

Programación Orientada a Objetos en C#

Unidad 5.- Excepciones

Autor:
Dr. Ramón Roque Hernández
<http://ramonroque.com/Materias/pooTec.htm>
ramonroque@yahoo.com

Colaboradores:
Ing. Gloria Ma. Rodríguez Morales
grodriguez@itnuevolaredo.edu.mx

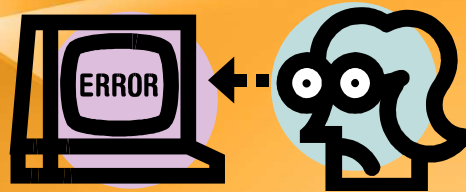
Ing. Bruno López Takeyas, M.C.
www.itnuevolaredo.edu.mx/takeyas
takeyas@itnuevolaredo.edu.mx

Introducción al manejo de excepciones

- Excepción
- Manejador de excepción
- Levantamiento de una excepción

Esperando lo inesperado !!

- Ocurrencia de sucesos que se consideran excepcionales.
- Cómo manejar situaciones anómalas
- Pueden ocurrir durante la ejecución de un programa



Excepciones

- Definición
- Tipos de excepciones
- Propagación de excepciones
- Gestión de excepciones

¿QUÉ ES UNA EXCEPCIÓN?

DEFINICIÓN DE EXCEPCIÓN:



Una excepción es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de ejecución

Un mecanismo de manejo de excepciones debe cumplir una serie de requerimientos generales

- Debe ser simple de usar y entender.
- Separación del código para el manejo de las excepciones del código normal
- Tratamiento uniforme de las excepciones
- Debe permitir que las acciones de recuperación sean programadas.

¿QUÉ HACER CUANDO OCURRE UNA EXCEPCIÓN?

Se levanta la excepción:



- Detener la ejecución normal del programa.
- Llamar a un subprograma (manejador de excepciones) que debe ejecutar acciones especiales.

Conceptos fundamentales

• MANEJADOR DE EXCEPCIONES

Subprograma encargado de llevar a cabo un conjunto de instrucciones que se ejecutan después de una excepción

• LEVANTAR EXCEPCIÓN

Acción de advertir una excepción en tiempo de ejecución.

Tipos de excepciones:

1. Manejo en lenguajes que no contenían implícito el manejo de excepciones
2. Manejo en lenguajes que sí lo contienen y permiten tener:
 - Excepciones predefinidas por el lenguaje (Implícitas)
 - Excepciones definidas por el programador (Explícitas)

Definición

C# envía una excepción cuando ocurre un error en el programa y detiene su ejecución.

Si deseamos que nuestro programa siga ejecutándose después del error usamos :

✓ **try** para poner en alerta al programa a cerca del código que puede lanzar una excepción.

✓ **catch** para capturar y manejar cada excepción que se lance.

✓ **finally** código que se ejecutará haya o no excepciones.

Definición

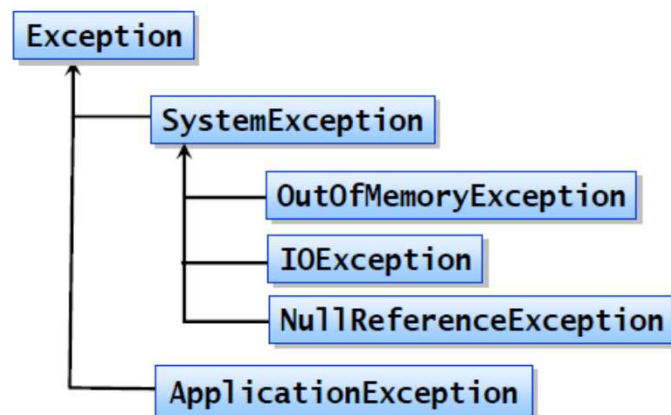
```
try
{
    [Bloque de código que puede causar errores]
}

catch
{
    [Qué hacer si sucede un error]
}

finally
{
    [De cualquier manera, hacer lo siguiente...]
}
```

Excepciones

- Todas las excepciones derivan de `System.Exception`



Algunas excepciones

| Clase de excepción | Significado |
|---------------------------------------|--|
| <code>DivideByZeroException</code> | Se produce cuando intenta dividir un valor entero o decimal entre cero |
| <code>IndexOutOfRangeException</code> | Un arreglo fue accedido con un índice ilegal (fuera de los límites permitidos) |
| <code>NullReferenceException</code> | Se intentó utilizar <code>null</code> donde se requería un objeto |
| <code>FormatException</code> | Se produce cuando el formato de un argumento no es el adecuado |

¿Qué hacer después de manejar la excepción?

