CAPÍTULO 14

MANEJO DE EXCEPCIONES

OBJETIVOS

- Asimilar el concepto de excepción.
- Comprender los problemas generados por la excepciones.
- Aprender a utilizar el bloque TRY – CATCH para tratamiento de las excepciones.
- Analizar el funcionamiento de la estructura TRY – CATCH.
- Diseñar aplicaciones Windows y de consola de capítulos anteriores introduciendo el tratamiento de excepcioenes.

14.1 TRATAMIENTO DE EXCEPCIONES

Una buena aplicación C# debe tener capacidad para enfrentarse a lo inesperado. Por muchas comprobaciones de error que se añadan al código, siempre habrá algo que pueda ir mal. El usuario puede dar una respuesta imprevista a una pregunta, por ejemplo, o tratar de escribir a un archivo en una carpeta que ha sido borrada. Las posibilidades son infinitas.

Si se produce un error de tiempo de ejecución en una aplicación C#, el sistema operativo lanza una excepción. Las excepciones se pueden capturar con una combinación de las instrucciones ***try*** y ***catch***, como se explica más adelante. Si alguna de las instrucciones en la parte *try* de la aplicación hace que se produzca una excepción, la ejecución pasará al bloque *catch*.

14.2 EXCEPCIONES PREDEFINIDAS COMUNES

Hay predefinidas múltiples excepciones que se corresponden con los errores más comunes que pueden surgir durante la ejecución de una aplicación. En la siguiente tabla recogen algunas:

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.
ArrayTypeMistmatchException	Asignación a tabla de elemento que no es de su tipo.
COMException	Excepción de objeto COM.
DivideByZeroException	División por cero.
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*, operanción *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.
TypeInizializationException	Ha ocurrido alguna excepción al inicializar los campos estáticos o el constructor estático de un tipo. En *InnerException* se indica cuál es.

Tabla 14-1 Excepciones predefinidas de uso frecuente

14.3 BLOQUE TRY CATCH

Los bloques ***try - catch*** son la solución que ofrece la orientación a objetos a los problemas de tratamiento de errores. La idea consiste en separar físicamente las instrucciones básicas del programa para el flujo de control normal de las instrucciones para tratamiento de errores. Así, las partes del código que podrían lanzar excepciones se colocan en un bloque *try*, mientras que el código para tratamiento de excepciones se pone en un bloque *catch* aparte.

La sintaxis de un bloque *catch* es la siguiente:

catch (tipo-de-clase identificador) { ... }

El tipo de clase tiene que ser *System.Exception* o un tipo derivado de *System.Exception*. El identificador, que es opcional, es una variable local de sólo lectura en el ámbito del bloque *catch*.

```
catch (Exception capturada) { ... }

Console.WriteLine(capturada); // Error en tiempo de compilación:

// capturada está fuera de ámbito
```

Un ejemplo de utilización:

```
  try {
```

```
        Console.WriteLine("Escriba un número");
        Int numero = int.Parse(Console.ReadLine());
    }
    catch (OverflowExceptioncapturada){
        Console.WriteLine(capturada);
    }
```

14.4 CAPTURA DE EXCEPCIONES. INSTRUCCIÓN TRY

Una vez lanzada una excepción es posible escribir código que es encargue de tratarla. Por defecto, si este código no se escribe la excepción provoca que la aplicación aborte mostrando un mensaje de error en el que se describe la excepción producida (información de su propiedad *Message*) y dónde se ha producido (información de su propiedad *StackTrace*) Así, dado el siguiente código fuente de ejemplo:

```
using System;
class PruebaExcepciones
{
    static void Main()
    {
        int divisor = 0;
        int cociente = 2/divisor;
    }
}
```

Al compilarlo no se detectará ningún error ya que al compilador no calcular el valor de *divisor* en tanto que es una variable, por lo que no detectará que dividir **2/divisor** no es válido. Sin embargo, al ejecutarlo se intentará dividir por cero en esa instrucción y ello provocará que aborte la aplicación mostrando el siguiente mensaje:

```
Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
  at PruebaExcepciones.Main()
```

Como se ve, en este mensaje se indica que no se ha tratado una excepción de división por cero (tipo ***DivideByZeroException***) dentro del código del método *Main*() de la clase *PruebaExcepciones*.

