# Principios del lenguaje

## Tipos, variables, constantes y ambito

### Cuestiones generales

PRIMERO EL EN FONDO LOS LENGUAJES DE .NET DEFINEN SUS TIPOS COMO SYSTEM.X, POR ELLO UTILIZAN ALIAS PARA NO ESCRIBIR TODO. POR EJEMPLO SYSTEM.BOOLEAN=BOOLEAN. SEGUNDO, COMO LA DECLARACION DE VARIABLES DEPENDE DEL USUARIO, ESTA PROPENSA A ERROR, POR ELLO EXISTE LA PROMOCIÓN DE TIPOS, PERO AUN ASI PUEDE EXISTIR ERROR. ESTO SE PUEDE EVITAR CON LA UTILIZACION DE LITERALES O CON LA CONVERSION EXPLICITA, LA CUAL SIRVE ADEMAS PARA NO PERDER PRECISION EN LA CONVERSION.

EN C# SE TIENEN QUE INICIALIZAR LAS VARIABLES ANTES DE USARSE. AUNQUE SE INICIALICEN EN DISTINTAS LÍNEAS, PERO ES NECESARIO HACERLO ANTES DE USARLAS

LITERALES NUMERICOS: EN TODOS LOS VALORES CON DECIMALES SE ASUME QUE SON DOUBLE. HACE PROMOCION DE TIPOS, POR EJEMPLO SI SE ASIGNA UN LITERAL A UN AVARIABLE INT, EL COMPILADOR BUSCARA EN INT, UINT, LONG, ULONG; PARA ASIGNARLA AUTOMATICAMENTE

AL TIPO MAS ADECUADO. PERO SI EL LITERAL NO CONCUERDA CON ALGUNO DE LOS 4 PROVOCA ERROR.

OJO: CUANDO POR ALGUNA RAZON HACEMOS QUE UNA VARIABLE DE CIERTO TIPO SE DEBORDE PUEDE DAR RESULTADOS CONTRADICTORIOS, POR EJEMPO LA MUNTIPLICACION DE DOS ENTEROS 1234567\*1234567 DA UN NUMERO NEGATIVO ESTO ES PORQUE EL COMPILADOR GENERA AUTOMATICAMENTE CODIGO DE DESBORDAMIENTO SILENCIOSO PORQUE ES RESULTADO ES UN NUMERO QUE REBASA EL LIMITE DEL ENTERO.

### Tipos de datos, operadores

SYSTEM.BOLEAN bool

SYSTEM.BYTE byte (0 a 255)

SYSTEM.INT16 short (-32.768 a 32.767)

SYSTEM.INT32 int (-2.147.483.648 a 2.147.483.647)

SYSTEM.INT64 long (-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807)

SYSTEM.SINGLE float (±1.401298E−45 a ±3.402823E+38)

SYSTEM.DOUBLE double (±4.94065645841246E−324 a ±1.79769313486232E+308)

SYSTEM.DECIMAL decimal (−7.9228162514264337593543950335 a +7.9228162514264337593543950335) \*\*\*CALC $

SYSTEM.CHAR char (16 BIT(2 BYTE)) REPRESENTA UN CARACTER, AUNQUE EN REALIDAD ES UN ENTERO

SYSTEM.STRING string

SYSTEM.OBJECT object

SYSTEM.DATETIME

SYSTEM.SBYTE sbyte (-128 a 127)

SYSTEM.UINT16 ushort (0 a 65.535)

SYSTEM.UIN32 uint (0 a 4.294.967.295)

SYSTEM.UINT32 ulong (0 a 18.446.744.073.709.551.615)

sumar uno: ++Numero (una variante de +=Numero)

antes de la suma: Numero++

restar uno: --Numero (una variante de -=Numero)

antes de la resta: Numero--

tambien aplican para los operadores \*, \ y %

OJO. LOS OPERADORES + Y - TAMBIEN SE UTILIZAN PARA EL MANEJO DE CADENAS, ASI QUE SE RECOMIENDA UTILIZAR LOS COMPUESTOS ++ Y -- PARA LOS CASOS DE SUMAR O RESTAR UNO A UNA VARIABLE, EN VEZ DE V+=1 O V-=1

OPERADORES LOGICOS (\*\*LOS DEMAS OPERADORES SON LOS MISMOS)

OPERADOR Y: &&

OPERADOR O: ||

OPERADOR NEGACION: !

OPERADOR IGUAL: ==

OPERADOR DISTINTO: !=

NUMEROS ALEATORIOS: SYSTEM.RANDOM ENTRE 0 E INT32.

-------------------------------------------------------------------------------------------------------------

LA DECLARACION DE VARIABLES NO REQUIERE DE LA CREACION DE UN NUEVA INSTANCIA CON \*\*\* New \*\*\*, AL ONTRARIO DE OTRO TIPO DE OBJETOS COMO LOS ARRAYS Y LAS CLASES LAS CUALES POR SER OBJETOS POR REFERENCIA REQUIEREN INSTANCIARSE

--------------------------------------------------------------------------------------------------------------

Random r=new Random();

Console.WriteLine( r.Next ());

byte bytB = 255;

sbyte sbytB = -128;

decimal decD = (decimal)12345678901234567890123456789.122222; //tambien se puede utilizar el literal D al final

float floMy = (float)123.12;//lo toma como double y no lo convierte implicitamente, por eso la conversion

double douMy = 123.12;

decimal decMy = (decimal)123.12;//lo toma como double y no lo convierte implicitamente, por eso la conversion

int intI = -123456789;

char chaMy = 'd';

string strMy = "cadena";

object objMy = "acepta cualquier tipo de dato, en este caso cadena";

decimal decD1;

decD1=(decimal)125.25\*12546;

