**Carrera:** Ing. Sistemas Computacionales

**Materia y Clave:** Programación orientada a objetos SCD-1020

**Unidad:** 5

**Tema:**

“Investigación”

**No. Control:** 14212104

**Nombre del alumno:** Aldo Emanuel Alfaro Moreno

**Nombre de la Maestra:** M.C.C. Luz Elena Cortés Galván

**Horario:** 11-12     **Aula:** 504

**Fecha de Entrega:** 13 noviembre del 2015

5.1 Definición Excepciones en C#

Una excepción se define como un evento que ocurre durante la ejecución de un programa y la interrumpe momentáneamente.

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. El motor de ejecución busca el controlador de excepciones más compatible.

El lenguaje de programación C# permite controlar la generación de una excepción a través de las construcciones try, catch, finally, y throw. La primera de estas -try-, encierra el bloque de código que pudiera generar una excepción, si ocurriera una excepción en el bloque try, el o los bloques catch se encargan de capturar la excepción para un tratamiento o manejo específico a la excepción generada. Con finally, se ejecutan operaciones una vez se finaliza el conjunto de sentencias encerradas en el bloque try, o después de haber sido la excepción en un bloque catch. (En las secciones posteriores nos detendremos en cada una de estas construcciones para mostrar varios detalles importantes a la manejar una excepción.)

Estos programas y dispositivos vienen preparados para responder a esas situaciones excepcionales a través de mensajes que nos advierten sobre el intento de ejecutar una operación que a razón matemática no está definida.

Este mismo concepto lo podemos aplicar en la programación de un artefacto o elemento de programa (e.g., método, propiedad, clase, enumeración) para responder a intentos de ejecución de operaciones inválidas.

Más específicamente una operación inválida la definimos, en el contexto de C#, como:

-División entre cero.

-Acceso a un elemento de un arreglo (o colección) por debajo del límite inferior (0), o por encima del límite superior (capacidad).

-Intento de escritura de un archivo inexistente en el sistema de archivos o en una unidad de red.

-Conexión fallida a un servidor de base de datos.

-Argumento de método no inicializado (null).

-Superación del límite de memoria de trabajo.

Entre muchas más...

Como ha sido mencionado, el lenguaje de programación C# provee las construcciones sintácticas y semánticas para manejar o controlar las excepciones que se pudieran generar en una sección o bloque de código específicos. Se tratan de las siguientes palabras claves:

try-catch

La instrucción try-catch consta de un bloque try seguido de una o más cláusulas catch que especifican controladores para diferentes excepciones.

Cuando se produce una excepción, Common Language Runtime (CLR) busca la instrucción catch que controla esta excepción. Si el método que se ejecuta actualmente no contiene un bloque catch, CLR busca el método que llamó el método actual, y así sucesivamente hasta la pila de llamadas. Si no existe ningún bloque catch, CLR muestra al usuario un mensaje de excepción no controlada y detiene la ejecución del programa.

El bloque try contiene el código protegido que puede producir la excepción. El bloque se ejecuta hasta que se produce una excepción o hasta que se completa correctamente. Por ejemplo, el intento siguiente de convertir un objeto null produce la excepción NullReferenceException:

Finally:

-se escribe código que se ejecuta si ocurre una excepción o si no de todos modos lo realiza.

-seccion opcional

Utilizando un bloque finally, puede limpiar los recursos que se asignan en un bloque try, y puede ejecutar código incluso si se produce una excepción en el bloque try. Normalmente, las instrucciones de un bloque finally se ejecuta cuando el control sale de una instrucción try. La transferencia del control puede producirse como resultado de la ejecución normal, de la ejecución de la instrucción break, continue, goto o return, o de la propagación de una excepción fuera de la instrucción try .

Describiré cada una de estas en las siguientes secciones.

Ejemplo

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static void Main()

{

Console.WriteLine("Example 1: re-throw inside of another try block:");

try

{

Console.WriteLine("--outer try");

try

{

Console.WriteLine("----inner try");

throw new Exception();

}

catch

{

Console.WriteLine("----inner catch");

throw;

}

finally

{

Console.WriteLine("----inner finally");

}

}

catch

{

Console.WriteLine("--outer catch");

// swallow

}

finally

{

Console.WriteLine("--outer finally");

}

Console.WriteLine();

Console.WriteLine("Example 2: re-throw outside of another try block:");

try

{

Console.WriteLine("--try");

throw new Exception();

}

catch

{

Console.WriteLine("--catch");

//throw;

}

finally

{

Console.WriteLine("--finally");

}

Console.ReadLine();

//detener

Console.ReadKey();

}

}

}

5.2 Tipos de las excepciones

-SystemException: permite que el código maneje las distintas excepciones de forma aplicada a un tipo de excepción específico

-System.ArithmeticException

-System.ArrayTypeMismatchException

-System.DivideByZeroException

-System.StackTrace: devuelve las invocaciones en el momento en que se captura la excepción, lo que puede ser útil para escenarios de registro de actividad y depuración

-etc.

5.3 Propagación de las excepciones (try catch finally throw)

Try-catch:

La instrucción try-catch. Consta de un bloque try seguido de una o más cláusulas catch, las cuales especifican controladores para diferentes excepciones.