Cuando el manejador termina se pueden hacer dos cosas:

- **Reanudar** la ejecución del bloque
- **Terminar** la ejecución del bloque y devolver el control al punto de invocación.

Tratamiento de excepciones

- Tratamiento de excepciones orientado a objetos :

```
try {
    ...      // bloque normal de código
}
catch {
    ...      // bloque que controla la excepción
}[ finally {
    ...      // bloque de finalización que siempre se ejecuta
}]
```

- Ejemplo:

```
try {
    Console.WriteLine("Escriba un número");
    int i = int.Parse(Console.ReadLine());
}
catch (OverflowException capturada)
{
    Console.WriteLine(capturada);
}
```

Tratamiento de excepciones

- Cada bloque *catch* captura una clase de excepción
- Un bloque *try* puede tener un bloque *catch* general que capture excepciones no tratadas (uno solo y el último de los bloques *catch*)
- Un bloque *try* no puede capturar una excepción derivada de una clase capturada en un bloque *catch* anterior

```
...
try
{
    Console.WriteLine("Escriba el primer número");
    int i = int.Parse(Console.ReadLine());
    Console.WriteLine("Escriba el segundo número");
    int j = int.Parse(Console.ReadLine());
    int k = i / j;
}
catch (OverflowException capturada) {
    Console.WriteLine(capturada); }
catch (DivideByZeroException capturada)
{Console.WriteLine(capturada); }
catch {...} // también: catch (Exception x) { ... }
```

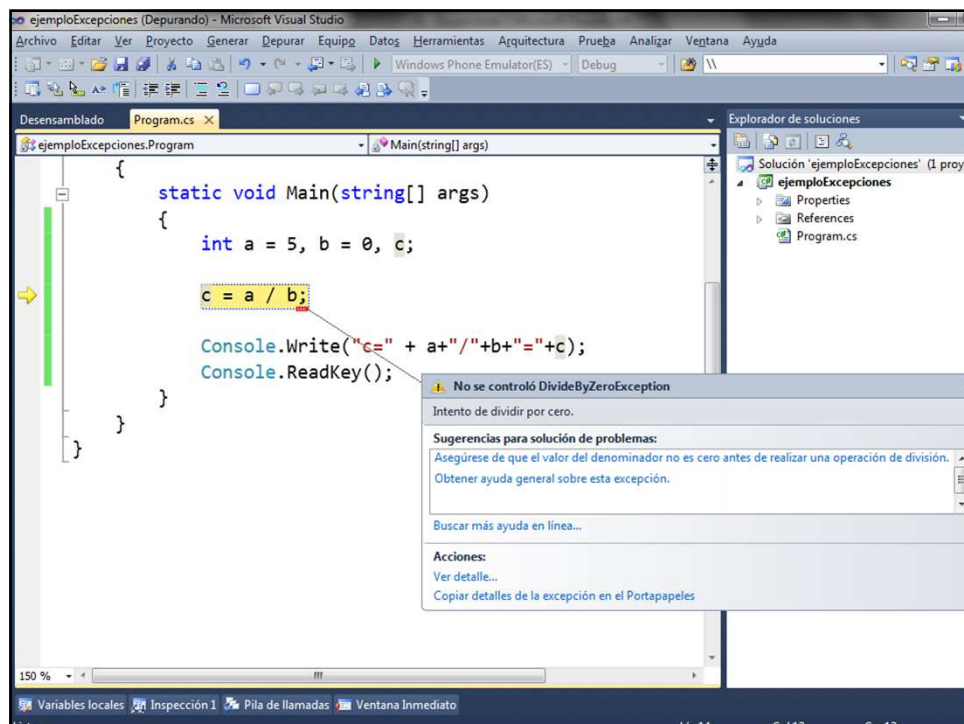

EJEMPLO SIN MANEJO DE EXCEPCIONES

```
static void Main(string[] args)
{
    int a = 5, b = 0, c;

    c = a / b;

    Console.WriteLine("c=" + a + "/" + b + "=" + c);
    Console.ReadKey();
}
```

