequeño tamaño centrados en datos que proporcionan poco o ningún comportamiento
- Utility Class → Clases que no tienen nada más que miembros estáticos (se les llama así porque solo aportan funcionalidades)

Const

- Implícitamente estáticos
- Se inicializan cuando las declaran
- Solo se pueden definir con tipos integrados numéricos, char, bool, o String
- Pre-procesamiento

Constructor estático

- No se pueden invocar explícitamente
- Es invocado por el motor de ejecución de .NET una única vez al principio del programa
- No se puede sobrecargar

Modificador static

- Keyword →static
- Los métodos y las variables estáticas son compartidas por todas las instancias de la clase, pero mo pueden ser accedidas con referencias de instancias
- Memoria Heap → Podemos pensar a una clase como un objeto único, independientemente de sus instancias, ya que cuenta con sus propias variables y métodos
- Nomenclatura convencional para campos privados estáticos →s_
- Sintaxis → <nombre_clase>

Propiedad

- Propiedad auto-implementada → Si no se escriben los campos del get{} y el set{}, pero igualmente se los escribe, entonces el compilador les asignará automáticamente el código necesario para leer y escribirlas

public int Nivel {get;set;}

- Propiedad de solo escritura → Propiedad que solo tiene implementado el bloque set*{}.* (Su uso es muy raro)
- Propiedad de solo lectura → Propiedad que solo tiene implementado el bloque get{}
- Sintaxis → { get{}; set{}; }
- Sintaxis tipo flecha

public int Nivel

{

get => _variable;

set => _variable = valor;

}

Interfaces

Es un tipo de referencia que especifica un conjunto de funciones sin implementarlas, son como una clase abstracta. Sirve para crear nuevas jerarquías (que no dependan de System.Object)

- Comienzan con una I (convención)
- Sus miembros son públicos de manera predeterminada
- Pueden especificar métodos, propiedades, indizadores, y eventos de instancia
- No se pueden instanciar
- No pueden contener campos/constructores de instancia ni finalizadores

Sintaxis

class Test : BaseClass, IInterfaz1, IInterfaz2, ...

{}

interface IInterfaz1

{}**

System.Collections

IComparable

El método CompareTo aprovecha que los elementos que ordena implementan la interfaz IComparable. Si se quiere usar el método Array.Sort() para ordenar un

vector con elementos de una clase que fueron definidos por el usuario, se debe implementar un método CompareTo dentro de la clase.

IComparer

Es esencialmente lo mismo que el anterior, nada más que permite desacoplamiento porque se puede crear una clase que contenga la lógica de comparación para una clase, y esta clase se la puede pasar como parámetro a un Array.Sort()

IEnumerable

Si se lo utiliza en un tipo, lo vuelve enumerable. Permite implementar un método IEnumerator GetEnumerator(), que permite la enumeración del tipo. Si no se desea construir un enumerador propio, se puede utilizar el método .GetEnumerator()

IEnumerator

Un enumerador es un objeto que puede devolver los elementos de una colección, uno por uno, en orden, según se solicite.

Permite implementar un enumerador (con métodos Current(), MoveNext(), y Reset()) para cualquier clase.

Iterador

Permiten construir enumeradores y enumerables de manera mas sencilla (el trabajo lo realiza el compilador)

- bloque iterador → bloque de código que contiene una o mas sentencias yield. su tipo de retorno debe declararse de tipo IEnumerator o IEnumerable
- yield return → devuelve un elemento de una colección y mueve la posición al siguiente elemento
- yield break → detiene la iteración

Delegados

Tipo de clase cuyos objetos almacenados referencian a uno o mas métodos de manera de poder ejecutar en cadena esos métodos

- Permiten pasar métodos como parámetros
- Proporcionan un mecanismo para implementar eventos
- Sintaxis → delegate tipo funcion(params)
- Invoke() (invoca los métodos de los delegados de forma explícita)

Método anónimo

Clausura → un método accede a una variable externa a su ámbito

Expresión lambda

f = (int n) ⇒ {return n * 2};

f = n ⇒ n * 2;

linea = () ⇒ Console.WriteLine();

Material optativo

Delegados

Multidifusión → Los delegados pueden acumular funciones, como si fueran una cola, y las ejecutarán todas en orden cuando se invoque el delegado

.GetInvocationList() → Devuelve un arreglo de objetos Delegate que corresponden a las funciones delegadas encoladas

Operaciones → Si se declara al tipo Action como nullable, y se inicializa el objeto en null, se pueden realizar las siguientes operaciones:

- a = a + Metodo (si a era null, a ahora pasa a ser Metodo)
- a = a - Metodo (si no hay otro elemento encolado, a pasa a ser null)

Eventos

Los eventos son sucesos importantes que pueden ser notificados por algunos objetos y pueden ser recibidos por otras clases u objetos.

