



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

Control de Excepciones

Buscar errores y manejarlos adecuadamente es un principio fundamental para diseñar software correctamente. En teoría un programador escribe cada línea pretendiendo su correcto funcionamiento, sin embargo existen factores externos, como el uso inadecuado del programa, que generan situaciones especiales interrumpiendo la ejecución de programa o devolviendo datos incorrectos.

Los mecanismos para informar de errores son muy diversos. Algunos métodos están diseñados para devolver un valor booleano que indica el éxito o el fracaso de su ejecución. Otros escriben los errores en un fichero de registro o una base de datos. Sin, embargo el común en cualquier método de detección de errores es dotar al código de mecanismos de detección de errores, procurando cubrir la mayor parte de ellos.

.NET Framework proporciona un mecanismo estándar, llamado control de excepciones estructurado (SEH), para informar de errores. Las excepciones son clases que describen un error. .NET Framework las usa para informar de errores y podemos utilizarlas en nuestro código. Puede escribir código que busque excepciones generadas por cualquier fragmento de código, tanto si procede del CLR como si procede de nuestro propio código y podemos ocuparnos de la excepción generada adecuadamente.

El proceso de detección y gestión de excepciones en el código de C# es sencillo. Se deben identificar tres bloques de código cuando se trabaja con excepciones:

El bloque de código que debe usar el procesamiento de excepciones.

Un bloque de código que se ejecuta si se encuentra una excepción mientras se procesa el primer bloque de código.

Un bloque de código opcional que se ejecuta después de que se procese la excepción.

En C# la generación de una excepción recibe el nombre de **iniciación de una excepción**; el proceso de informar de una excepción ocurrida **capturar** y el fragmento que se ejecuta después de procesar la excepción **finally**.

Cómo especificar el procesamiento de excepciones

La palabra clave de C# try especifica que hay un bloque de código que debe buscar cualquier excepción iniciada mientras se está ejecutando el código.

Sintaxis

```
Try
{
    //Instrucciones donde se deberá informar si existe una excepción
}
catch
{
    //Instrucciones que se deben ejecutar cuando se atrape una excepción
}
```

Las instrucciones del bloque catch se ejecutan si se inicia una excepción desde el bloque try. Si ninguna de las instrucciones del bloque try inicia una excepción, entonces no se ejecuta el código del bloque catch.

Por medio del bloque catch se pueden controlar clases específicas de excepciones. La forma de declararlas es:

1. La palabra catch
2. Un paréntesis de apertura
3. La clase de excepción que desea controlar
4. Un identificador de variable para la excepción
5. Un paréntesis de cierre



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

6. Una llave de apertura
7. Las instrucciones que deben ejecutarse cuando se inicia una excepción del tipo especificado desde el anterior bloque try
8. Una llave de cierre

Ejemplo:

```
Try
{
    //Código que se va a validar
}
catch(Exception thrownException)
{
    //Instrucciones que se ejecutarán si ocurre esta excepción
}
```

El código del bloque try puede iniciar diferentes clases de excepciones y queremos controlar cada una de ella. Por ejemplo:

```
Try
{
    //Instrucciones que se validarán
}
catch(Exception ThrownException1)
{
    //Bloque 1
}
catch(Exception ThrownException2)
{
    //Bloque 2
}
```

También se puede añadir un bloque catch genérico a la lista de bloques de código catch. Como en el siguiente ejemplo:

```
Try
{
    //Código que se debe validar
}
catch(Exception ThrownException)
{
    //Bloque 1
}
catch
{
    //Bloque 2
}
```

Liberar recursos después de una excepción

Los bloques catch pueden ir seguidos por bloque de código. Este bloque de código se ejecuta después de que se procese una excepción y cuando no ocurre ninguna excepción. Si se quiere ejecutar este código, se puede escribir un bloque finally. La palabra clave de C# finally especifica que hay un bloque de código que debe ejecutarse después de que se ejecute un bloque de código try.

Ejemplo:



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

```
Try
{
    //abrir archivo
    //leer archivo
}
catch
{
    //atrapar excepciones
}
finally
{
    //Cerrar archivos
}
```

Se puede escribir un bloque finally inmediatamente después de un bloque try.

Lanzamiento de excepciones propias

Es posible utilizar la instrucción **throw** para lanzar excepciones propias, como se ve en el siguiente ejemplo:

```
namespace control_excepciones
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                int minuto;
                //Solicita los minutos
                Console.WriteLine("Escribe el numero de minutos");
                minuto = Convert.ToInt32(Console.ReadLine());
                if (minuto < 1 || minuto >= 60)
                {
                    string fallo = minuto + " no es un minuto válido";
                    throw new ApplicationException(fallo);
                    // !!No alcanzado!!
                }
            }
            catch (ApplicationException mensaje_error)
            {
                //Mostrar mensaje_error
                Console.WriteLine(mensaje_error);
            }
        }
    }
}
```

En este ejemplo se emplea la instrucción **throw** para lanzar una excepción definida por el usuario, `InvalidTimeException`, si el tiempo analizado no es válido.



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

Clase Exception

Todas las excepciones iniciadas por .NET framework son clases derivadas de la clase System.Exception

Excepciones definidas por .NET Framework

.NET framework define varias excepciones que pueden iniciarse cuando se encuentran ciertos errores en el código de C# o en los métodos que se pueden invocar.