Intenta ejecutar una
división por cero
(b = 0)



DivideByZeroException

```
static void Main(string[] args)
{
    int a = 5, b = 0, c;

    try
    {
        c = a / b;
    }
    catch (DivideByZeroException x)
    {
        Console.WriteLine(x.Message);
        Console.ReadKey();
        return;
    }

    Console.Write("c=" + a + "/" + b + "=" + c);
    Console.ReadKey();
}
```

Intenta ejecutar una
división por cero
(b = 0)

Captura la excepción
DivideByZeroException

Propiedad con el
mensaje de la
excepción

MENSAJES DE EXCEPCIONES



EJEMPLO CON MANEJO DE EXCEPCIONES

Form1

Manejo de Excepciones

Dividendo 2983

Divisor percy

Resultado Operación sin éxito

Calcular Salir

Excepciones

Ingrese números

Aceptar

EJEMPLO DE DIVISIÓN POR CERO

Form1

Manejo de Excepciones

Dividendo 2983

Divisor 0

Resultado Error: 'Division entre cero no es posible'

Calcular Salir

CODIFICACIÓN DEL BOTÓN PARA HACER LA DIVISIÓN

```
private void button1_Click(object sender, EventArgs e)
{
    int dividendo=0, divisor=0, resultado=0;
    try
    {
        dividendo = int.Parse(textBox1.Text);
        divisor = int.Parse(textBox2.Text);
        resultado = dividendo / divisor;
    }
    catch (Exception x)
    {
        MessageBox.Show("ERROR: "+x.Message);
    }
    finally
    {
        textBox3.Text = resultado.ToString();
    }
}
```

Captura cualquier excepción que se dispare

Propiedad con el mensaje de la excepción

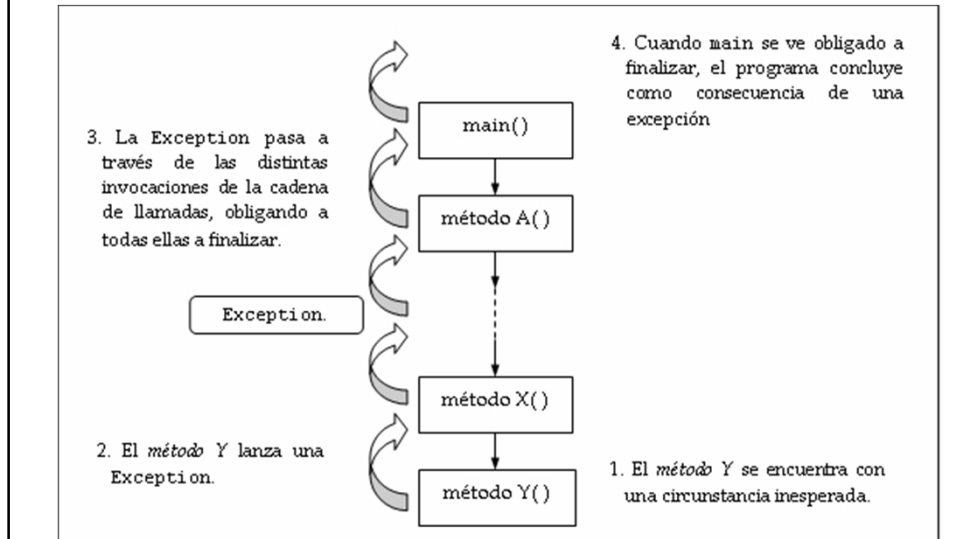
OTRO EJEMPLO PARA VALIDAR LA CAPTURA DE DATOS

```
int a=0;
bool Bandera = true;
do
{
    Bandera = false;
    try
    {
        Console.Write("Capture un número entero: ");
        a = int.Parse(Console.ReadLine());
    }
    catch (Exception x)
    {
        Console.WriteLine("ERROR: " + x.Message);
        Console.ReadKey();
        Bandera = true;
    }
} while (Bandera);
```