- El que produce el evento se denomina editor, y los que reciben la notificación se llaman suscriptores
- Los suscriptores codifican un método manejador del evento para realizar aquello cuando sean notificados
- Nunca se provocan eventos que no tienen suscriptores

Convenciones para los nombres → utilizar verbos en gerundio o en participio, dependiendo de si se produce antes o después del hecho

public delegate void EventHandler(object sender, EventArgs e)

Event - control de operaciones

Event permite ejecutar código cada vez que se añade o elimina un método del campo delegado, como si fuera una propiedad. Las operaciones con eventos solo son += y -=

Sintaxis:

//public event EventHandler NombreDelEvento; (sintaxis abreviada)

public event NombreDelEvento

{

add {}

remove {}

}

Excepciones

Su función esencial es proporcionar una forma estructurada, uniforme y con seguridad de tipos de controlar situaciones de error de nivel de sistema y de nivel de aplicación. Las excepciones solo pueden ser generadas por el runtime de .NET o el mismo código de una aplicación. Todas las excepciones derivan en última instancia de System.Exception

Escenarios comunes que requieren control de excepciones

- Entrada del usuario → formato incorrecto | fuera de intervalo
- Procesamiento de datos y cálculos
- Operaciones de entrada y salida de archivos → archivo no existe | está abierto | permisos
- Operaciones de base de datos
- Comunicación de red

Proceso de control de excepciones

Cuando se produce una excepción, el entorno de ejecución de .NET busca la cláusula catch más cercana que puede controlar la excepción. El proceso comienza con el método que provocó la excepción. En primer lugar, el método se examina para ver si el código que provocó la excepción está dentro de un bloque de código try. Si el código está dentro del bloque de código try, las cláusulas catch asociadas a la instrucción try se consideran en orden. Si las cláusulas catch no pueden controlar la excepción, se busca el método que llamó al método actual. Este método se examina para determinar si la llamada al método (al primer método) está dentro de un bloque de código try. Si la llamada está dentro de un bloque de código try, se tienen en cuenta las cláusulas catch asociadas. Este

proceso de búsqueda continúa hasta que se encuentra una cláusula catch que pueda controlar la excepción actual.

Algunas excepciones generadas por el compilador

- ArrayTipeMismatchException → Producida cuando una matriz no puede almacenar un elemento determinado porque el tipo real del eleemnto es incompatible con aquel de la matriz
- DivideByZeroException → Producida cuando se intenta dividir un valor entero entre cero
- FormatException → Producida cuando el formato de un argumento no es válido
- IndexOutOfRangeException → Producida cuando se intenta indexar una matriz y el índice es menor que cero o queda fuera de los límites de la matriz
- InvalidCastException → Producida cuando se produce un error en una conversión explícita de un tipo base en una interfaz o un tipo derivado en tiempo de ejecución
- NullReferenceException → Producida cuando se intenta hacer referencia a un objeto cuyo valor es nulo
- OverflowException → Producida cuando se desborda una operación aritmética en un contexto de comprobación

Algunos tipos de excepciones comunes

- ArgumentException o ArgumentNullException → Sirve cuando se llame a un método o constructor con un valor de argumento no válido o una referencia nula
- InvalidOperationException → Sirve cuando las condiciones de funcionamiento de un método no permiten la finalización correcta de una llamada determinada al método
- NotSupportedException → Sirve cuando no se admite una operación o característica
- IOException → Sirve cuando se produzca un error en una operación de entrada/salida

FormatException → Sirve cuando el formato de una cadena o datos es incorrecto

Operadores de prueba de tipos

is

Es un operador que comprueba si el tipo en tiempo de ejecución del resultado de una expresión es compatible con un tipo determinado(menor precisión)

- El primer operando no puede ser un método anónimo ni una expresión lambda
- El operador considera las conversiones boxing y unboxing pero no las conversiones numéricas
- El operador is no toma en consideración las conversiones definidas por el usuario

as

Es un operador que convierte explícitamente el resultado de una expresión en una referencia determinada o un tipo de valor que acepta valores null, y si esta no es posible, se devuelve null.

- No devuelve excepción

typeof()

Es un operador que obtiene la instancia System.Type para un tipo. El argumento del operador debe ser el nombre de un tipo un parámetro de tipo.
No se puede utilizar tipos que requieran anotaciones de metadatos como dynamic o string?