DateTime dat = new DateTime();

dat = DateTime.Parse("11/12/2011 00:00:00");

dat = Convert.ToDateTime("11/12/2011 00:00:00");

dat = DateTime.Now;

DECLARACION MULTIPLE

int a = 1, b = 2; short c = 3, d = 10; long e = 123456789;

CONSULTAR LIMITES DE LOS TIPOS

int A;

A = int.MaxValue;

A = int.MinValue;

double B = 0;

B = double.MaxValue;

B = double.MinValue;

ERROR POR PERDIDA DE DATOS, PARA FORZAR AL COMPILADOR UTILIZAR UNCHECKED

short shoX = 1234567890; //\*\*\* COMENTADO PARA EFECTOS DE COMPILACION

short shoX1 = unchecked((short)1234567890);

FUNCIONES DE CONVERSION, NO ES UNA FUNCION NATIVA DE C#, SINO DEL NET F. TAMBIEN SE PUEDE UTILIZAR LA FUNCION NATIVA INT.PARSE()

intI = Convert.ToInt16(decD); //\*\*\* COMENTADO PARA EFECTOS DE COMPILACION

DEJAR QUE EL COMPILADOR INFIERA EL TIPO DE DATO. SIEMPRE SE DEBEN INICIALIZAR. INFIERE QUE ES UN INT32 O INT. VAR ESTA RELACIONADA CON LA DECLARACION DE LAS CLASES ANONIMAS, PARA QUE EL COMPILADOR INFIERA EL TIPO. TAMBIEN ES UTIL CUANDO SOBRECARGAMOS METODOS Y NO SABEMOS QUE TIPO DE DATO NOS VA A REGRESAR.

var varMy = 255;

GENERA ERROR AL QUERERLE PASAR UN LONG

varMy = 123456789012345789; //\*\*\* COMENTADO PARA EFECTOS DE COMPILACION

INFIERE QUE ES UN ARRAY, SE DEBE INICIALIZAR PARA QUE EL COMPILADOR INFIERA EL TIPO

var declaracion = new { Nombre = "Carlos", Edad = 27 };

CADENAS. STRING TIENE UN COMPORTAMIENTO LENTO PORQUE CUANDO SE MODIFICA UN STRING ELIMINA AL PRIMERO Y SE CREA UNO NUEVO. POR ELLO, CUANDO SE REALIZAN MUCHAS OPERACIONES CON CADENAS ES MEJOR STRINGBUILDER

string strS1="hola";

strS1 = strS1 + " mundo";

STRINGBUILDER

StringBuilder sb = new StringBuilder();

sb.Append("Hola, ");

sb.Append("mundo");

VARIABLES QUE ADMITEN VALORES NULL: NULL ES SI ES UNA REFERENCIA, SOLO SE ADMITE COMPROBAR SI CIERTO VALOR ESTA ASIGNADO (VALOR==NULL) PERO NO ASIGNARLE UN NULL, A MENOS QUE SE LE PONGA EL ?, LA VARIABLE SIGUIENTE TENDRA UN NULL POR DEFECTO. NO SE LE PUEDE ASIGNAR UNA VARIABLE QUE ADMITE NULL A OTRA QUE NO LO ADMITE

int? number = null;

ESTAS VARIABLE NULABLES CONTIENEN METODOS UTILIES

if (! number.HasValue) //indica si la variable contiene algun valor

### Constantes con nombre

ESTE TIPO DE DATOS SE DEBEN INICIALIZAR, CONTRARIAMENTE A LOS ANTERIORES QUE A FIN DE CUENTAS ERAN CONSTANTES LITERALES (NUMERICAN O NO) Y SE PODIAN DECLARAR SIN INICIALIZAR.

const int intMyEntero=1234565;

### Ambito

PUBLIC (VB PUBLIC) ACCESO NO RESTRINGIDO.

PRIVATE (VB PRIVATE) ACCESO LIMITADO AL TIPO CONTENEDOR.

INTERNAL (VB FRIEND) ACCESO LIMITADO AL PROYECTO ACTUAL.

PROTECTED (VB PROTECTED) ACCESO LIMITADO A LA CLASE CONTENEDORA O A LOS TIPOS DERIVADOS DE ESTA CLASE.

PROTECTED INTERNAL (PROTECTED FRIEND) ACCESO LIMITADO AL PROYECTO ACTUAL O A LOS TIPOS DERIVADOS DE LA CLASE CONTENEDORA.

STATIC (VB SHARED) UTILIZA LOS OBJETOS SIN NECESIDAD DE CREAR INSTANCIAS NUEVAS PORQUE SE COMPARTE LA MISMA INSTANCIA EN MEMORIA, NO SE REQUIEREN COPIAS DE LA INSTANCIA

ESPACIOS DE NOMBRES PUBLIC IMPLICITA Y DECLARADA, NO SE ADMITE OTRA DECLARACION

OBJETOS EN CLASES PUEDEN DECLARARSE CON LOS 5 TIPOS DE ACCESIBILIDAD, Y POR DEFECTO PRIVATE

OBJETOS EN ESTRUCTURAS PUEDEN SER PUBLIC, PRIVATE O INTERNAL, Y POR DEFECTO SON PRIVATE

OBJETOS EN INTERFACES Y ENUMERACIONES SON PUBLIC Y NO PERMITEN DECLARACION

OBJETOS EN PROCEDIMIENTOS SON PRIVATE IMPLICITA Y NO ADMITEN OTRA DECLARACION

SI NO SE DECLARA UN MODIFICADOR DE ACCESIBILIDAD (PUBLIC, PRIVATE...) SE APLICARA POR DEFAULT DEPENDIENDO DEL AMBITO EN QUE SE HAN CREADO LOS OBJETOS.