A continuación se listan algunas de estas excepciones:

Tipo de la excepción	Causa de que se produzca la excepción
ArgumentException	Pasado argumento no válido (base de excepciones de argumentos)
ArgumentNullException	Pasado argumento nulo
ArgumentOutOfRangeException	Pasado argumento fuera de rango
ArrayTypeMismatchException	Asignación a tabla de elemento que no es de su tipo
COMException	Excepción de objeto COM
DivideByZeroException	División por cero
FormatException	Cuando se trata de aplicar una conversión no permitida a un tipo de dato
IndexOutOfRangeException	Índice de acceso a elemento de tabla fuera del rango válido (menor que cero o mayor que el tamaño de la tabla)
InvalidCastException	Conversión explícita entre tipos no válida
InvalidOperationException	Operación inválida en estado actual del objeto
InteropException	Base de excepciones producidas en comunicación con código inseguro
NullReferenceException	Acceso a miembro de objeto que vale null
OverflowException	Desbordamiento dentro de contexto donde se ha de comprobar los desbordamientos (expresión constante, instrucción checked, operación checked u opción del compilador /checked)
OutOfMemoryException	Falta de memoria para crear un objeto con new
SEHException	Excepción SHE del API Win32
StackOverflowException	Desbordamiento de la pila, generalmente debido a un excesivo número de llamadas recurrentes.
TypeInitializationException	Ha ocurrido alguna excepción al inicializar los campos estáticos o el constructor estático de un tipo. En InnerException se indica cuál es.



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

Ejercicios de la Práctica

A través de los siguientes ejercicios el alumno aprenderá a utilizar los bloques de excepciones para controlar el correcto funcionamiento del programa.

1. Escribe el siguiente código, deberás compilarlo, ejecutarlo y anotar tus observaciones.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace control_errores
{
    class Program
    {
        static void Main(string[] args)
        {
            int i;
            //Básico
            Console.WriteLine("\n/*****CONTROL DE EXCEPCIONES BASICO*****/");
            try
            {
                //Para probar esta excepcion ingrese letras en lugar de numeros
                Console.WriteLine("Escribe un número:");
                i = Convert.ToInt32(Console.ReadLine());
            }
            catch
            {
                Console.WriteLine("No es un número");
            }
            /*****/
            Console.WriteLine("\n/*****CONTROL DE EXCEPCIONES CON MENSAJE SIMPLE*****/");
            try
            {
                Console.WriteLine("Escribe un número:");
                i = Convert.ToInt32(Console.ReadLine());
            }
            catch(Exception mensaje_error)
            {
                Console.WriteLine(mensaje_error.Message);
            }
            /*****/
            Console.WriteLine("\n/*****MENSAJE ERROR COMPLETO*****/");
            try
            {
                Console.WriteLine("Escribe un número:");
                i = Convert.ToInt32(Console.ReadLine());
            }
            catch (FormatException mensaje_error)
            {
                Console.WriteLine(mensaje_error);
            }
            catch (Exception mensaje_error)
            {
                Console.WriteLine(mensaje_error);
            }
            /*****/
            Console.WriteLine("\n/*****CONTROL DE VARIOS ERRORES*****/");
            try
            {
                int[] A = new int[5];
                for (int j = 0; j <= 5; j++)
```



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

```
{
  Console.WriteLine("Elemento de la matriz A[{0}]: ", j);
  A[j] = Convert.ToInt32(Console.ReadLine());
}
}
catch (FormatException)
{
  Console.WriteLine("Eso no es un número");
}
catch (IndexOutOfRangeException)
{
  Console.WriteLine("Te pasaste de elementos"); //Mensaje personalizado
}
catch (Exception mensaje_error)
{
  Console.WriteLine(mensaje_error.Message);
}
}
}
```



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

2. Escribe un método para cada uno de los problemas y adicione control de errores.

- a) Dada una fecha en formato *aaaammdd* como *parámetro*. Retornar una cadena en formato *dd de cccc de aaaa*. Donde *cccc* es la cadena de caracteres correspondiente al mes *mm*.

Por ejemplo:

La cadena: 20110922

Retornará: 22 de septiembre de 2011

- b) Hacer un método que genere e imprima un cuadrado latino de orden *n* (matriz *n* x *n*). Un cuadrado latino de orden *n* es aquel en el que la primera fila tiene los *n*^o naturales del 1 a *n*, y cada una de las filas restantes de la matriz es igual a su fila superior desplazada una posición hacia la derecha, entrando por la izquierda el número que se pierde en el desplazamiento. Por ejemplo, un cuadrado latino de orden 5 es:

```
1 2 3 4 5
5 1 2 3 4
4 5 1 2 3
3 4 5 1 2
2 3 4 5 1
```

- c) Escribe un método que reciba una cadena y retorne otra cadena transformando las mayúsculas en minúsculas y viceversa. Para cambiar las mayúsculas a minúsculas y viceversa hay que chequear el carácter y ver si es alfabético o no. Una vez que se sabe que es un carácter alfabético, hay que tener en cuenta que el código ASCII correspondiente a la "A" es el 65 y el correspondiente a la "a" es el 97, es decir, la diferencia entre ambos es 32. La misma diferencia se mantiene para otras letras entre la minúscula y la mayúscula.

- d) Diseñar un método que reciba dos matrices cuadradas y retorne la matriz solución que resulta de multiplicar dos matrices cuadradas. La fórmula matemática para esta operación matricial es:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Ejemplo:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \times \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{pmatrix}$$