Array

- Clear() → Reemplaza con su valor predeterminado a un(os) elemento(s) específico(s)
- Resize() → Agrega o quita elementos
- Reverse() → Invierte el orden que tiene
- Sort() → Ordena (con algún criterio)

Convenciones

Bloque if

- Nunca use formularios de una sola línea (por ejemplo: if (marca) Console.WriteLine(flag);
- El uso de llaves siempre se acepta y es necesario si algún bloque de una instrucción compuesta if/else if/.../else usa llaves o si un único cuerpo de instrucciones abarca varias líneas.

- Las llaves solo se pueden omitir si el cuerpo de cada bloque asociado a una instrucción compuesta if/else if/.../else se coloca en una sola línea.

String

- Contains(str) → devuelve true si la cadena str está contenida dentro
- EndsWith(str) → devuelve true si termina con la cadena str
- Format()
- IndexOf(char/str) → devuelve la posición (indexada en base 0) de la primera ocurrencia de ese carácter o cadena. devuelve -1 si no encuentra ninguna coincidencia
- IndexOfAny(array) → similar al IndexOf(), pero devuelve cualquier primera ocurrencia de un valor dentro de la matriz
- Join(char, str) → concatena los elementos de una matriz con el separador indicado
- LastIndexOf(char/str) → similar al IndexOf(), pero se enfoca solo en la última ocurrencia
- PadLeft() → Completa el relleno hacia la izquierda
- PadRight()
- Remove(int) → devuelve una nueva cadena con todos los carácteres borrados a partir de int
- Split(char) → devuelve una matriz de subcadenas separadas por el separador indicado
- StartsWith(str) → devuelve true si comienza con la cadena str
- Substring(int) → devuelve una subcadena que comienza en int y sigue hasta el final
- ToCharArray() → devuelve una matriz de carácteres de la cadena
- TrimStart() → quita todos los carácteres de espacio en blanco iniciales

Sufijos literales

- f → literal float
- m → literal decimal

Tipos de datos

Enteros con signo

- sbyte → -128 to 127
- short→ -32768 to 32767
- int → -2147483648 to 2147483647
- long → -9223372036854775808 to 9223372036854775807

Enteros sin signo

- byte → 0 - 255
- ushort → 0 - 65535
- uint → 0 - 4294967295
- ulong → 0 - 18446744073709551615

Misceláneo

- break → Finaliza la iteración y transfiere el control a la instrucción que sigue
- continue → Transfiere la ejecución al final de la iteración actual
- Operador condicional ternario → (expresión booleana) ? (valor si es verdadera) : (y si es falsa)
- Typeof es más preciso que is
- El casting trunca y la conversión redondea
- Evitar los magic values hardcodeados
- Debug.Assert(bool, string) → Herramienta para detectar errores lógicos durante el desarrollo

params → Permite pasar un montón de argumentos a un método separados por coma. Si hay otros parámetros en el método, params debe ser colocado último de todos

youtu.be
What is C#? [Pt 1] | C# for Beginners
View full playlist:
https://aka.ms/dotnet/beginnervideos/youtube/csharpSet up C# in VS...

learn.microsoft.com
System.Diagnostics Espacio de nombres
Proporciona clases que permiten interactuar con procesos del sistema, registros de eventos y contadores de rendimiento.

struct

Tipo de valor que permite encapsular datos y funcionalidades relacionadas

```
Texto simple
public struct Coords
{
    public Coords(double x, double y)
    {
        X = x;
        Y = y;
    }

    public double X { get; }
    public double Y { get; }

    public override string ToString() => $"({X}, {Y})";
}
```

Tipos anónimos

Tuplas

Representan una composición simple de valores

Nulleable logic

Table 2.2 Truth table for `Nullable<bool>` operators

x	y	x & y	x | y	x ^ y	!x
true	true	true	true	false	false
true	false	false	true	true	false
true	null	null	**true**	null	false
false	true	false	true	true	true
false	false	false	false	false	true
false	null	**false**	null	null	true
null	true	null	**true**	null	null
null	false	**false**	null	null	null
null	null	null	null	null	null