SI EXISTE MAS DE UN ESPACIO DE NOMBRE EN EL PROYECTO, Y QUEREMOS UTILIZAR OBJETOS DE UNO EN OTRO SE UTILIZA LA PALABRA "USING" SEGUIDA DEL ESPACIO DE NOMBRES, PROPIOS O DE .NET

ÁMBITO DE BLOQUE: DISPONIBLE ÚNICAMENTE EN EL BLOQUE DE CÓDIGO EN EL QUE SE HA DECLARADO.

ÁMBITO DE PROCEDIMIENTO: DISPONIBLE ÚNICAMENTE DENTRO DEL PROCEDIMIENTO, (FUNCIÓN O PROPIEDAD), EN EL QUE SE HA DECLARADO.

ÁMBITO DE MÓDULO: DISPONIBLE EN TODO EL CÓDIGO DE LA CLASE O LA ESTRUCTURA DONDE SE HA DECLARADO.

ÁMBITO DE ESPACIO DE NOMBRES: DISPONIBLE EN TODO EL CÓDIGO DEL ESPACIO DE NOMBRES.

### Tipos por valor y por referencia

DATOS POR VALOR: SON DATOS QUE CONTIENEN UNA PORCION DE LA MEMORIA PARA CONTENERLOS, POR LO CUAL CADA UNO CONTIENE SU PROPIO ESPACIO \*\*\*SE TOMAN DE LA PILA- STACK\*\*\* SE ASEMEJA A UNA FORMACION VERTICAL APILADA EN LA QUE LA REFERENCIA A LA ULTIMA VARIABLE LLAMADA PASA AL INICIO

int i = 42;

int ii = i;

i++;

TANTO I COMO II OCNTIENEN SU PROPIA PORCION DE MEMORIA, POR LO CUAL AUMENTAR I NO TIENE IMPLICACIONES EN II, A PESAR DE QUE INICIALMENTE FUE UNA COPIA DE I

Console.WriteLine(i + " " + ii);

DATOS POR REFERENCIA: SON DATOS QUE CONTIENEN UNA DIRECCION DE LA MEMORIA QUE QUE CONTIENE EL OBJETO AL QUE ESTAN INSTANCIANDO. UN EJEMPLO DE ESTOS SON LAS CLASES Y TODA SU FUNCIONALIDAD QUE IMPLEMENTAN ES DEBIDO A ESTA ARACTERISTICA. \*\*\*SE TOMAN DEL MONTON- HEAP\*\*\* SE ASEMEJA A UNA FORMACION HORIZONTAL

claDatos miC1=new claDatos();

claDatos miC2=miC1;

AQUI TANTO MIC1 Y 2 CONTIENEN LA DIRECCION DE UN OBJETO UNICO QUE ES CLADATOS.

CUANDO PASAMOS ARGUMENTOS A UN PROCEDIMIENTO POR DEFAULT SON POR VALOR, ES DECIR, NUNCA SE MODIFICARA EL VALOR ORIGINAL, AUNQUE LOS PARAMETROS SEAN TIPOS POR REFERENCIA COMO POR EJEMPLO UNA INSTANCIA DE UNA CLASE, ETC.

int v1 = 5;

Console.Write(v1);

claDatos.Aumenta(ref v1);

Console.Write(v1);

A MENOS QUE SE ANTEPONGA REF EN LAS DOS INSTRUCCIONES, AL QUE LLAMA Y LA QUE TOMA EL ARGUMENTO, CON LO CUAL LE QUEDA CLARO AL PROGRAMADOR QUE LOS VALORES VAN A CAMBIAR

Console.Read();

OJO: OBJECT ES UNA CLASE QUE PUEDE APUNTAR A VARIABLE SPOR VALOR Y POR REFERENCIA. VER BOXING Y UNBOXING

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ESTE APARTADO SE APOYA EN LA CLASE SIGUIENTE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class claDatos

{

public static void Aumenta(ref int a)

{

a++;

}

}

### Datos a pantalla

Console.Read; //lee el siguiente caracter

Console.ReadLine; //lee la siguiente linea

Console.Write; //escribe el siguiente caracter

Console.WriteLine; //escribe la siguiente linea

Console.WriteLine("inserte el nombre");

string strNom = Console.ReadLine();

Console.WriteLine("inserte el apellido");

string strAp = Console.ReadLine();

Console.WriteLine("Bienvenido " + strNom + " " + strAp);

Console.Read();

int intNum = 10;

Console.WriteLine(--intNum);

Console.ReadLine() ;

MANDAR MENSAJES BOX A DOS O MAS LINEAS

MessageBox.Show(CalcularComision(120).ToString() + Environment.NewLine + CalcularComision(120, 3).ToString());

PARAMETROS PARA PASAR VALORES A CONSOLE

Console.WriteLine("The byte value is {0}.", Variable);

PERSONALIZAR EL MESSAGEBOX

MessageBox.Show("MENSAJE", "TEXT DE VENTANA", MessageBoxButtons.OK, MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Button1);

CACHAR EL RESULTADO DEL DIALOG MESSAGEBOX.SHOW

if (MessageBox.Show("¿Desea continuar?", "Eliminar", MessageBoxButtons.OKCancel, MessageBoxIcon.Warning) == DialogResult.OK)

**ENTORNO WEB**

MANDAR MESSAGEBOX EN ASP.NET

ES NECESARIO AGRAGAR LA REFERENCIA A SYSTEM.WINDOWS.FORMS, ASI COMO EL USING AL MISMO ESPACIO DE NOMBRES. UNA VEZ HECHO ESTO YA SE PUEDE UTILIZAR EL MESSAGE. CON LA MISMA SINTAXIS.

