Curso de Programación .NET nivel I

Unidad 8

“Debug”

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Índice

Contenido

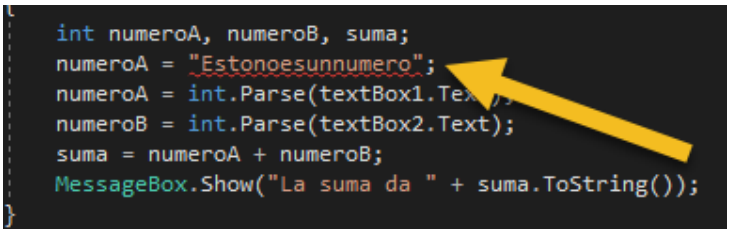
Introducción

Si bien con todo lo que hemos aprendido ya estamos listos para realizar aplicaciones en .NET tanto de tipo consola como de tipo visual de manera profesional, hay un tema fundamental que no cubrimos y lo vamos a hacer en esta unidad: El manejo y la depuración de errores. Por más que hagamos mucho esfuerzo por disminuir los errores siempre vamos a tener que lidiar con ellos por lo tanto es importante que un desarrollador conozca los mecanismos para evitarlos y si lamentablemente ocurren como tratarlos.

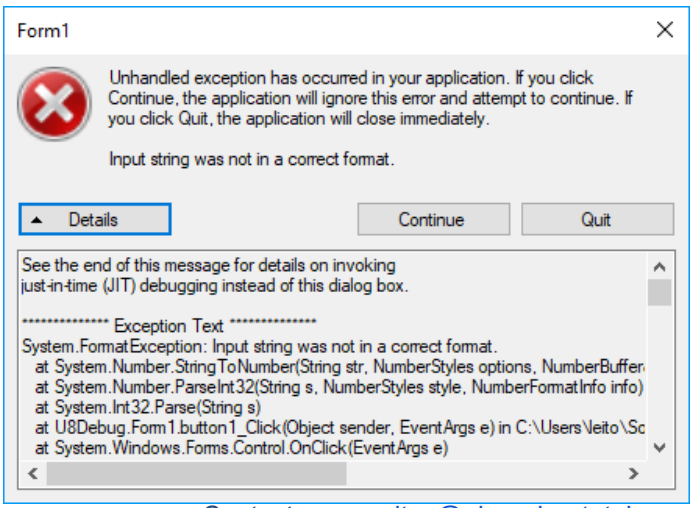
Tipos de Errores

Podemos diferenciar dos tipos principales de errores en el desarrollo de las aplicaciones:

- **Errores de compilación:** los errores de compilación como su nombre lo sugiere son los que se presentan en el momento de querer compilar el código, en ese momento el compilador es capaz de detectarlos y nos indica generalmente como corregirlos correctamente. Al no permitir compilar la aplicación, estos errores nunca van a llegar a impactar al usuario.



Errores de runtime: Los errores de runtime son más complejos ya que son errores que suceden en el momento de ejecutar la aplicación, son los que errores con que hemos lidiado toda la vida como usuarios de aplicaciones, los famosos carteles de errores de las aplicaciones que interrumpen al usuario y dan una experiencia desagradable.



Los errores de compilación son tratados con los mensajes que el compilador nos presenta ene tiempo de compilación así podemos corregirlos e intentar otra vez el proceso. Los errores de runtime en cambio son tratados por medio de una herramienta llamada **Debug o Debugger** (depurador de aplicaciones según el español).

Además del Debug también podemos dejar que la aplicación trate las excepciones por medio de código y para eso existe la estructura try / catch.

Ambas técnicas son las que veremos en este capitulo.

Try Catch

Introducción

Cuando hay un error de runtime en nuestra aplicación un objeto de tipo Exception es creado y quien se encarga de gestionarlo es la estructura de código llamada Try Catch.

El bloque de Try Catch se utiliza para manejar excepciones en nuestro código. Esta estructura denota dos partes fundamentales:

- **Try** (intentar): Esta parte de la estructura va a contener el código que estoy interesado en ejecutar que va a estar custodiado en caso de que exista alguna excepción.

- **Catch** (atrapar): En caso que el código ejecutado en la parte del Try sufra alguna excepción, este otro código va a "atrapar" la excepción y realizar todas las acciones de corrección o alerta necesarias.