Table 2.1 Examples of lifted operators applied to nullable integers

Expression	Lifted operator	Result
`-nullInt`	`int? -(int? x)`	`null`
`-five`	`int? -(int? x)`	`-5`
`five + nullInt`	`int? +(int? x, int? y)`	`null`
`five + five`	`int? +(int? x, int? y)`	`10`
`four & nullInt`	`int? &(int? x, int? y)`	`null`
`four & five`	`int? &(int? x, int? y)`	`4`
`nullInt == nullInt`	`bool ==(int? x, int? y)`	`true`
`five == five`	`bool ==(int? x, int? y)`	`true`
`five == nullInt`	`bool ==(int? x, int? y)`	`false`
`five == four`	`bool ==(int? x, int? y)`	`false`
`four < five`	`bool <(int? x, int? y)`	`true`
`nullInt < five`	`bool <(int? x, int? y)`	`false`
`five < nullInt`	`bool <(int? x, int? y)`	`false`
`nullInt < nullInt`	`bool <(int? x, int? y)`	`false`
`nullInt <= nullInt`	`bool <=(int? x, int? y)`	`false`

IEnumerable

IEnumerable → Secuencia que puede ser iterada

IEnumerator → Controlador de una secuencia

C# 3

- Autoprops
- Implicit typing
- Variables locales tipeadas implícitamente → el tipo de la variable es inferido por el compilador en compile-time
- Vectores tipeados implícitamente → Array = new[] { 1, 2, 3, 4 }
- **Object initializer** → Sintácticamente, es una secuencia de inicializadores de miembros, puestos entre corchetes → Orden = new Orden { Id = 1, ProductName = "Zapato" }
- Collection initializer → List<int> = { 1, 2, 3, 4 }
- Se puede omitir el constructor si se usa un **object initializer**
- Anonymous objecct creation expression
- Proyection initializer
- Lambda expression bodies, statement bodies

Expresiones lambda

Capturan las variables en si mismas (sus referencias) y NO el VALOR de las variables en el momento de la creación del delegado.

- Depende de cuantas variables se capturen en una expresión lambda (y de qué scopes sean estas), su código IL va a ser diferente → conviene revisar esto por temas de performance (ildasm.exe)

Expression trees

- La clase Expression<TDelegate> representa una expresión lambda fuertemente tipada como una estructura de datos con el formato de un árbol de expresión → SOLO LOS EXPRESSION-BODIED LAMBDA EXPRESSIONS pueden ser convertidos a expression trees
- Los objetos que sean de la clase Expression<TD> pueden ser *compilados* a una clase delegado de vuelta (func.Compile())

Extension Methods

Solo se pueden crear en una clase no-genérica estática. El primer parámetro es el tipo extendido

```
Texto simple
public static Instant ToInstant(this DateTimeOffset dateTimeOffset)
{
    return Instant.FromDateTimeOffset(dateTimeOffset);
}
```

Es importante recalcar que si *la clase del tipo extendido* tiene un método con el mismo nombre, el compilador primero va a agarrar ese método antes que el **método extendido**

- Permite el Method Chaining

Query Expressions

Son traducciones sintacticas que ocurren antes de cualquier resolución de sobrecarga o binding

```
Texto simple
string[] words = { "keys", "coat", "laptop", "bottle" };
IEnumerable<string> query = from word in words
            where word.Length > 4
            orderby word
            select word.ToUpper();
```

```
foreach (string word in query)
{
            Console.WriteLine(word);
}
```

- transparent identifier
- https://dotnet.microsoft.com/en-us/apps/aspnet

Directory - Algunas funciones útiles

- Directory.CreateDirectory(filePath)
- Directory.Exists(filePath)
- Directory.EnumerateDirectories(".", *searchPattern,* SearchOption.AllDirectories)
- Directory.EnumerateFiles(".", *.txt)
- Directory.GetCurrentDirectory()

File - Algunas funciones útiles

- **File.WriteAllText(filePath, content)** → si el archivo ya existe, se sobrescribe
- File.ReadAllText(filePath)
- File.AppendAllText(filePath) → anexar nuevos datos

Path - Algunas funciones útiles

- **Path.DirectorySeparatorChar** → el carácter que separa los directorios en una ruta de directorios de un SO
- **Path.Combine("carpetaSuperior", "carpetaInferior")** → crea un Path de la forma: *carpetaSuperior\carpetaInferior*
- Path.GetExtension(extension)

Newtonsoft.Json - Librería para analizar y manipular datos Json

- dotnet add package Newtonsoft.Json
- **JsonConvert.DeserializeObject(obj)** → lee un objeto Json (generalmente un archivo) y lo convierte en un tipo de dato que cuenta con el/los campos que maneja el objeto Json

Funciones para obtener información sobre un archivo o un path

- **FileInfo info = new FileInfo(fileName);** → Console.WriteLine($"Full Name: {info.FullName}{Environment.NewLine}Directory: {info.Directory} {Environment.NewLine}Extension: {info.Extension} {Environment.NewLine}Create Date: {info.CreationTime}");