## Arrays

ES UNA COLECCION DE VALORES CARENTE DE ORDEN, VALORES QUE DEBEN SER DEL MISMO TIPO (IMPORTANTE A LA HORA DE VER LAS COLECCIONES) Y SE ACCESA A CADA ELEMENTO A TRAVES DE SU INDICE. TODAS LAS MATRICES HEREDAN DEL TIPO SYSTEM ARRAY EL CUAL IMPLEMENTA LA INTERFACE IENUMERABLE.

LOS ARRAYS SON TIPOS POR REFERENCIA SEA CUAL SEA EL TIPO DE DATOS QUE CONTENGA (LA MAYORÍA DE LAS VECES GUARDAN TIPOS POR VALOR, PERO TAMBIEN PUEDEN ALMACENAR TIPOS POR REFERENCIA), O SEA QUE SOLO CONTIENEN UNA REFERENCIA DE LA DIRECCION DEL VALOR QUE SE ENCUENTRA EN MEMORIA. POR LO TANTO SOLO SE INICIALIZAN CUANDO CREAMOS LA NUEVA INSTANCIA. SI NO SE CREA LA NUEVA INSTANCIA NO ES NECESARIO INICIALIZARLA

OJO: SE PUEDEN CREAR CLASES QUE IMPLEMENTEN LA INTERFACE IENUMERABLE, PARA CREAR UN ENUMERADOR

### formas de declarar, inicializar y recorrer:

//1.- DECLARANDO, LUEGO INICIALIZANDO Y DESPUES ASIGNANDO

String[] Camisas;

Camisas = new String[2];

//Camisas1[0] = "AZUL";

//Camisas1[1] = "BLANCO";

//2.- DECLARAR E INICIALIZAR DE UNA SOLA VEZ, DESPUES SE LE ASIGNAN ELEMENTOS

String[] Camisas1 = new String[2];

Camisas1[0] = "AZUL";

Camisas1[1] = "BLANCO";

//3.- DECLARAR, INICIALIZAR Y ASIGNAR VALORES DE UNA SOLA VEZ. LA LONGITUD DEL ARRAY NO SE DECLARA

//YA QUE LA LONGITUD LA TOMA DE LA ASIGNACION. ABAJO ARRAY DE DOS DIMENSIONES

string[] nombres = { "Pepe", "Juan", "Luisa","Pepe", "Jaime" };

string[,] nombres1 = { { "Juan", "Pepe" }, { "Ana", "Eva" } };

//RECORRER UNA ARRAY. CHECAR EL LENGTH -2 EN CASO DE BIDIMENSIONALES

for (int i = 0; i < nombres1.Length - 2; i++)

Console.WriteLine(nombres1[0 + i, 0] + " " + nombres1[i, 1]);

//OJO, RECORRER TODOS LOS ESTRING EN EL ARRAY STRING, no requiere saber su tamaño

foreach (String j in nombres)

Console.WriteLine(j);

### Matrices con tamaño dinamico

Int size=int.parse(Console.Readline());

Int[] pins=new int[size];

### Matrices con tipos anónimos

checar que tiene una forma peculiar de declaracion y asignacion, sobretodo la posicion de corchete. el compilador determinara el tipo de datos

var arrAnonimo = new[] { "a", "b", "c" };

//Array de arrays

//var arrArr = new[] { new{ nombre="a", eded=1 }, new{ nombre="b", edad=2 }, new{ nombre="c", edad=3 }};

### Redimensionar y copiar

REDIMENSIONAR, SOLO SE DECLARA NUEVAMENTE

nombres1 = new String[8, 8];

COPIA DE UN ARRAY \*\*\*ESTE METODO SIRVE PARA HACER UNA COPIA REAL DE LOS ARRAYS, RECORDEMOS QUE SON DATOS POR REFERENCIA ??? CORROBORAR ESTO POR EJEMPLO PARA COPIAR EL ARRAY NOMBRES, DEBE DECLARAR LA COPIA DEL MISMO TAMAÑO UTILIZANDO LA FUNCION COPY INDICANDO EL INDICE DESDE EL CUAL SE QUIERE COPIAR, EN ESTE CASO EL PRIMERO QUE ES CERO PARA COMPROBAR LA COPIA

string[] nombresCopy = new string[nombres.Length];

nombres.CopyTo(nombresCopy, 0);

for (int k = 0; k < nombresCopy.Length; k++)

Console.WriteLine(nombresCopy[k]);

Console.Read();

OTRA FORMA ES CLONAR LA MATRIZ, SE HACE UN CAST A LA MATRIZ ORIGEN

string[] nombresCopy1 = (string[])nombres.Clone();

SE PUEDEN CREAR ARRAYS DE CLASES, ESTRUCTURAS Y ENUMERACIONES, POR EJEMPLO

claParaArray[] miArr;

### Arrays de objetos

SE TIENE UN PROCEDIMIENTO DONDE SE CREA UN ARRAY DE TIPO CD, QUE ES LA CLASE QUE SE HA CREADO

CD[] cdLibrary = new CD[20];

// Populate the CD library with CD objects.

for (int i=0; i<20; i++)

{

cdLibrary[i] = new CD();

}

// Assign details to the first album.

cdLibrary[0].Album = "See";

cdLibrary[0].Artist = "The Sharp Band";

cdLibrary[0].Rating = 10;

### Funciones para arrays

JOIN: UNE LOS ELEMENTOS DEL VECTOR

//string strNombres = Join(nombres, "/");

SPLIT: LO CONTRARIO DE JOIN

ARRAY.INDEXOFF BUSCA EN EL ARRAY EL INDEX DE LA COINCIDENCIA

ARRAY.LASTINDEXOFF BUSCA EL INDEX DE LA ULTIMA APARICION DE LA COINCIDENCIA

int intIndexArray = Array.IndexOf(nombres, "Pepe");

int intIndexArrayLAst = Array.LastIndexOf(nombres, "Pepe");

Console.WriteLine(intIndexArray + intIndexArrayLAst );