Ejemplo:  
using System; class MainClass {

static void ProcessString(string s)

{

if (s == null)

{

throw new ArgumentNullException();

}

}

static void Main()

{

try

{

string s = null;

ProcessString(s);

}

catch (Exception e)

{

Console.WriteLine("{0} Exception caught.", e);

}

}

}

Try:

-intenta ejecutar las sentencias (instrucciones) de un programa

El bloque try contiene el código protegido que puede causar la excepción. Este bloque se ejecuta hasta que se produce una excepción o hasta completarse satisfactoriamente.

Catch:

-atrapa las excepciones que ocurren en el programa

-pueden ser mas de una en un programa (todas las necesarias)

La cláusula catch se puede utilizar sin argumentos, en cuyo caso captura cualquier tipo de excepción y se conoce como cláusula catch general. También puede aceptar un argumento de objeto derivado de System.Exception, en cuyo caso trata una excepción específica.

Throw:

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.

-se utiliza para lanzar o producir excepciones durante el código del programa

-señala una situación anómala durante la ejecución del programa.

Normalmente, la instrucción throw se usa con las instrucciones try-catch o try-finally. Se puede usar una instrucción throw en un bloque catch para volver a iniciar la excepción detectada por el bloque catch. En este caso, la instrucción throw no toma un operando de excepción.

Finally:

es el comando al que se acudirá siempre (si sucede el error o no).

Ejemplo:

try

{

smtp.Send(Email);

}

catch (Nombredelaexcepcion excepcion)

{

Console.WriteLine(“mensaje del error" + excepcion.message);

Console.WriteLine(“mensaje del error" + excepcion.StackTrace);

}

finally

{

Console.WriteLine(“mensaje enviado exitosamente");

}

5.4 propiedades de las excepciones

1.-Message: proporciona mensaje que describe la excepción actual

2.-StackTrace: proporciona la cadena de texto de la ruta inmediata

3.-Source: proporciona el nombre de la aplicación o nombre del objeto que causa el error

4.-InnerException: proporciona la instancia que causó la excepción actual

5.-ToString: convierte a una cadena de texto

6.-TargetSite: proporciona el método que arroja que excepción actual

7.-HelpLink: proporciona o pone la ruta de la ayuda asociada con la excepción

8,-Data: proporciona la colección de la llave o valor que el usuario definió en la aplicación

9.-HResult: proporciona o pone un valor numérico que es asignada a una excepción específica.

5.5 Creación y manejo de excepciones definidas por el usuario

-crear una clase que herede de : ApplicationException

-formato general: public class NombreException: ApplicationException

Formato general para la Creación de una excepción definida por el usuario

class nombre de la excepción : Exception { // mensaje}

-Agregar 3 constructores (constructor vacío opcional)

-Terminar nombre de la excepción con palabra Exception <opcional>

Evite las jerarquías de excepciones profundas.

Derive las excepciones de System.Exception o de una de las demás excepciones de base comunes.

Observe que Capturar y generar tipos de excepción estándar incluye una instrucción que especifica que no debería derivar excepciones personalizadas de ApplicationException.

Finalice los nombres de clases de excepción con el sufijo Exception.

Haga que las excepciones sean serializables. Una excepción debe ser serializable para funcionar correctamente atravesando dominios de aplicación y límites de interacción remota.

Asegúrese de que los nombres y tipos de los parámetros de los constructores sean iguales.

El manejo de excepciones consiste en solo cuatro palabras reservadas: try, catch, throw y finally.

Cuando un método fracasa en su objetivo y no puede continuar, devuelve una excepción al método que lo ha invocado mediante la utilización de la palabra throw.

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

Ejemplo:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace empacadora

{

class Program

{

class ExcepcionEmpacadora : Exception

{

public ExcepcionEmpacadora()

{

Console.WriteLine("Son demasiados paquetes, favor de detener produccion por hoy");

}

}

static void Main(string[] args)

{

//variables auxiliares

int opcion = 0;

int paquetes=0;

do

{

try

{

//menu

Console.Clear();

Console.WriteLine("\t--Menu--");

Console.WriteLine("\n<1> para empaquetar mercancia");

Console.WriteLine("\n<2> para salir");

opcion = Convert.ToInt16(Console.ReadLine());

switch (opcion)

{

case 1:

Console.Clear();

Console.Write("Ingrese cantidad de paquetes en existencia: ");

paquetes = Convert.ToInt32(Console.ReadLine());

if (paquetes > 10000)

{

throw new ExcepcionEmpacadora();

}

if (paquetes < 0)

{

throw new InvalidOperationException();

}

else

{

Console.Write("{0} paquete(s) de mercancia se ha(n) envuelto exitosamente", paquetes);

Console.ReadKey();

}

break;

case 2:

Console.Clear();

Console.WriteLine("saliendo...");

Console.ReadKey();

break;

default:

Console.Clear();

Console.WriteLine("Ingrese opcion válida");

Console.ReadKey();

break;

}

//capturar datos

}

catch(ExcepcionEmpacadora EE)

{

Console.WriteLine("\nMensaje de error :" + EE.Message);

Console.WriteLine("\nRuta de error :" + EE.StackTrace);

Console.ReadKey();

}

catch (InvalidOperationException IE)

{

Console.WriteLine("\nMensaje de error :" + IE.Message);

Console.WriteLine("\nRuta de error :" + IE.StackTrace);

Console.ReadKey();

}

catch (FormatException FE)

{

Console.WriteLine("\nMensaje de error :" + FE.Message);

Console.WriteLine("\nRuta de error :" + FE.StackTrace);

Console.ReadKey();

}

} while (opcion != 2);

}

}

}