Captura cualquier excepción que se dispare

Propiedad con el mensaje de la excepción

CONTROL DE EXCEPCIONES



EJEMPLO CON TRY-CATCH

```
public static void Main(string[] args)
{
    int dato1 = 0, dato2 = 0, dato3;
    System.Console.WriteLine("Se inicia la aplicacion");
    try
    {
        dato1++;
        dato3 = dato1 / dato2;
        dato2++;
    }
    catch (System.DivideByZeroException e) {
        Console.WriteLine("Error: " + e.Message);
        dato3 = dato1;
    }
    //Otras sentencias
    Console.WriteLine(dato1 + " " + dato2 + " " + dato3);
}
```

CATCH atrapará solamente excepciones de tipo **DIVIDEBYZERO EXCEPTION**

TAREA

- Buscar 2 ejemplos con TRY CATCH que usen diferentes tipos de excepciones.



OverflowException

Por defecto, no se verifica el desborde aritmético

```
checked {
    int number = int.MaxValue;
    Console.WriteLine(++number);
}
```

OverflowException

Dispara una excepción
No se ejecuta la impresión.

```
unchecked {
    int number = int.MaxValue;
    Console.WriteLine(++number);
}
```

¿MaxValue + 1 es negativo?

-2147483648

Ejemplo sin verificación

```
static void Main(string[] args)
{
    int x = int.MaxValue;

    Console.WriteLine("x=" + x);
    // 2, 147, 483, 647

    x++;

    Console.WriteLine("x=" + x);
    // -2, 147, 483, 648 ??????
}
```

ERROR:
No verifica
OverflowException !!!

Ejemplo con verificación

```
static void Main(string[] args)
{
    int x = int.MaxValue;
    Console.WriteLine("x=" + x);
    // 2, 147, 483, 647
    try
    {
        checked
        {
            x++;
        }
    }
    catch(OverflowException)
    {
        Console.WriteLine("Número demasiado grande !!!");
        return;
    }
}
```

Solución:
Verificar
OverflowException

TIPOS DE EXCEPCIONES

Excepciones de sistema:

Cuando se realiza alguna operación no válida se lanza automáticamente.

Acceso a algún objeto que no existe, división por cero...

Excepciones de programador:

Se define una clase que herede de *Throwable* o de *Exception*.

Excepciones de usuario:

Gestiona la excepción mediante los bloques de código *try, catch, finally*.

Indica que el código producirá una excepción que no se tratará dentro de él y se pasará al método superior utilizando **throw**.

INSTRUCCION THROW

- La instrucción **throw** se utiliza para señalar la aparición de una situación anómala (excepción) durante la ejecución del programa.
- Se puede utilizar una instrucción [throw](#) en el bloque **catch** para volver a producir la excepción, la cual ha sido capturada por la instrucción **catch**.
- El programador puede disparar una excepción mediante:
 - `throw new Exception("Error:");`

EJEMPLO DE LA INSTRUCCION THROW (Programa principal)

```
static void Main(string[] args)
{
    int a=3, b=0, c=0;

    try
    {
        c = CalcularDivision(a, b);
    }
    catch (Exception x)
    {
        Console.WriteLine(x.Message);
    }
    finally
    {
        Console.WriteLine(a+"/"+b+"="+c);
    }
    Console.ReadKey();
}
```

EJEMPLO DE LA INSTRUCCION THROW (Método CalcularDivision)

```
static int CalcularDivision(int numerador, int denominador)
{
    if (denominador == 0)
        throw new Exception("El denominador NO debe ser cero");
    else
        return (numerador / denominador);
}
```

EJEMPLO 1: Excepción General

```
static void Main( )
{
    try
    { System.Console.WriteLine(" Introduce un número: ");
      int a = System.Convert.ToInt32
        (System.Console.ReadLine() );
    }
    catch ( Exception e )
    { System.Console.WriteLine(" Ha habido un error..." +
      e.Message);
    }
    finally
    { System.Console.WriteLine(" Con error y Sin error, este
      mensaje aparece. ");
      System.Console.ReadLine();
    }
}
```

