

Excepciones

El mecanismo de control de excepciones es muy parecido al de C++.

Las excepciones son un mecanismo de los lenguajes de programación para tratar situaciones anómalas (generalmente errores inesperados) de una forma sencilla y elegante, si bien no conviene abusar de ellas.

Las excepciones pueden generarse en un proceso o hebra de nuestra aplicación (con la sentencia `throw`) o pueden provenir del entorno de ejecución de la plataforma .NET.

Son mejores que las sentencias `return` porque no pueden ser ignoradas y no tienen por qué tratarse en el punto en que se producen.

Exactamente igual que en C++ y en Java:

- La sentencia **throw** lanza una excepción (una instancia de una clase derivada de **System.Exception**, que contiene información sobre la excepción:
 - **Message**: Texto legible para el usuario donde se describe el error,
 - **StackTrace**: Estado de la pila de llamadas en el momento en que se inició la excepción,
 - **HelpLink**: el archivo de ayuda de la excepción ,
 - **InnerException**: Cuando existe una relación causal entre dos o más excepciones, la propiedad **InnerException** incluye esta información. La excepción externa se inicia en respuesta a esta excepción interna.
- El **bloque try** delimita código que podría generar una excepción.
- El **bloque catch** indica cómo se manejan las excepciones. Se puede relanzar la excepción capturada o crear una nueva si fuese necesario. Se pueden especificar distintos bloques `catch` para capturar distintos tipos de excepciones. En ese caso, es recomendable poner primero los más específicos (para asegurarnos de que capturamos la excepción concreta).
- El **bloque finally** incluye código que siempre se ejecutará (se produzca o no una excepción).

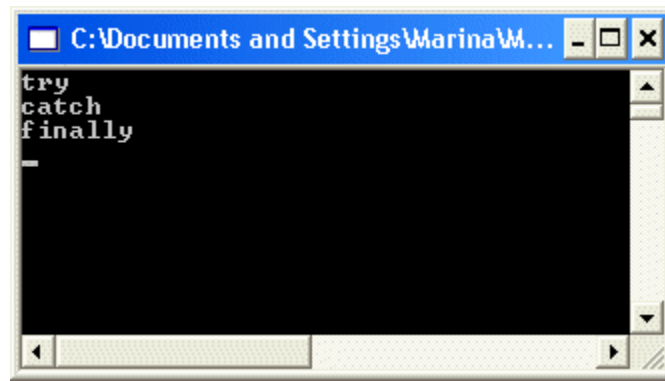
Ejemplo 1

```
try {  
    Console.WriteLine("try");  
    throw new Exception("message");  
}  
catch (ArgumentNullException e) {  
    Console.WriteLine("Null argument");  
}
```

```

catch {
    Console.WriteLine("catch");
}
finally {
    Console.WriteLine("finally");
}

```



Ejemplo 2

Aquí se usan las siguientes clases del framework:

ArithmeticException (Clase)

Excepción iniciada a causa de los errores de una operación aritmética, de conversión de tipos o de conversión de otra naturaleza.

ArgumentNullException (Clase)

Excepción que se inicia cuando se pasa una referencia nula (**Nothing** en Visual Basic) a un método que no la acepta como argumento válido.

```

static void Main(string[] args)
{
    int numerador = 10;
    Console.WriteLine ("Numerador es = {0}", numerador);
    Console.Write ("Denominador = ");
    string strDen = Console.ReadLine();

    int denominador, cociente;

    try
    {
        Console.WriteLine("--> try");
        denominador = Convert.ToInt16(strDen);
        cociente = numerador / denominador;
        Console.WriteLine ("Cociente = {0}", cociente);
    }

    catch (ArithmeticException e)
    {
        Console.WriteLine("--> catch");
    }
}

