Excepciones

Programación 2

Introducción

Las excepciones en C# ofrecen un modo estructurado, uniforme y con seguridad de tipos para el manejo de situaciones de error tanto a nivel de sistema como a nivel de aplicación.

Introducción

El control de excepciones utiliza las palabras clave **try, catch y finally** para intentar realizar acciones que podrían plantear problemas, controlar errores cuando consideras que sea razonable y limpiar los recursos después.

Las excepciones se crean mediante la palabra clave **throw**.

Ejemplo

```
class TestExcepcion {
    static double Dividir(double x, double y) {
        if (y == 0)
            throw new System.DivideByZeroException();
        return x / y;
    }
    static void Main() {
        double a = 98, b = 0;
        double result = 0;
        try {
            result = Dividir(a, b);
            Console.WriteLine("{0} dividido por {1} = {2}", a, b, result);
        } catch (DivideByZeroException e) {
            Console.WriteLine("Se intentó dividir por cero.");
        }
    }}
```

Características

Las excepciones tienen las siguientes características:

- Las excepciones son tipos que se derivan en última instancia de **System.Exception**.
- Utilice un bloque **try** alrededor de las instrucciones que puedan generar excepciones.

Características

- Cuando se produce una excepción dentro del bloque **try**, el flujo de control salta al primer controlador de excepciones asociado que se encuentre en cualquier parte de la pila de llamadas. En C#, la palabra clave **catch** se utiliza para definir un controlador de excepciones.

Características

- Si no hay un controlador de excepciones para una excepción determinada, el programa deja de ejecutarse y presenta un mensaje de error.

Características

- No detecte una excepción a menos que pueda controlarla y dejar la aplicación con un estado conocido. Si detecta una excepción System.Exception, vuelva a producirla mediante la palabra clave **throw** al final del bloque **catch**.

Ejemplo

```
...
catch (Exception e) {
    // Hacemos algo y lanzamos una nueva excepción.
    throw new MiException("ERROR!!.", e);
}
```

Características

- Si un bloque **catch** define una variable de excepción, puede utilizar dicho bloque para obtener más información sobre el tipo de excepción que se ha producido.
- Un programa que utiliza la palabra clave **throw** puede generar explícitamente excepciones.

Ejemplo

```
try {
        sw = new StreamWriter("C:\test.txt");
        sw.WriteLine("Hola");
} catch (FileNotFoundException ex) {
        // Mostramos información específica de la
        // excepción.
        System.Console.WriteLine(ex.ToString());
}
```

Características

- Los objetos de excepción contienen información detallada sobre el error, tal como el estado de la pila de llamadas y una descripción de texto del error.

Características

- El código de un bloque finally se ejecuta aunque se produzca una excepción. Use un bloque finally para liberar recursos, por ejemplo, para cerrar las secuencias o archivos que se abrieron en el bloque try.

Características

Una instrucción **try** puede contener más de un bloque **catch**.

Se ejecuta la primera instrucción catch que puede controlar la excepción; cualquier instrucción catch posterior, aun cuando sea compatible, se omite.

Por consiguiente, los bloques catch siempre deberían ordenarse de más específico (o más derivado) a menos específico.

Ejemplo

```
static void Test(){
    StreamWriter sw = null;
    try {
        sw = new StreamWriter("C:\test.txt");
        sw.WriteLine("Hola");
    } catch (FileNotFoundException ex) { // Excepcion mas específica
        System.Console.WriteLine(ex.ToString());
    } catch (IOException ex) { // Excepcion menos específica
        System.Console.WriteLine(ex.ToString());
    } finally  {
        sw.Close(); // Siempre se ejecuta, nos aseguramos que siempre se cierre el archivo.
    }
    System.Console.WriteLine("Fin!");
}
```

Crear y producir excepciones

Las excepciones se utilizan para indicar que se ha producido un error mientras el programa está en ejecución.

Los objetos de excepción que describen un error se crean y, a continuación, se producen con la palabra clave **throw**.

Crear y producir excepciones

Los programadores deberían iniciar excepciones cuando se cumpla al menos una de las siguientes condiciones:

- El método no puede finalizar su funcionalidad definida.

Crear y producir excepciones

```
static void CopyObject(SampleClass original){
    if (original == null) {
        throw new System.ArgumentException("Parameter cannot be null", "original");
    }
}
```

Crear y producir excepciones

- Se realiza una llamada inadecuada a un objeto, basada en el estado del objeto.
- Ejemplo: tratar de escribir en un archivo de sólo lectura. En los casos en que el estado del objeto no permite realizar una operación, produzca una instancia de InvalidOperationException o un objeto basado en una derivación de esta clase.

Crear y producir excepciones

```
class ProgramLog {
    FileStream logFile = null;
    void OpenLog(FileInfo fileName,FileMode mode) { /* … */ }
    void WriteLog() {
      if (!this.logFile.CanWrite) {
        throw new InvalidOperationException("Log es read-only");
      }
      // Escribimos los datos en el log.
    }
}
```

Crear y producir excepciones

- Cuando un argumento para un método provoca una excepción.
- En este caso, se debería detectar una excepción y se debería crear una instancia de ArgumentException. La excepción original se debería pasar al constructor de ArgumentException como el parámetro.

Crear y producir excepciones

```
static int GetValueFromArray(int[] array, int index) {
    try {
        return array[index];
    } catch (IndexOutOfRangeException ex) {
       ArgumentException argEx = new ArgumentException("Index is
out of range", "index", ex);
        throw argEx;
    }
}
```

Excepciones propias

- Si desea mediante programación, distinguir ciertas condiciones de error de otras, puede crear sus propias excepciones definidas por el usuario.
- NET Framework proporciona una jerarquía de clases de excepción que, en última instancia, derivan de la clase base Exception.

Excepciones propias

- Cada una de estas clases define una excepción específica, por lo que en muchos casos sólo hay que detectar la excepción. También se pueden crear clases de excepción personalizadas derivándolas de la clase Exception.

Excepciones propias

- Cuando se creen excepciones personalizadas, es recomendable finalizar el nombre de la clase de la excepción definida por el usuario con la palabra "Excepción".

Excepciones propias

```
public class EmployeeListNotFoundException: Exception {
    public EmployeeListNotFoundException()
    {
    }
    public EmployeeListNotFoundException(string message)
        : base(message)
    {
    }
    public EmployeeListNotFoundException(string message, Exception inner)
        : base(message, inner)
    {
    }
}
```

Clases de excepciones más comunes

- **System.OutOfMemoryException**: Se lanza cuando falla un intento de reservar memoria mediante el operador **new**.
- **System.StackOverflowException**: Se lanza cuando la pila de ejecución se llena por tener demasiadas llamadas a método pendientes. Situación que suele darse en recursiones muy profundas.

Clases de excepciones más comunes

- **System.NullReferenceException**: Se lanza cuando se pretende acceder mediante una referencia **null** a un supuesto objeto.
- **System.IndexOutOfRangeException**: Se lanza cuando se intenta acceder a un array mediante un índice menor que cero o mayor que el límite del array.

Clases de excepciones más comunes

- **System.ArithmeticException**: Es la clase base de las excepciones que se dan durante las operaciones aritméticas, tales como DivideByZeroException y OverflowException
- **System.DivideByZeroException**: Se lanza cuando se intenta dividir un valor de tipo integral por cero.