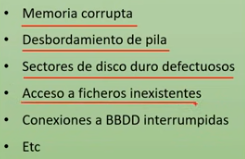
Excepciones

Video 22

Las excepciones son errores en tiempo de ejecución del programa, que escapan al control del programador:



Bloque try….Catch para que el progama sepa qué hacer si aparece una excepción, para que se ejecuten el resto de las líneas.

Try=intenta; catch=captura.

class Program

{

static void Main(string[] args)

{

Random numero = new Random();

int aleatorio = numero.Next(0, 100);

int miNumero;

int intentos = 0;

Console.WriteLine("Introduce un N° entre 0 y 100");

do

{

intentos++;

miNumero = Int32.Parse(Console.ReadLine()); //Si se introduce un texto el programa no será capaz de convertirlo a entero. Ahí da error el programa y se detiene.

if (miNumero > aleatorio) Console.WriteLine("El N° es mas bajo");

if (miNumero < aleatorio) Console.WriteLine("El N° es mas alto");

} while (aleatorio != miNumero);

Console.WriteLine($"¡Correcto! Has necesitado {intentos} intentos" );

}

}

Se mete la línea del error en un try. Si lo consigue no hay problemas, pero si lo hay, se entra en el bloque catch, y que tiene que hacer el programa.

static void Main(string[] args)

{

Random numero = new Random();

int aleatorio = numero.Next(0, 100);

int miNumero;

int intentos = 0;

Console.WriteLine("Introduce un N° entre 0 y 100");

do

{

intentos++;

try

{

miNumero = Int32.Parse(Console.ReadLine()); //Si se introduce un texto el programa no será capaz de convertirlo a entero. Ahí da error el programa y se detiene.

}

catch (FormatException ex)

{

Console.WriteLine("No ha introducido un valor numérico válido. Se toma como número introducido el 0");

miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

}//entre estas llaves se le dice qué debe hacer

if (miNumero > aleatorio) Console.WriteLine("El N° es mas bajo");

if (miNumero < aleatorio) Console.WriteLine("El N° es mas alto");

} while (aleatorio != miNumero);

Console.WriteLine($"¡Correcto! Has necesitado {intentos} intentos" );

Video 23

Qué pasa si introduce un número muy grande, como 5298355394553453. Se genera una excepción, que no se captura.

Se le agrega al catch anterior, otro. En este caso, de tipo OverFlowException. Se usa el mismo try.

try

{

miNumero = Int32.Parse(Console.ReadLine()); //Si se introduce un texto el programa no será capaz de convertirlo a entero. Ahí da error el programa y se detiene.

}

catch (FormatException ex)

{

Console.WriteLine("No ha introducido un valor numérico válido. Se toma como número introducido el 0");

miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

}//entre estas llaves se le dice qué debe hacer

catch (OverflowException ex)

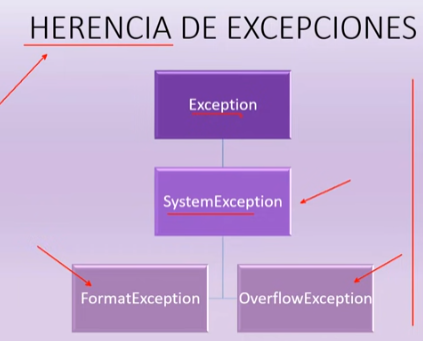
{

Console.WriteLine("Has introducido un valor demasiado alto . Se toma como número introducido el 0");

miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

}

Comenta que podría un programa tener unas 15 excepciones ¿Qué se hace ahí? Poner 15 catchs. Dice que no: herencia de excepciones.



Se pone un solo bloque catch con exception, el padre de todas las clases de excepciones.

try

{

miNumero = Int32.Parse(Console.ReadLine()); //Si se introduce un texto el programa no será capaz de convertirlo a entero. Ahí da error el programa y se detiene.

}

catch (Exception ex)

{

Console.WriteLine("Has introducido un valor demasiado alto . Se toma como número introducido el 0");

miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

}

¿Por qué no se hace siempre así? Porque no le daría mucha info al usuario, de la otra forma, se puede especificar mucho mas.

Se podría poner simplemente catch, quedando implícito (exception ex)

Ex o como se le escriba es un objeto. Es útil para proporcionar información:

Console.WriteLine (e.Message); (esto va dentro del mismo catch)

try

{

miNumero = Int32.Parse(Console.ReadLine()); //Si se introduce un texto el programa no será capaz de convertirlo a entero. Ahí da error el programa y se detiene.

}

catch (Exception ex)

{

Console.WriteLine("Has introducido un valor demasiado alto . Se toma como número introducido el 0");

miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

Console.WriteLine(ex.Message);

}

Video 24

Cuando se usan varios catch pueden surgir conflictos: Se puede usar un exception (genérico) y otro catch, mas específico. El catch genérico debe escribirse después el específico, si no, tira error.

catch (FormatException ex)

{

Console.WriteLine("Has introducido un valor demasiado alto . Se toma como número introducido el 0");

miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

}

catch (Exception ex)

{

Console.WriteLine("Has introducido un valor demasiado alto . Se toma como número introducido el 0");

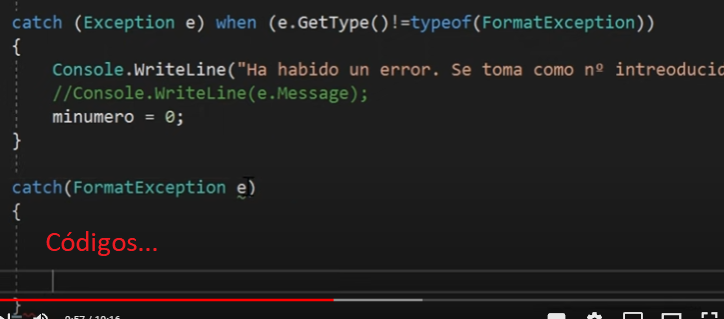
miNumero = 0; //dice que esta linea es una solución poco elegante. Si no hacía esto, el próximo miNumero iba a dar error.