Console.Read();

FILTER: FILTRA EN EL ARRAY Y DEJA O IGNORA LAS COINCIDENCIAS SEGUN EL TRUE O FALSE

//string strFilArray = Filter(nombres, "P", true);

ARRAY DE PARAMETROS VER EN EL APARTADO DE CLASES Y ESTRUCTURAS

ORDEN DEL ARRAY

Array.Sort(nombres );

Array.Reverse(nombres );

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ESTE APARTADO SE APOYA EN LA CLASE SIGUIENTE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class claParaArray

{

//implementacion

}

class CD

{

private string album;

private string artist;

private int rating;

public string Album

{

get {return album;}

set {album = value;}

}

public string Artist

{

get {return artist;}

set {artist = value;}

}

public int Rating

{

get {return rating;}

set {rating = value;}

}

}

## Estructuras de control

try

{

string strCadena = "hola";

int intNumero = 100;

//Boolean bolBoleano = false;

### Sentencias if

RECORDAR QUE AQUI APLICAN LOS OPERADORES LOGICOS QUE SON EXCLUSIVOS DE C#

if (strCadena == "hola")

Console.WriteLine(strCadena);

//if con opciones

if (strCadena == "hola")

{

Console.WriteLine(strCadena);

}

else if (strCadena == "anonimo")

{

Console.WriteLine(strCadena + " anonimo");

}

else

{

Console.WriteLine("no tiene acceso al sistema");

}

//if anidada

if (intNumero < 1000)

{

if (intNumero > 50)

{

Console.WriteLine("cantidad correcta !!!!");

}

}

int intA = 4, intB = 5, intC = 9;

OJO AL APLICAR ESTOS OPERADORES. CORTO CIRCUITO OPERADORES LOGICOS. EN EL CASO DE LOS OPERADORES LOGICOS OPERA && Y || OPERA EL CORTO CIRCUITO:

ES DECIR:SI P ES FALSO, P && Q ES FALSO INDEPENDIENTE DEL VALOR DE Q. DE IGUAL MANERA SI P ES VERDADERO, P || Q ES VERDADERO INDEPENDIENTEMENTE DEL VALOR DE Q. POR ELLO SE RECOMIENDA ANALIZAR BIEN EL CODIGO Y PONER LAS ASIGNACIONES MAS SENCILLAS A LA IZQUIERDA, PARA QUE, DE AMERITARLO, YA NO SE TENGAN QUE EVALUAR LAS COMPLICADAS DE LA DERECHA. VER NOTAS

if (intC > intA && intB > intA)

Console.WriteLine("Logico and ambos son mayores");

//or

if (intC > intA || intB > intA)

Console.WriteLine("Logico or alguno es mayor");

//not e igual

if (!(intC == intA))

Console.WriteLine("Logico not no es igual");

//distinto

if (intC != intA)

Console.WriteLine("Logico distinto si es distinto");

ME EXPLICO. TODO EL MUNDO SABE QUE PARA QUE UNA CONDICIÓN AND PROPORCIONE UN VALOR POSITIVO (TRUE) AMBAS CONDICIONES QUE SE COMPRUEBAN DEBEN SER POSITIVAS (TRUE). POR ELLO, CUANDO C# ENCUENTRA UNA CONDICIÓN AND, Y VERIFICA QUE LA PRIMERA DE LAS DOS CONDICIONES QUE SE COMPARAN ES FALSA, AUTOMÁTICAMENTE DEJA DE COMPROBAR LA QUE FALTA, PUESTO QUE AUNQUE FUESE CIERTA EL RESULTADO SERÁ FALSO AL SERLO YA LA PRIMERA Y, ENTONCES, ¿PARA QUÉ SEGUIR PERDIENDO EL TIEMPO?.

LO MISMO OCURRE CON LOS OTROS TIPOS DE OPERADORES BOOLEANOS. POR EJEMPLO, EN UNA EXPRESIÓN OR, SI LA PRIMERA CONDICIÓN ES CIERTA YA NO SE SIGUE COMPROBANDO EL RESTO PUESTO QUE EL RESULTADO YA SE SABE CON LA PRIMERA (VERDADERO OR LO-QUE-SEA = VERDADERO).

TODO ESTO, APARTE DE LAS OBVIAS IMPLICACIONES DE RENDIMIENTO QUE PROPORCIONA, SIRVE TAMBIÉN PARA SALVAR CIERTAS SITUACIONES CON EXPRESIONES ELEGANTES. POR EJEMPLO... IMAGINEMOS QUE EL RESULTADO DE LLAMAR A UNA FUNCIÓN ES UNA CADENA (STRING). SI EL PROCESO QUE GENERA DICHA CADENA FRACASA O (POR CUALQUIER OTRO MOTIVO) NO PUEDE OFRECER UN RESULTADO RAZONABLE, EL MÉTODO DEVUELVE UN NULO EN LUGAR DE UNA CADENA. NOSOTROS QUEREMOS COMPROBAR QUE EL RESULTADO DE LLAMAR AL MÉTODO NO DEVUELVA UN NULO NI TAMPOCO UNA CADENA VACÍA, ASÍ QUE ESCRIBIMOS:

string s = MiMetodo();

if (s.Trim() == "" || s == null)

{

....//Lo que sea

}

ESTA EXPRESIÓN ES ERRÓNEA Y FALLARÁ CUANDO EL MÉTODO DEVUELVA UN NULO, AUNQUE PARECERÁ CORRECTA EN EL RESTO DE LOS CASOS

(A PRIORI LOS MÁS COMUNES). ¿POR QUÉ? PUES PORQUE SI 'S' ES UN VALOR NULO, EN CUANTO SE INTENTE LLAMAR A SU MÉTODO TRIM() EN LA PRIMERA CONDICIÓN, SE PRODUCIRÁ UN ERROR PUESTO QUE NO EXISTE TAL MÉTODO PARA EL OBJETO NULO.