Iconos hechos por those-icons

Sin la existencia del Try Catch y ante la eventual aparición de un error nuestra aplicación se interrumpiría forzando a nuestros usuarios a volver a ejecutarla. Con esta estructura puedo evitar esa interrupción.

Implementación

Implementar Try Catch es muy simple, veamos el siguiente código:

```
int numeroA, numeroB, suma;
numeroA = int.Parse(textBox1.Text);
numeroB = int.Parse(textBox2.Text);
suma = numeroA + numeroB;
MessageBox.Show("La suma da " + suma.ToString());
```

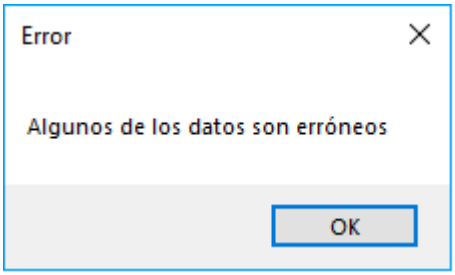
Este código lo que hace es recibir de dos Textbox de con números, los convierte a int y los suma.
Ahora que sucede si en vez de ingresar un número ingresamos una letra, bueno efectivamente al intentar transformar el contenido del TextBox de tipo String en un int va a fallar lo que va a generar una Excepción y por lo tanto va a tirar abajo mi aplicación.

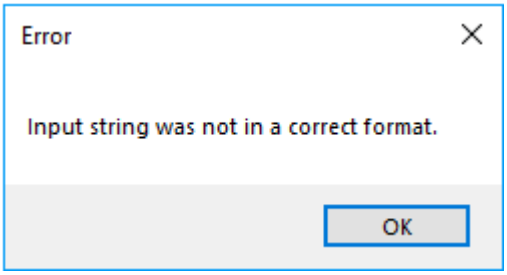
Quiero evitar ese problema y además quiero informar al usuario que hubo un error con sus datos, veamos como lo hacemos con Try Catch:

```
try
{
    int numeroA, numeroB, suma;
    numeroA = int.Parse(textBox1.Text);
    numeroB = int.Parse(textBox2.Text);
    suma = numeroA + numeroB;
    MessageBox.Show("La suma da " + suma.ToString());
}
catch (Exception ex)
{
    MessageBox.Show("Algunos de los datos son erróneos", "Error");
    MessageBox.Show(ex.Message, "Error");
}
```

Recordemos que dentro de **Try** va a ir el código que quiero vigilar, en este caso todo mi código original.

Luego en el **Catch** el código que va a decidir que acción tomar luego que se atrape a la excepción, en mi caso quiero mostrar un cartel al usuario junto con otro donde doy mas información para que se la pueda reportar a su desarrollador y así poder mejorar la aplicación.





Finally

Tenemos además del Try y del Catch otra sub-estructura que se puede agregar llamada **Finally** o finalmente. Para explicar el uso del Finally veamos primero cual es el flujo de mi aplicación al suceder una Excepción:

1. La excepción sucede en la linea 2
2. El flujo de la aplicación se pasa a la primer linea del Catch y se ejecutan los dos MessageBox de error.
3. La ejecución del Finally sucede que simplemente vacia los texboxs.

¿Qué hubiera sucedido si el código que tengo en finally lo hubiera puesto debajo de la línea 2? Nunca se hubiera ejecutado luego de la Excepción ya que cuando sucede una, el programa saltea todo lo que sigue en el bloque **Try** y se va directo al **Catch**. **Finally** aparece al rescate para asegurarme que el código contenido en él si o si va a ejecutarse.

Finally generalmente tiene código para cerrar archivos o conexiones creadas anteriormente que si sucediera una Excepción no me queden abiertas.