}

Hay otra forma de realizar esta tarea. Se pueden usar excepciones con filtro.

Excepciones con filtro:

Con la clausula when (excepción que se quiere excluir de la captura general)



Expresiones checked y unchecked

Checked:

static void Main(string[] args)

{

int numero = int.MaxValue; //esto devuelve el valor mas alto que admita int.

int resultado = numero + 20; //debería dar error, ya que superaría el macxvalue de int.

Console.WriteLine(resultado); //El programa no cae, pero el resultado es incorrecto: llega hasta el número positivo mas grande y luego va hasta el número negativo mas pequeño

}

checked //esto es para que compruebe lo que viene y si hay error, que se lance la excepción. Que caiga el programa.

{

int numero = int.MaxValue;

int resultado = numero + 20;

Console.WriteLine(resultado);

}

El Visual Studio tiene una opción para chequear todo. Hay que quitar el checked y tocar en explorador de soluciones, ir al proyecto:propiedades:Compilación:avanzadas:”Comprobar el desbordamiento y subdesbordamiento aritmético”, eso por default viene desactivado. Activarlo es como poner el checked.

Video 25

Afirma que hay otra forma de usar el checked: poniendo el checked directamente en la línea que se quiere chequear:

Int resultado=checked (numero +20);

checked

{

int numero = int.MaxValue;

int resultado = checked(numero + 20); //agregué este segundo checked. Sería redundante porque está el primero

Console.WriteLine(resultado);

}

Si se le dice al programa que chequee todo pero se quiere que algo no lo chequee, se le pone unchecked de la misma forma que se pone el checked.

Solo funcionan con int y long, con double y float no.

En el menú ventana está la configuración de excepciones, donde se pueden ver y modificar las excepciones.

Throw

public static string NombreDelMes (int mes)

{

switch (mes)

{

case 1:

return "Enero";

case 2:

return "Febrero";

case 3:

return "Marzo";

case 4:

return "Abril";

case 5:

return "Mayo";

case 6:

return "Junio";

case 7:

return "Julio";

case 8:

return "Agosto";

case 9:

return "Septiembre";

case 10:

return "Octubre";

case 11:

return "Noviembre";

case 12:

return "Diciembre";

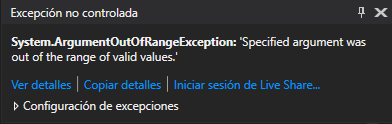
/\*default: // Esta es una de las opciones que se podrían llevar a cabo

return "Dato erróneo"; //WOW. Jamás visto. \*/

default:

throw new ArgumentOutOfRangeException(); //se crea un objeto de este tipo.

}



Esto hace que el programa caiga. Por lo tanto, para que siga funcionando, obliga a usar un try catch.

Console.WriteLine("Introduce el número de mes");

try

{

int numeroMes = Int32.Parse(Console.ReadLine()); //esta línea la puse yo dentro del try, si no, se rompe el programa si uno le pasa un string en vez de un numero

Console.WriteLine(NombreDelMes(numeroMes));//Le pasa por parámetro lo escrito por el usuario.

}

catch (Exception ex)

{

Console.WriteLine("Mensaje de la excepción: " +ex.Message);

}

Console.WriteLine("Aquí se continúa ejecutando el programa");

}

Video 26

Bloque Finally. Nos aseguramos que tal pedazo de código se ejecute siempre, salte o no alguna excepción. Se usa cuando hay que liberar recursos (como para cerrar una base de datos o para cerrar con ficheros externos).

System.IO.StreamReader archivo = null; //se crea un objeto archivo.

try

{

string linea;

int contador = 0;

string path = @"C:\Users\Pcablo\Desktop\Programando\C#\tirar.txt"; //acá está el archivo tirar.txt

archivo = new System.IO.StreamReader(path);

while ((linea=archivo.ReadLine())!=null)// para recorrer lo que haya dentro del archivo. Lo de null es para que en el archivo las lineas tengan algo, no estén vacías. Cuando haya linea vacía sale del bucle.

{

Console.WriteLine(linea);

contador++;

}

}catch (Exception ex)

{

Console.WriteLine("Error con la lectura del archivo.");

}

Esto de acá lee el bloc de notas tirar.txt.

Plantea que si en el try hubiese algo que se quiere si o si que se ejecute, a pesar de que el programa caiga. Podría estar también en el catch o después. Para eso se crea el bloque finally.

Hay que cerrar la lectura del fichero:

finally

{

if (archivo != null) archivo.Close(); //Distinto de nulo, porque si es nulo, no hay nada que cerrar. Incluso daría un error.

Console.WriteLine("Conexión con el fichero cerrada");

}