LA FORMA CORRECTA DE ESCRIBIR ESTA COMPROBACIÓN ES JUSTO AL REVÉS:

string s = MiMetodo();

if (s == null || s.Trim() == "")

{

....//Lo que sea

}

DE ESTE MODO SI 'S' ES NULO LA PRIMERA CONDICIÓN ES CIERTA Y POR LO TANTO NO ES NECESARIO COMPROBAR LA SEUGDA (HAY UN OR), Y NUNCA SE PRODUCE EL ERROR POR INTENTAR LLAMAR A TRIM(). SI POR EL CONTRARIO LA CADENA ES VÁLIDA LA PRIMERA CONDICIÓN ES FALSA Y SE DEBE COMPROBAR LA SEGUNDA, QUE ES LO QUE NECESITAMOS. LA COMPROBACIÓN FUNCIONES PERFECTAMENTE EN TODOS LOS CASOS GRACIAS AL CORTOCIRCUITO DE EXPRESIONES LÓGICAS.

### Sentencia switch

LAS SETENCIAS SWITCH EVALUAN SOLO UNA VEZ LA SENTENCIA DE CONTROL, LO CONTRARIO QUE IF EN EL QUE SE DEBE REPETIR LAS VECES QUE SE REQUIERA EVALUARLA, PERO OJO, SOLO FUNCIONAN PARA DATOS PRIMITIVOS COMO INT O STRING; SOLO EVALUAN CONSTANTES, SI SE REQUIERE CALCULOS SERA NECESARIO UTILIZAR

switch (strCadena)

{

case "hola":

Console.WriteLine("bienvenido al sistema");

break;

default:

Console.WriteLine("usuario no autenticado");

break;

}

### Sentencias for y for each

(DECLARACION, PRUEBA, ACCION)

string[] arrNombres = { "A", "B", "C" };

for (int i = 0; i < arrNombres.Length; i++)

{

Console.WriteLine("desde for {0} ", arrNombres[i + 100]); //le puse + 100 para que genere error

}

Console.Read();

foreach (string StrX in arrNombres)

{

Console.WriteLine("desde foreach {0} ", StrX);

}

Console.Read();

### Sentencias while y do

LA SENTENCIA WHILE EVALUA UNA CONDICION BOLEANA QUE VA ENTRE PARENTESIS

int inti = 0;

while (inti < 2)

{

Console.WriteLine("desde while {0} ", inti);

inti++;

}

Console.Read();

LA VENTAJA DE LAS SENTENCIAS DO ES QUE POR EJEMPLO WHILE Y FOR SOLO SE EJECUTAN SI LA CONDICIÓN BOLEANA ES VERDADERA, DE LO CONTRARIO NO. Y DO SE EJECUTA POR LO MENOS UNA VEZ AUNQUE LA CONDICIÓN BOLEANA SEA FALSA

int intI = new int();

intI = 3;

do

{

Console.WriteLine(intI.ToString());

intI++;

}

while (intI < 3);

}

### Sentencia try, catch, finally

SI QUEREMOS CAPTURAR UN ERROR ESPECIFICO, POR EJEMPLO UNO DE LECTURA O ESCRITURA DE ARCHIVOS CATCH(SYSTEM.IO.IOEXCEPTION EX), DE FORMATO DE NUMEROS FORMATEXCEPTION, DE DESBORADAMIENTO DE LA VARIABLE OVERFLOWEXCEPTION, ETC. PARA LO CUAL SE PUEDEN ANIDAR TIPOS DE EXCEPCIONES.SI NO QUEREMOS QUE SE PARE LA EJECUCION SOLO PONEMOS CATCH TODOS LOS ERRORES, UTILIZAMOS LA CLASE GENERICA, EXCEPCION ES LA CLASE BASE DE LAS EXCEPCIONES: SYSTEM.EXCEPTION EX. ASI MISMO, COMO ES LA EXCEPCION POR DEFECTO PODRIA OMITIRSE Y AUN ASI SE EJECUTARIA, CON SOLO UTILIZAR CATCH Y LAS INSTRUCCIONES DE MANEJO DEL ERROR, PERO CON ESTA FORMA NO PODEMOS OBTENER INFORMACION SOBRE EL ERROR OCURRIDO. UNA PRACTICA BUENA ES PONER LOS CATCH ESPECIFICOS Y AL FINAL EL GENERICO, QUE ES EL QUE ATRAPA "TODAS LAS DEMAS".

OJO: SE MENCIONA QUE LOS TRY CATCH SE DEBEN PONER EN LA CAPA DE PRESENTACION. SI SE USAN DE MANERA INDISCRIMINADA LOS CATCH SE PUEDEN OCULTAR UNOS A OTROS Y MANDAR COMPORTAMIENTO NO DESEADO

catch (System.Exception ex)

{

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

Console.Read();

//ANIDADNDO LOS CATCH

//catch (DivideByZeroException ex){} captura una división entre cero

//catch (FormatException ex){} captura errores en el tipo de datos proporcionado

//\*\*\*\*GENERAR ERRORES MANUALMENTE\*\*\*\* REALMENTE NO ES ESTA LA SECCION ADECUADA PARA ESTE CODIGO

//EL PUNTO INTERESANTE ES QUE EL FRAMEWORK YA TIENE EXCEPCIONES PREDEFINIDAS QUE SE PUEDEN VALIDAR, O EN //OTRO CASO, SE PUEDEN GENERAR MANUALMENTE COMO EN EL CODIGO SIGUIENTE:

//SE PODRIA DECIR QUE EL SIGUIENTE ES COMO UN GENERICO, PERO EXISTEN MAS TIPOS

//throw new ArgumentException("EXISTE UN ERROR");

//throw new ArgumentNullException("");

//throw new InvalidOperationException("");

//THROW NEW DivideByZeroException();

}

finally

{

//CON UNA INSTRUCCION FINALLY SE ASEGURA DE QUE SEA CUAL SEA EL FLUJO QUE SIGA EL CODIGO SIEMPRE SE

//EJECUTARA LO QUE HAYA EN ESTE BLOQUE, POR EJEMPLO MUY UTIL PARA EL CIERRE DE CONEXIONES O LA DESTRU

//CCION MANUAL DE CIERTOS OBJETOS.