Debug

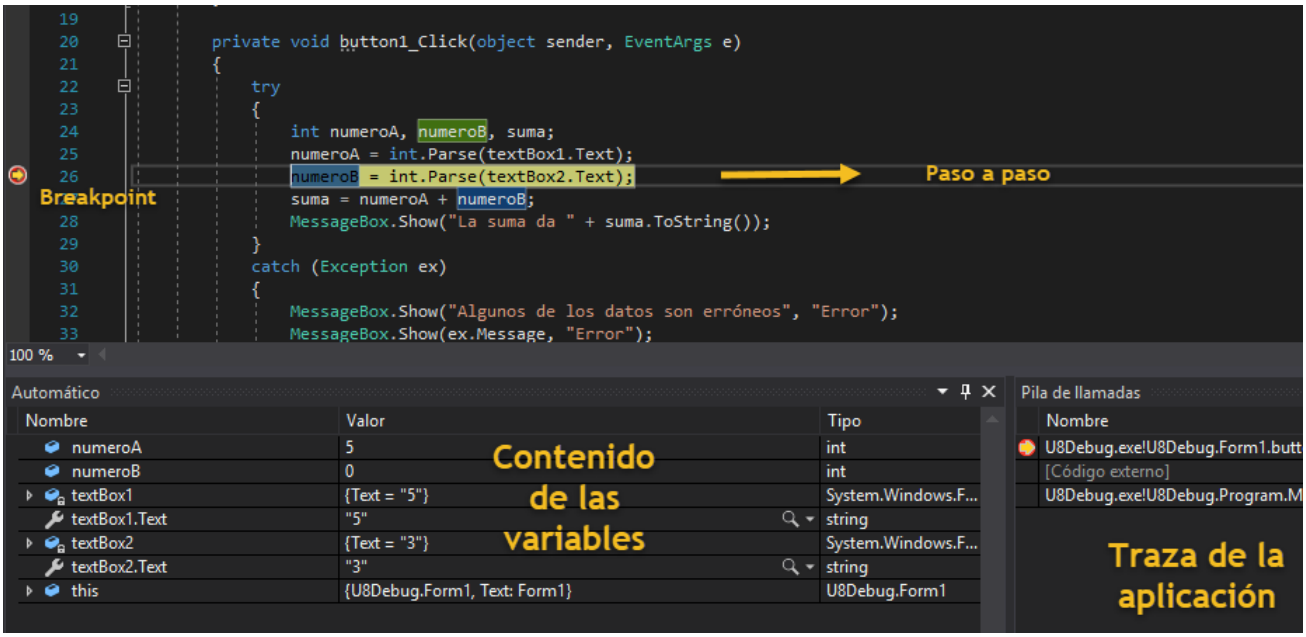
Explicación

Ya hemos visto que tan fácil es perdernos en la ejecución de un programa simple, imaginen algo mas complejo con miles de objetos y referencias por acá y por allá, algo muy difícil de seguir. Por suerte tenemos una herramienta que nos ayuda en esta tarea y es el **Debug**.

El **Debug** es una herramienta o más bien un concepto muy útil a la hora de investigar que esta sucediendo con mi aplicación cuando se ejecuta, me permite **seguir** sus pasos.

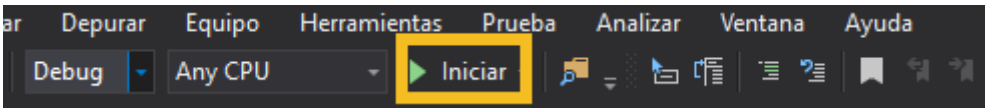
En el momento que ejecuto una aplicación con el debugger sobre ella (o en modo debug), el debugger me permite varias maneras de ver que está haciendo mi código cuando corre. Podés ir paso a paso por instrucción viendo los contenidos de las variables como así también monitorearlas y ver como van cambiando a lo largo de la ejecución y por supuesto ver el trazado que realizar la aplicación a lo largo de su vida.

El **Debug** es una ayuda indispensable cuando mi aplicación falla y no se que está sucediendo.



Uso del Debug

Para iniciar el modo debug simplemente debemos hacer click en el símbolo de Play en el menú superior.

