INVESTIGACIÓN SOBRE EXCEPCIONES DE VISUAL BASIC .NET

Durante la codificación, los errores inesperados surgen con la frecuencia suficiente, lo que requiere depuración y pruebas. Pero a veces los errores son realmente esperados y, para evitarlo, está el bloque **Try..Catch..Throw..Finally..End Try .**

Para administrar un error correctamente, el código se coloca en un bloque **Try..Catch** , por lo que el Catch , como su nombre indica, detectará todas las excepciones que surjan en este bloque.

Y en caso de excepción, tenemos la posibilidad de **Throw** el error, es decir, devolverlo para notificarlo al usuario o administrarlo internamente en el propio código.

El **Finally** parte es el código final que, cualquiera que sea el resultado, si hay una excepción o no, el código se ejecutará antes de salir del bloque.

En caso de que tengamos que salir del reloj, existe la instrucción **Exit Try** que se puede usar. Pero aquí también, el código en la sección F**inally** se ejecutará antes de terminar.

La sintaxis es simple:

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

donde solo es obligatorio el **Try** y el **End Try**. El resto se puede ignorar, pero como buena práctica, incluya la parte **Finally**, incluso si se deja en blanco.

Llegando a la excepción, hay diferentes tipos de excepción que pueden ser capturados. Están listas las excepciones disponibles desde .Net Framework, como se muestra a continuación;

Clase de excepción	Breve descripción
System.IO.IOException	Maneja errores de E / S
System.IndexOutOfRangeException	Se refiere a un índice de matriz fuera de rango
System.ArrayTypeMismatchException	Cuando el tipo no coincide con el tipo de matriz
System.NullReferenceException	Maneja los errores generados al hacer referencia a un objeto nulo.
System.DivideByZeroException	Maneja los errores generados al dividir un dividendo con cero.
System.InvalidCastException	Maneja los errores generados durante el encasillado.
System.OutOfMemoryException	Maneja los errores generados desde la memoria libre insuficiente.
System.StackOverflowException	Maneja los errores generados por el desbordamiento de pila.
----------	----------

En **Visual Basic**, los errores se clasifican en una de tres categorías

1. Errores de sintaxis

Los errores de sintaxis son aquellos que aparecen mientras escribe código. Si está utilizando Visual Studio, Visual Basic verifica su código a medida que lo escribe en la ventana del Editor de código y le advierte si comete un error, como escribir mal una palabra o usar un elemento de lenguaje de manera incorrecta. Si compila desde la línea de comandos, Visual Basic muestra un error de compilador con información sobre el error de sintaxis. Los errores de sintaxis son el tipo de error más

común. Puede solucionarlos fácilmente en el entorno de codificación tan pronto como ocurran.

2. Errores en tiempo de ejecución

Los errores en tiempo de ejecución son aquellos que aparecen solo después de compilar y ejecutar su código. Se trata de un código que puede parecer correcto en el sentido de que no tiene errores de sintaxis, pero que no se ejecutará. Por ejemplo, puede escribir correctamente una línea de código para abrir un archivo. Pero si el archivo no existe, la aplicación no puede abrir el archivo y lanza una excepción. Puede corregir la mayoría de los errores en tiempo de ejecución reescribiendo el código defectuoso o utilizando el manejo de excepciones , y luego recompilando y volviendo a ejecutarlo.

3. Errores lógicos

Los errores lógicos son aquellos que aparecen una vez que la aplicación está en uso. La mayoría de las veces son suposiciones erróneas realizadas por el desarrollador o resultados no deseados o inesperados en respuesta a las acciones del usuario. Por ejemplo, una clave mal escrita puede proporcionar información incorrecta a un método, o puede asumir que siempre se proporciona un valor válido a un método cuando ese no es el caso. Aunque los errores lógicos se pueden manejar mediante el manejo de excepciones (por ejemplo, probando si un argumento lo es Nothingy lanzando una ArgumentNullException), lo más común es que se aborden corrigiendo el error en la lógica y volviendo a compilar la aplicación.