# C sharp NET/Capítulo 8

## Excepciones

Las excepciones son errores imprevistos que suceden en los programas.

La mayor parte del tiempo el programador puede, y debe, detectar y controlar los errores en el código del programa. Por ejemplo:

- la validación de entrada del usuario
- la comprobación de los objetos nulos
- la verificación de otros valores
- ...

Todos estos son ejemplos de errores que el programador debe controlar y que son muy habituales.

Sin embargo, hay ocasiones en las que no se puede saber si se producirá un error. Por ejemplo, no se puede predecir cuándo recibirá un archivo de error de E / S, cuándo se queda sin memoria el sistema, o cuándo puede haber un error de base de datos. Estas cosas son por lo general poco probables, pero podrían suceder. Aún así se debe ser capaz de tratar con ellos cuando se producen. Aquí es donde entra en juego el **manejo de excepciones**.

### Identificación de excepciones

Cuando se producen excepciones, se usa el término "thrown". Lo que en realidad se produce es un objeto que se deriva de la clase System.Exception.

La clase System.Exception proporciona varios métodos y propiedades para obtener información sobre lo que salió mal. Por ejemplo, la propiedad Message proporciona información resumida acerca de lo que el error era, la propiedad StackTrace proporciona información de la pila de donde se produjo el problema y el método ToString () se reemplaza para revelar una descripción detallada de la excepción entero.

La identificación de las excepciones que se necesitan para manejar depende de la rutina que se esté escribiendo. Por ejemplo, si la rutina abre un archivo con el System.IO.File.OpenRead (),  podría pasar alguna de las siguientes excepciones:

- *ArgumentException*
- *ArgumentNullException*
- *PathTooLongException*
- *DirectoryNotFoundException*
- *UnauthorizedAccessException*

- *FileNotFoundException*
- *NotSupportedException*
- *SecurityException*

Es fácil averiguar qué excepciones son las adecuadas buscando documentación en la que se haga referencia al Namespace usado (System.IO en nuestro caso) y mirando las clases asociadas a esta biblioteca. Entre ellos encontraremos los referidos a las excepciones.

### Bloques Try/catch

Cuando se producen excepciones, el programador tiene que ser capaz de manejarlas. Esto se realiza mediante la aplicación de un bloque try / catch.

El código que podría lanzar una excepción se pone en el bloque try y el código de manejo de excepciones va en el bloque catch.

Ejemplo 8.1

```
using System;
using System.IO;

try
 {
 File.OpenRead("NonExistentFile");
}
catch(Exception ex)
{
Console.WriteLine(ex.ToString());
}
```

Aunque el código del ejemplo 8.1 sólo contenga un catch, pueden haber tantos como se quiera o como errores se puedan producir. Serán colocados, eso sí, en un orden, del más específico (el primero) al más general (el último). A continuación pondremos el ejemplo que intentaría atrapar dos posibles excepciones con dos catch del try del ejemplo 8.1 anterior.

Ejemplo 8.2

```
catch(FileNotFoundException fnfex)
{
Console.WriteLine(fnfex.ToString());
 }
catch(Exception ex)
{
 Console.WriteLine(ex.ToString());
}
```

Si el archivo no existiera, la excepción FileNotFoundException sería lanzadada y atrapada por el primer bloque catch. Sin embargo, si se produjera una excepción PathTooLongException, el segundo catch capturaría la excepción. Esto se debe a que no hay un bloque catch para la excepción PathTooLongException y el tipo genérico de excepciones bloque catch es la única opción disponible para detectarla.

## Bloques Finally

Una excepción puede dejar el programa en un estado incoherente por no liberar recursos o hacer algún otro tipo de limpieza. Un bloque catch es un buen lugar para averiguar lo que pudo haber salido mal y tratar de recuperar el programa. Sin embargo, a veces es necesario llevar a cabo acciones de limpieza si su programa tiene éxito. Estas situaciones son buenos candidatos para el uso de un bloque finally.

```
FileStream outStream = null;
FileStream inStream = null;

try
{
outStream = File.OpenWrite("DestinationFile.txt");
inStream = File.OpenRead("BogusInputFile.txt");
}
catch(Exception ex)
{
Console.WriteLine(ex.ToString());
}
finally
{
      if (outStream != null)
      {
            outStream.Close();
            Console.WriteLine("outStream closed.");
      }
      if (inStream != null)
      {
            inStream.Close();
            Console.WriteLine("inStream closed.");
      }
}
```

Hay que tener en cuenta que el bloque finally no es necesario.

Es cierto que, en circunstancias normales, si se detecta la excepción, todo el código a raíz de la captura se ejecutará. Sin embargo, try / catch / finally es para circunstancias excepcionales y es mejor planear para el peor de los casos, y así poder hacer el programa más robusto.