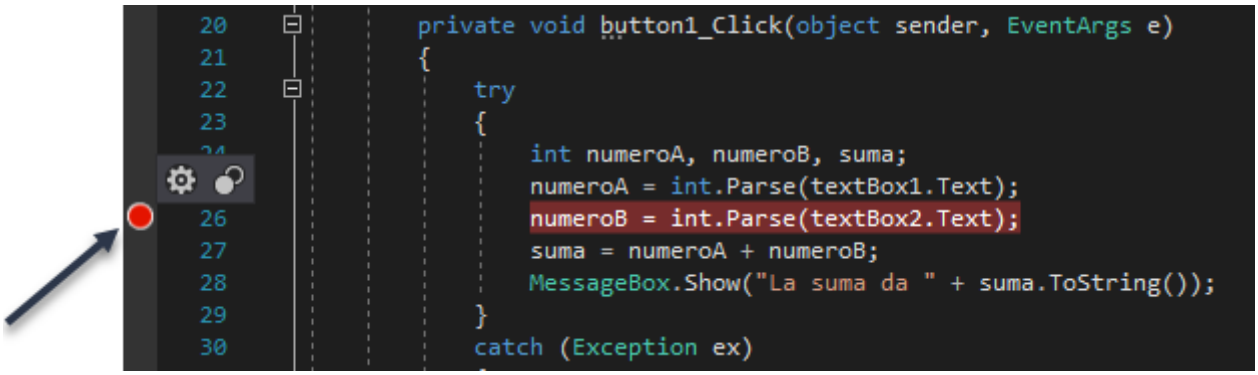


Al realizar esa acción, mi aplicación va a ser compilada y el modo Debug se va a ejecutar sobre ella. Si no hemos aplicado ningún punto de parada o **Breakpoint** la ejecución va a ser similar a estar siendo ejecutada sin Debug.

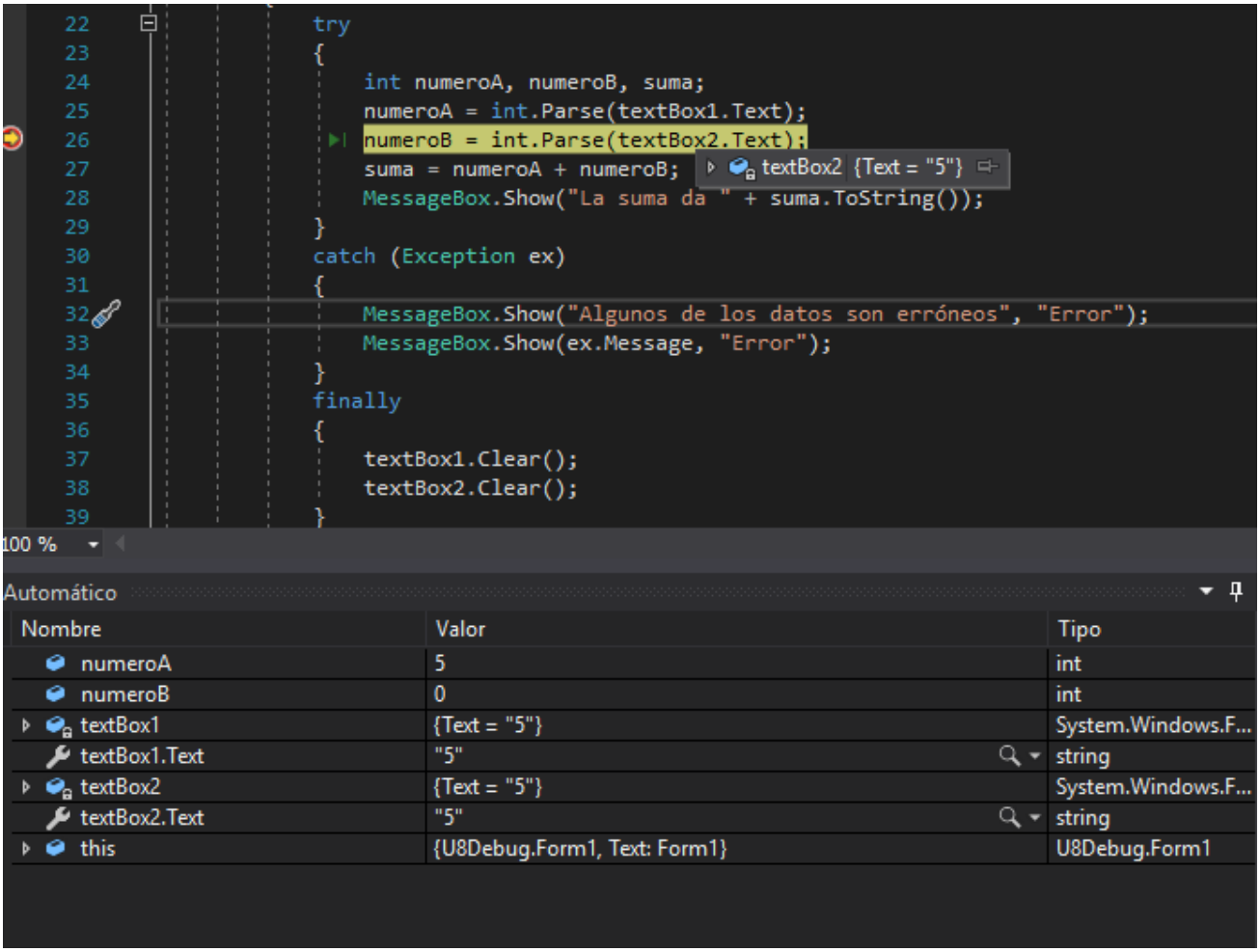
Cada línea de nuestro código puede seleccionarse como linea de Breakpoint; Al seleccionar una, el proceso de ejecución de la aplicación se va a detener permitiéndome investigar que esta sucediendo en ese momento.
Para trazar una analogía imaginen pausar un video en un momento dado para ver mejor algunos detalles que me estoy perdiendo. El Breakpoint es definir en que hora, minuto y segundo ese video va a ser pausado, solamente que en este caso no hablamos de tiempo sino de lineas de código.