}

EXISTEN OCASIONES EN QUE LA FINALLY TIENE INCONVENIENTES A LA HORA DE LIBERAR ALGUN RECURSO U OBJETO PAG 375 LIBRO PARA ESO EXISTE LA SENTENCIA USING (QUE NO ES IGUAL A LA QUE LLAMA A ESPACIOS DE NOMBRES), LA CUAL SE COMPORTA COMO UN AMBITO DE LA APLICACION POR SI MISMO, POR LO CUAL CUALQUIER OBJETO O REFERENCIA FUERA DE ESTE AMBITO QUEDA INACCESIBLE A LA APLICACIÓN, Y CUALQUIER OBJETO INICIALIZADO CON USING AL TERMINO DE LA MISMA SE CERRARA. POR EJEMPLO:

USING(TEXTREADER READER=NEW STREAMREADER(FILENAME))

{

STRING LINE;

WHILE ((LINE=READER.READLINE())!=NULL)

{

CONSOLE.WRITELINE(LINE);

}

}

NOTAR QUE SOLO SE UTILIZA USING PARA VARIABLES QUE IMPLEMENTEN LA INTERFAZ IDISPOSABLE, EN ESTE CASO TEXTREADER, UNA VEZ

QUE EL READER ALCANCE UN VALOR NULL, SE LIBERARA EL RECURSO Y NO SE PUEDEN ACCESAR A SUS OBJETOS DESDE FUERA DEL BLOQUE.

### Validación

ERROR PROVIDER

EL ERROR PROVIDER SOLO FUNCIONA EN EL CONTEXTO DE WINDOWS FORMS. ES MAS FACIL UTILIZAR UN PROCEDIMIENTO EN DONDE SE DESENCADENE LA VALIDACION DE TODOS LOS ERROR PROVIDER, PORQUE ASI LA INTERFACE ES MAS FUIDA PARA EL USUARIO.

{

//PRIMERO SE LIMPIA LA INSTANCIA PARA QUE NO PERSISTA EL MENSAJE

errorProvider1.Dispose();

//VALIDA QUE EL TEXTBOX TENGA UN VALOR

If(TextBox1.Text.Length==0)

{

errorProvider1.SetError(TextBox1, “Debe proporcionar un valor”);

}

}

PARA EL AMBITO WEB SE COMPLICA UN TANTO, YA QUE SE MANEJAN DIFERENTE LOS MENSAJES DE ERROR. LA OPCION MAS ELEGANTE SERIA UTILIZAR LOS CONTROLES AJAX O LOS VALIDATORS INCLUIDOS, OTRA OPCION SERIA MANEJAR UNA VALIDACION MAS PROFUNDA EN LOS SET DE LAS CLASES, PERO A VECES SE SUELE COMPLICAR POR LA DECLARACION DE TIPOS.

## Enumeraciones y estructuras

### Enumeraciones

SON TIPOS POR VALOR "A LA MEDIDA", ES DECIR: LAS PODEMOS UTILIZAR PARA REPRESENTAR COSAS REALES DE MANERA MAS PROFESIONAL Y ROBUSTA, POR EJEMPLO, SI QUIEREAMOS REPRESENTAR LOS MESES DEL AÑO POR MEDIO DE NÚMEROS ENTEROS SIN DUDA SERVIRIA, PERO ESTARIAN PROPENSOS A ASIGNARLES CUALQUIER OTRO ENTERO QUE NO TUVIERA NADA QUE VER CON LA IDEA ORIGINAL. EN CAMBIO CON UNA ENUMERACION SE PUEDE REPRESENTAR LO MISMO PERO DE MANERA EFICIENTE.

enum enumEstaciones

{

Primavera,

Verano,

Otoño,

Invierno

};

OTRO EJEMPLO QUE RELACIONA LOS TIPOS CON CONSTANTES NUMERICAS DEFINIDAS MANUALMENTE, LAS DE ARRIBA ESTAN DEFINIDAS POR DEFAULT COMENZANDO POR CERO

[Flags()]

enum Colores : short

{

Rojo = 1,

Verde = 2,

Azul = 4

};

struct estE

{

LOS CAMPOS NO SE PUEDEN INICIALIZAR A MENOS QUE SE DECLAREN COMO CONSTANTES O ESTÁTICOS. COMO NO SE PUEDEN INICIALIZAR AL MISMO TIEMPO, EL VALOR QUE CONTIENEN SON, EN CASO DE NUMEROS CEROS Y EN CASO DE CADENAS VALORES NULL (CUANDO NO SE CREA UN CONSTRUCTOR PROPIO).

private int x;

public int proX

{

get {

return x;

}

set {

x = value;

}

}

private int y;

public int proY

{

get {

return y;

}

set {

y = value;

}

}

public int funSuma(int a, int b)

{

return a + b;

}

public estE(int j)

{

//EL CONSTRUCTOR DEBE INICIALIZAR LOS CAMPOS PORQUE NO SE INICIALIZAN SOLOS

this.x = j;

this.y = j;

}

}

class Program

{

static void Main(string[] args)

{

ANTES DE LEER EL VALOR DE UNA VARIABLE TIPO ENUMERACION LA TENEMOS QUE INICIALIZAR, COMO CUALQUIER OTRA

enumEstaciones enumE = enumEstaciones.Primavera;

Console.WriteLine(enumE.ToString());

//SE LE PUEDE ASIGNAR UN NULL POR SER TIPO POR VALOR

enumEstaciones? enumE1 = null;

SE PODRIA UTILIZAR EL NUMERO PERO HACIENDO REFERENCIA AL NOMBRE. LAS ENUMERACIONES SE DECLARAN A NIVEL MODULO.