```

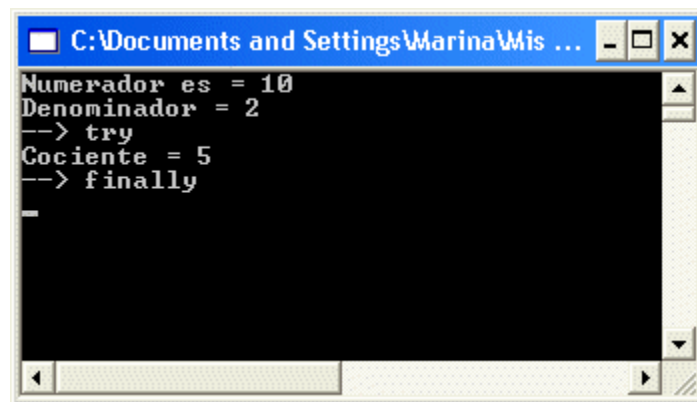
```

        Console.WriteLine("Excep. aritmética");
        Console.WriteLine("ArithmeticException Handler: {0}", e.ToString());
    }
    catch (ArgumentNullException e)
    {
        Console.WriteLine("--> catch");
        Console.WriteLine("Excep. de argumento nulo");
        Console.WriteLine("ArgumentNullException Handler: {0}",
e.ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine("--> catch");
        Console.WriteLine("generic Handler: {0}", e.ToString());
    }
    finally
    {
        Console.WriteLine("--> finally");
    }

    Console.ReadLine();
}
}

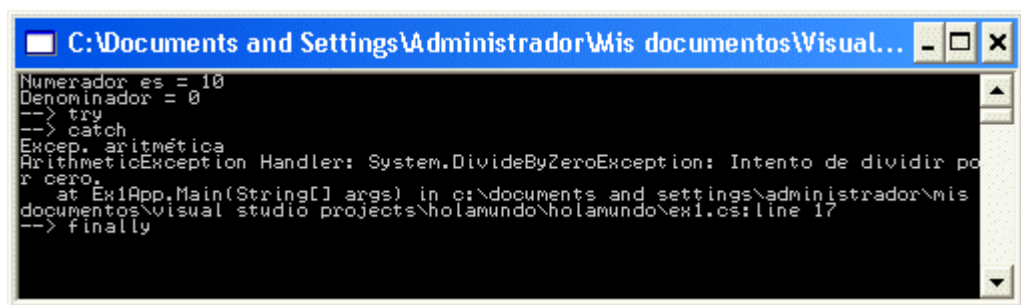
```

Cuando toda funciona sin problemas:



A screenshot of a Windows command prompt window. The title bar reads "C:\Documents and Settings\Marina\Mis ...". The command prompt shows the following text: "Numerador es = 10", "Denominador = 2", "--> try", "Cociente = 5", "--> finally", and a single hyphen "-" on the next line.

Cuando se intenta dividir por cero:



A screenshot of a Windows command prompt window. The title bar reads "C:\Documents and Settings\Administrador\Mis documentos\Visual...". The command prompt shows the following text: "Numerador es = 10", "Denominador = 0", "--> try", "--> catch", "Excep. aritmética", "ArithmeticException Handler: System.DivideByZeroException: Intento de dividir por cero.", "at Ex1App.Main(String[] args) in c:\documents and settings\administrador\mis documentos\visual studio projects\holamundo\holamundo\ex1.cs:line 17", and "--> finally".

Cuando se produce desbordamiento:

Si una aplicación controla las excepciones que se producen durante la ejecución de un bloque de código de la aplicación, el código debe incluirse dentro de una instrucción **try**. El código de la aplicación que se está incluido dentro de una instrucción **try** es un bloque **try**. El código de la aplicación que controla las excepciones iniciadas por un bloque **try** se coloca dentro de una instrucción **catch**, y se denomina bloque **catch**. Puede haber cero o más bloques **catch** asociados a un bloque **try**, y cada uno de los bloques **catch** incluye un filtro de tipos que determina los tipos de excepciones que controla.