Definir un Breakpoint

Para definir un Breakpoint basta ir a la linea de código de mi interés y hacer click en ese espacio al lado del número hasta que aparezca esa pelotita roja junto a que mi código sea resaltado en rojo así:

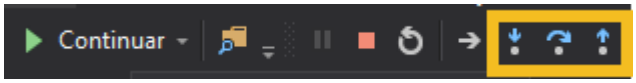


Luego al ejecutar el código, por medio del botón Iniciar, una vez que la ejecución llegue a ese punto voy a poder ver el contenido de las variables como también todas sus propiedades :


try
{
int numeroA, numeroB, suma;
numeroA = int.Parse(textBox1.Text);
numeroB = int.Parse(textBox2.Text);
suma = numeroA + numeroB;
textBox2 {Text = "5"}
MessageBox.Show("La suma da " + suma.ToString());
}
catch (Exception ex)
{
MessageBox.Show("Algunos de los datos son erróneos", "Error");
MessageBox.Show(ex.Message, "Error");
}
finally
{
textBox1.Clear();
textBox2.Clear();
}
100 %
Automático
Nombre
Valor
Tipo
numeroA
5
int
numeroB
0
int
textBox1
{Text = "5"}
System.Windows.F...
textBox1.Text
"5"
string
textBox2
{Text = "5"}
System.Windows.F...
textBox2.Text
"5"
string
this
{U8Debug.Form1, Text: Form1}
U8Debug.Form1

Ejecución paso a paso

Para reanudar la ejecución tengo varias opciones:



1. **Continuar**: Simplemente va a ejecutar todas las lineas hasta el próximo breakpoint o fin de ejecución
2. **Paso a Paso Por instrucciones:** Esta es las primeras de las flechas encerradas en amarillo del gráfico y lo que va a hacer es ejecutar la línea en que me encuentro y parar en la próxima linea. Es como si pusiera un breakpoint en la linea de abajo. Si hay una función va a entrar a ella y seguir su ejecución dentro.
3. **Paso a Paso Por procedimiento:** Mismo concepto que la anterior solamente que si se encuentra con una función no va a entrar pero si la va a ejecutar.
4. **Paso a Paso para Salir:** En esta última mi programa se va a ejecutar hasta que encuentre un corchete de cerrar la función **}** en ese momento, que tiene que volver a la función que fue llamada, se va a producir un breakpoint.

El **Debug** es una herramienta de uso fundamental para todo desarrollador en el proceso de troubleshooting o depuración de errores como también lo es para **aprender** como funciona cierto programa que no fue desarrollado por mí o que no recuerdo como lo había programado; Me ayuda a entender la lógica y verla en funcionamiento en el sistema o aplicación.