EJEMPLO 2: Generar una excepción THROW

```
using System;
class MainClass {
    static void ProcesarCadena(string s) {
        if (s == null)
        { throw new ArgumentNullException(); }
    }
    static void Main()
    {
        try
        { string s = null;
          ProcesarCadena(s);
        }
        catch (Exception e)
        { Console.WriteLine("{0} Excepcion capturada.", e);
        }
    }
}
```

EJEMPLO 3: Ordenar instrucciones CATCH

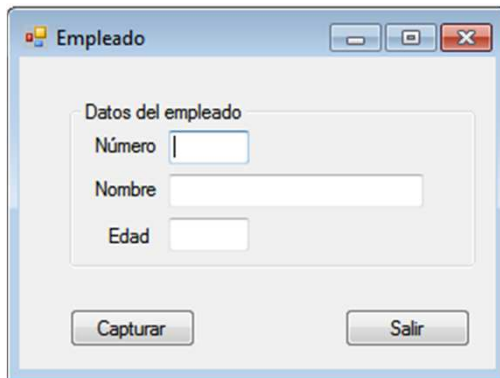
```
static void ProcesarCadena (string s)
{
    if (s == null)
    { throw new ArgumentNullException(); }
}
static void Main() {
try {
    string s = null;
    ProcesarCadena(s);
}
//Mas específico
catch (ArgumentNullException e)
{ Console.WriteLine("{0} First exception caught.", e); }
//Menos específico
catch (Exception e)
{ Console.WriteLine("{0} Second exception caught.", e); }
} }
```

TAREA

- Modificar 4 ejemplos anteriores de CLASES, incluyendo en ellos el manejo de errores (TRY-CATCH-FINALLY) donde sea necesario.
- Voluntarios para presentar **1 ejemplo en clase** se contará como **doble participación**.



Uso de propiedades para validar la captura de datos



The screenshot shows a window titled 'Empleado' with a standard Windows title bar. Inside, there's a group box labeled 'Datos del empleado'. It contains three text input fields: 'Número' (with a small cursor), 'Nombre', and 'Edad'. At the bottom of the window are two buttons: 'Capturar' and 'Salir'.

- Capturar datos y validar que se tecleen correctamente.
- No dejar datos en blanco
- Rango de valores permitido

Uso de propiedades para validar la captura de datos (declaración de atributos)

```
class Empleado
{
    // Atributos privados
    private int numero;
    private string nombre;
    private int edad;

    :
    :
    :
```

Validar la captura del número de empleado

```
// Propiedad pública del número
public int Numero
{
    get { return numero; }
    set
    {
        numero = value;
        if (numero <= 0)
            throw new Exception("Dato incorrecto para el número");
    }
}
```

Validar la captura del nombre del empleado

```
public string Nombre
{
    get { return nombre; }
    set
    {
        nombre = value;
        if (nombre == "")
            throw new Exception("No debe dejar en blanco el nombre");

        foreach (char letra in nombre) {
            // Caracteres permitidos
            switch (letra)
            {
                case 'A': continue;
                case 'É': continue;
                case 'I': continue;
                case 'Ó': continue;
                case 'U': continue;
                case 'Ñ': continue;
                case 'Ú': continue;
                case ' ': continue;
            }

            if (letra < 'A' || letra > 'Z')
                throw new Exception("Solamente se permiten mayúsculas en el nombre (no capturar números ni otros caracteres)");
        }
    }
}
```

Validar la captura de la edad del empleado

```
// Propiedad pública de la edad
public int Edad
{
    get { return edad; }
    set
    {
        edad = value;
        if (edad < 0 || edad > 110)
            throw new Exception("Dato fuera de rango en la edad");
    }
}
```

Validar la captura de datos desde el botón

```
private void btnCapturar_Click(object sender, EventArgs e)
{
    Empleado miEmpleado = new Empleado();

    try
    {
        miEmpleado.Numero = int.Parse(txtNumero.Text);
    }
    catch (Exception x)
    {
        MessageBox.Show(x.Message);
        txtNumero.Text = "";
        txtNumero.Focus();
        return;
    }
}
```