Cuando se produce una excepción en un bloque **try**, el sistema busca los bloques **catch** asociados en el orden en el que aparecen en el código de la aplicación, hasta que encuentra un bloque **catch** que controla la excepción. Un bloque **catch** controla una excepción de tipo **T** si el filtro de tipos del bloque **catch** especifica **T** o cualquier tipo del que se derive **T**. El sistema detiene la búsqueda cuando encuentra el primer bloque **catch** que controla la excepción. Por este motivo, el bloque **catch** que controla un tipo debe especificarse en el código de la aplicación delante del bloque **catch** que controla sus tipos base, tal y como se muestra en el ejemplo que se incluye tras esta sección. El bloque **catch** que controla **System.Exception** se especifica en último lugar.

Si ninguno de los bloques **catch** asociados al bloque **try** actual controla la excepción y el bloque **try** actual está anidado dentro de otros bloques **try** en la llamada actual, se realiza una búsqueda de los bloques **catch** asociados al siguiente bloque **try** envolvente. Si no se encuentra ningún bloque **catch** para la excepción, el sistema busca los niveles de anidamiento anteriores de la llamada actual. Si en la llamada actual no se encuentra ningún bloque **catch** para la excepción, ésta asciende por la pila de llamadas y en el marco de pila anterior se busca un bloque **catch** que controle la excepción. La búsqueda en la pila de llamadas continúa hasta que se controla la excepción o hasta que no hay más marcos en la pila de llamadas. Si se alcanza la parte superior de la pila de llamadas sin haber encontrado un bloque **catch** que controle la excepción, el controlador de excepciones predeterminado controla la excepción y la aplicación termina.

Los tipos de excepción admiten las siguientes características:

- Texto legible para el usuario donde se describe el error. Cuando se produce una excepción, el motor de tiempo de ejecución pone a disposición del usuario un mensaje de texto donde se informa de la naturaleza del error y se recomienda llevar a cabo una determinada acción para solucionar el problema. Este mensaje de texto se guarda en la propiedad [Message](#) del objeto de excepción. Durante la creación del objeto de excepción, puede pasarse una cadena de texto al constructor para que se describan los detalles de esa excepción en concreto. Si no se le proporciona al constructor ningún argumento de mensaje de error, se utiliza el mensaje de error predeterminado.
- Estado de la pila de llamadas en el momento en que se inició la excepción. La propiedad [StackTrace](#) incluye un seguimiento de pila que puede utilizarse para determinar la parte del código en la que se ha producido un error. El seguimiento de pila muestra todos los métodos a los que se ha llamado y los números de línea del archivo de código fuente donde se realizan las llamadas.

Existen dos categorías de excepciones en la clase base **Exception**:

- Las clases de excepción predefinidas de Common Language Runtime que se derivan de [SystemException](#).
- Las clases de excepción de aplicaciones definidas por el usuario que se derivan de [ApplicationException](#).

Exception incluye varias propiedades que ayudan a identificar la ubicación en el código de la excepción, el tipo de excepción, el archivo de ayuda de la excepción y el motivo de la excepción: **StackTrace**, [InnerException](#), **Message**, [HelpLink](#), [HResult](#), [Source](#) y [TargetSite](#).

Cuando existe una relación causal entre dos o más excepciones, la propiedad **InnerException** incluye esta información. La excepción externa se inicia en respuesta a esta excepción interna. El código que controla la excepción externa puede utilizar la información de la excepción interna anterior para controlar el error de forma más adecuada.

La cadena de mensaje de error que se pasa al constructor durante la creación del objeto de excepción debe localizarse, y puede suministrarse desde un archivo de recursos mediante [ResourceManager](#). Para obtener más información sobre los recursos localizados, vea la información general sobre el espacio de nombres [System.Resources](#) y Empaquetar e implementar aplicaciones de .NET Framework.

Para proporcionar al usuario abundante información sobre el motivo de la excepción, la propiedad **HelpLink** puede incluir una dirección URL (o URN) a un archivo de ayuda.

Exception utiliza HRESULT COR_E_EXCEPTION, que tiene el valor 0x80131500.

Para obtener una lista con los valores de propiedad iniciales de una instancia de **Exception**, vea los constructores **Exception**.