Si se desea tratar la excepción hay que encerrar la división dentro de una instrucción ***try*** con la siguiente sintaxis:

```
try
        <instrucciones>
catch (<excepción1>)
        <tratamiento1>
catch (<excepción2>)
        <tratamiento2>
```

De una forma simplificada el significado de *try* es el siguiente:

1º. Si durante la ejecución de las <instrucciones> se lanza una excepción de tipo <excepción1> (o alguna subclase suya) se ejecutan las instrucciones <tratamiento1>, si fuese de tipo <excepción2> se ejecutaría <tratamiento2>, y así hasta que se encuentre una cláusula *catch* que pueda tratar la excepción producida.

2º. Si no se encuentra ningún *catch* compatible, entonces se buscaría en el código desde el que se llamó al método que produjo la excepción. Si tampoco allí se encuentra un tratamiento apropiado se aborta dicho programa y se muestra el mensaje de error con información sobre la excepción lanzada ya visto.

Así, para tratar la excepción del ejemplo anterior de modo que una división por cero provoque una reacción, se podría añadir un catch de esta otra forma:

```
catch (DivideByZeroException)
    { d=0; }
}
```

14.5 (*)DEPURACIÓN JUST - IN-TIME

Si no se utiliza tratamiento de excepciones se producirá una excepción en tiempo de ejecución. Si en lugar de ello se prefiere depurar un programa utilizando depuración ***Just-in-Time***, es preciso activarla antes. Una vez activada, depuración *Just-in-Time* indicará el depurador que hay que utilizar dependiendo del entorno y de las herramientas instaladas.

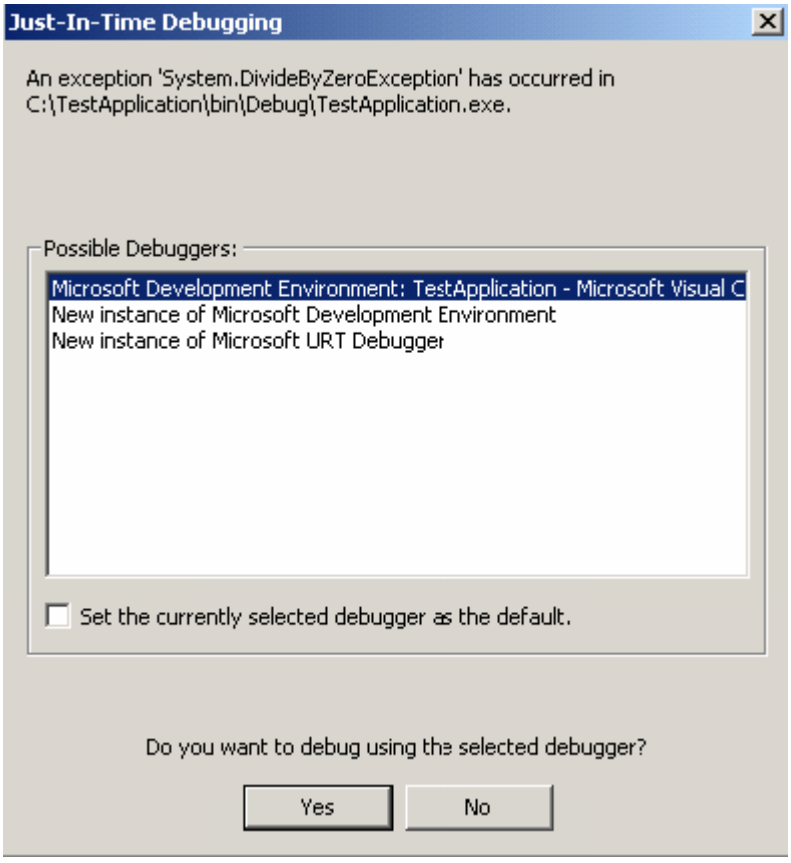

Figura 14-1 Depuración Just In Line

Ejecute los siguientes pasos para activar la depuración Just-in-Time:

1. En el menú **Tools** (Herramientas), pulse **Options** (Opciones).

2. En el cuadro de diálogo **Options**, haga clic en la opción **Debugging** (Depuración).
3. En la carpeta **Debugging**, pulse **Just-In-Time Debugging** (Depuración Just-in-Time).
4. Active o desactive la depuración **Just-in-Time** (JIT) para distintos tipos de programas y pulse **OK**.

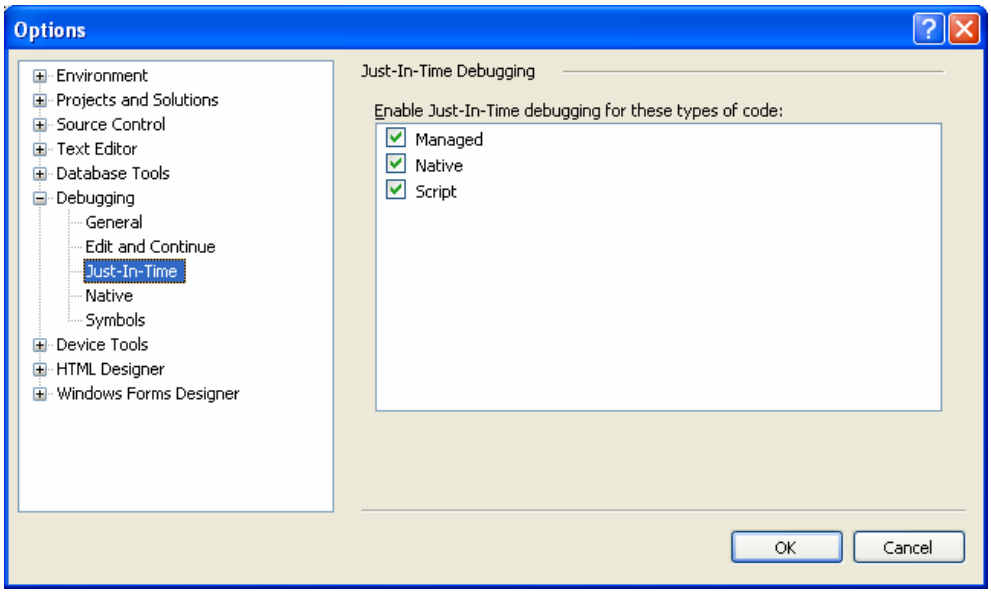

Figura 14-2 Habilitar la depuración Just In Line

14.6 AÑADIR TRATAMIENTO DE EXCEPCIONES A UN PROGRAMA C#

En este ejemplo escribirá un programa que utiliza tratamiento de excepciones para capturar errores inesperados en tiempo de ejecución. El programa pide al usuario dos valores enteros, divide el primero por el segundo y muestra el resultado.

14.6.1 Ejemplo de programa sin tratamiento de excepción

Según lo explicado hasta ahora, el programa sin tratamiento de excepciones podría contener el siguiente código:

Dividir.cs

DividirEnteros_01.Dividir | Main(string[] args)

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

namespace DividirEnteros_01
{
    class Dividir
    {
        static void Main(string[] args)
        {
            string dato;
            int dividendo, divisor, cociente;

            Console.WriteLine("\n División de números ENTEROS:");
            Console.Write("\n Escriba el Dividendo: ");
            dato = Console.ReadLine();
            dividendo = Int32.Parse(dato);
            Console.Write("\n Escriba el Divisor: ");
            dato = Console.ReadLine();
            divisor = Int32.Parse(dato);
            cociente = dividendo / divisor;
            Console.Write("\n El resultado es: {0}", cociente);
            Console.ReadLine();
        }
    }
}
```

Figura 14-3 Programa de división sin tratamiento de excepciones

Si se prueba el funcionamiento de este programa con dos números enteros cualquiera, producirá el efecto de la siguiente figura:

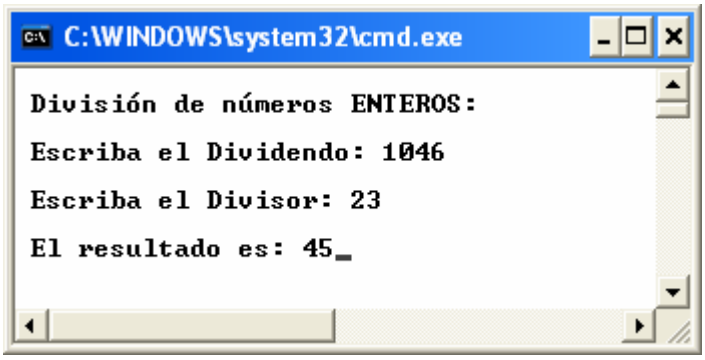

Figura 14-4 Probando el programa de división con dos números enteros

Sin embargo si se prueba el programa con un cero como divisor se provoca el lanzamiento de una excepción (división por cero) y la siguiente ventana:

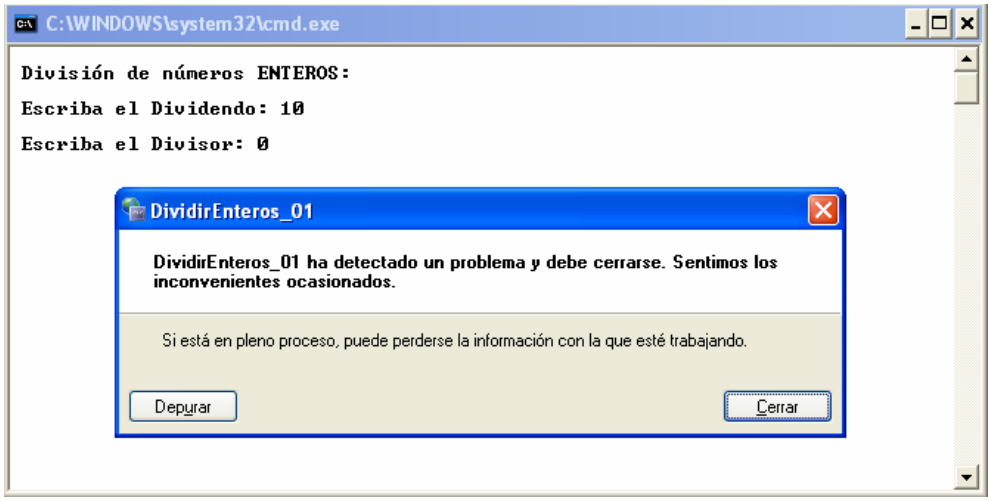

Figura 14-5 Probando el programa de división con un divisor cero

14.6.2 Añadir bloque TRY - CATCH

Si se añade al programa anterior un bloque *try – catch* como se indica a continuación:

```
static void Main(string[] args)
{
     try
     {
      string dato;
      int dividendo, divisor, cociente;

      Console.WriteLine("\n División de números ENTEROS:");
      Console.Write("\n Escriba el Dividendo: ");
      dato = Console.ReadLine();
      dividendo = Int32.Parse(dato);
      Console.Write("\n Escriba el Divisor: ");
      dato = Console.ReadLine();
      divisor = Int32.Parse(dato);
      cociente = dividendo / divisor;
      Console.Write("\n El resultado es: {0}", cociente);
      Console.ReadLine();
     }

     catch (Exception e) // Captura la excepción de cualquiera de las
formas:.
     {
      Console.WriteLine("\n Excepción lanzada: {0}", e);
      Console.WriteLine("\n\n Excepción lanzada: {0}", e.Message);
     Console.WriteLine
         ("\n El divisor es cero." +
        "Vuelve a introducir los datos.");
      Console.ReadLine();
             }
```

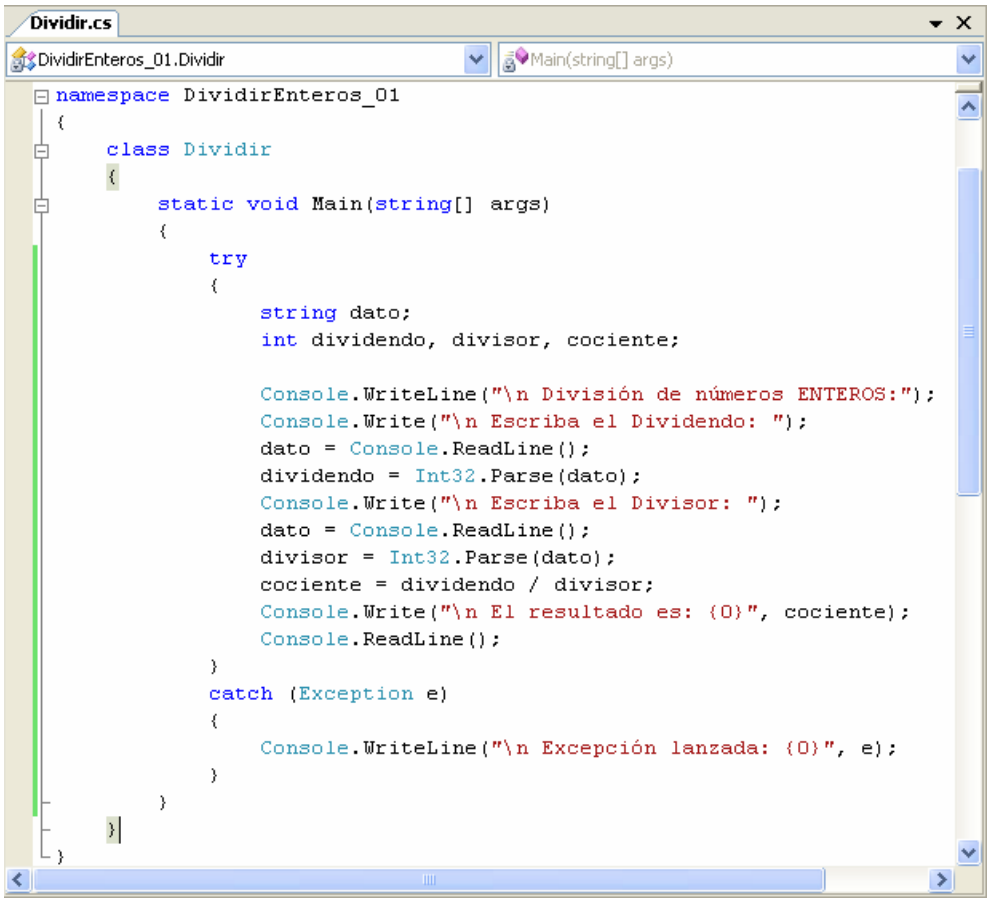

Figura 14-6 Programa de división al que se ha añadido tratamiento de excepción divisón entre cero

Se observa dos diferencias respecto del código anterior:

- El método *Main* se ha situado dentro de un bloque *try*.
- Se ha añadido una instrucción *catch* después del bloque *try*. La instrucción *catch* tiene que imprimir un mensaje.

Si se vuelve a ejecutar el programa con una división entre cero, el programa sigue provocando el lanzamiento de una excepción (división por cero), pero esta vez el error es capturado y aparece su mensaje.

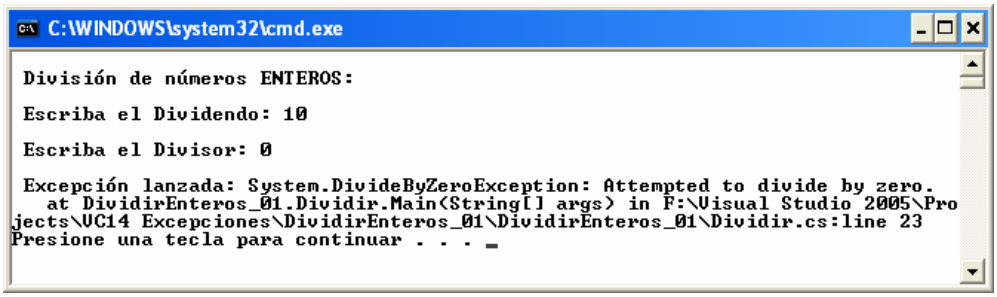

Figura 14-6 Probando el tratamiento de la excepción

Este mismo programa permite capturar excepciones de otro tipo, por ejemplo si se introduce un número demasiado grande para almacenarlo en un tipo *int* (-2147483648 a 2147483647) se obtendría:

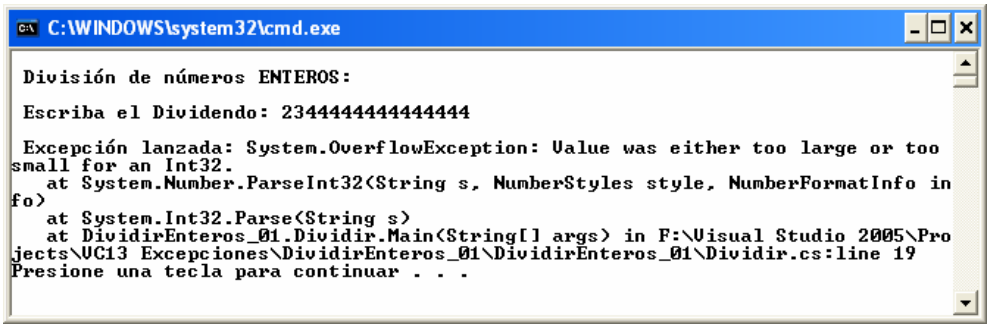

Figura 14-7 Probando el tratamiento de la excepción

14.7 PRÁCTICAS

Ejemplos del capítulo. Practica los ejemplos de este capítulo introduciendo todas las mejoras que estime conveniente, especialmente utilizar siempre que sea posible controles de ventana Windows.

Modificar aplicaciones capítulos anteriores. Volver a diseñar aplicaciones de capítulos anteriores introduciendo el tratamiento de las excepciones.