EN LA DECLARACION SE LE PUSO TIPO SHORT, SI SE OMITE LA DECLARACION DE TIPO LE PONE INT POR DEFAULT LAS OPERACIONES POSIBLES EN LAS ENUMERACIONES SON AND (&) Y OR (|). OR HACE ALGO SIMILAR A LA SUMA, EN ESTE CASO QUE ES A NIVEL BIT.

Colores colorSumado = Colores.Azul | Colores.Rojo;

LA ENUMERACIONES SON UN OBJETO CERRADO EN SI MISMAS, SOLO SE REPRESENTAN CON NOMBRE LOS VALORES QUE COINCIDEN CON EL DE UN MIEMBRO O LA COMBINACION DE LOS MISMOS, DE LO CONTRARIO SOLO SE REPRESENTARA LA PARTE NUMERICA. SI NO SE LE DA UN VALOR NUMERICO A SUS ELEMENTOS, LO TOMAN POR DEFAULT COMENZANDO DE 0.

SI SE UTILIZA EL ATRIBUTO [Flags()] ENTONCES YA NO SE UTILIZARAN LOS NUMEROS, SINO QUE PERSISTIRAN LOS NOMBRES. POR EJEMPLO colorSumado DARA ROJO AZUL, EN VES DE 5

string colorANombre = Colores.Azul.ToString();

SI SE QUIERE CREAR UNA VARIABLE TIPO ENUM, SE DEBE HACER UN CAST PARA COMPROBAR QUE EL ELEMENTO EXISTE, SOLO QUE ESTE cast SOLO EVALUA VALORES DIRECTOS DE LOS ELEMENTOS, NO LAS COMBINACIONES.

EVALUA SI Gris ESTA DEFINIDO EN LA ENUMERACION COLORES, SI NO ES ASI, LE PONE AZUL.

Colores Gris = (Colores)20;

if (Enum.IsDefined(typeof(Colores), Gris) == false)

{

Gris = Colores.Azul;

}

### Estructuras

EL OTRO TIPO POR VALOR SON LAS ESTRUCTURAS SE UTILIZAN CUANDO SE REQUIERE DE RAPIDEZ Y QUE NO SEA UN CONJUNTO DE DATOS MUY GRANDE, ENTONCES LAS ESTRUCTURAS SE COMPORTAN COMO LAS CLASES PERO CON MUCHO MAYOR RAPIDEZ. UN EJEMPLO DE CUANDO UTILIZAR ESTRUCTURAS ES EL EJERCICIO DE THREADING EN WPF (MAXIMIZAR EL RENDIMIENTO DE UNA APLICACION PARA QUE NO SE PASME EN UN PROCESO LARGO), VER PROYECTO O LA PAGINA 639 DEL LIBRO C#

estE P = new estE(5);

P.proX = 100;

P.proY = 75;

//MOSTRARA LOS DATOS DEFINIDOS MEDIANTE EL CONSTRUCTOR

Console.WriteLine(P.proX + " " + P.proY );

Console.Read();

}

}

STRUCT STRCPUNTO

{

SE UTILIZAN CUANDO SE REQUIERE DE RAPIDEZ Y QUE NO SEA UN CONJUNTO DE DATOS MUY GRANDE, ENTONCES LAS ESTRUCTURAS SE COMPORTAN COMO LAS CLASES PERO CON MUCHO MAYOR RAPIDEZ LAS ESTRUCTURAS SON LOS TIPOS POR VALOR DE C# Y COMO TAL NO REQUIEREN SER INSTANCIADAS. PUEDEN TENER LA MISMA ESTRUCTURA QUE UNA CLASE, PERO SU COMPORTAMIENTO ES UN TANTO DIFERENTE. SE ALMACENAN EN LA PILA DE MEMORIA, POR LO TANTO ESTRUCTURAS GRANDES CONSUMEN GRAN CANTIDAD DE MEMORIA \*\*SON UTILES POR SU RAPIDEZ DE ACCESO\*\*.

EL COMPILADOR GENERA UN CONSTRUCTOR SIN PARAMETROS, POR LO TANTO SI SE QUIERE DEFINIR UNO DEBERA TENER PARAMETROS AUNQUE

INSTANCIARLO SERIA REDUNDANTE, ESTO SE HARA EN CASO DE QUERER EJECUTAR UN CONSTRUCTOR DEFINIDO POR EL USUARIO)SE DEFINEN A NIVEL DE MODULO, NO PUEDE CREAR UN DESTRUCTOR POR TRATARSE DE TIPOS POR VALOR, NO PUEDEN HEREDAR

NO SE PUEDEN INICIALIZAR AL MISMO TIEMPO, EL VALOR QUE CONTIENEN SON, EN CASO DE NUMEROS CEROS Y EN CASO DE CADENAS VALORES NULL.

PUBLIC INT X;

PUBLIC INT Y;

PUBLIC STRCPUNTO(INT INTA)

{

SOLO ACEPTA ASIGNACIONES DIRECTAS COMO LA SIGUIENTE.SE UTILIZA EL THIS PARA LLAMAR A LOS CAMPOS DE LA ESTRUCTURA, POR EJEMPLO SI LOS PARAMETROS SE LLAMARAN IGUAL ESTOS OCULTARIAN A LOS PRIMEROS, POR ELLO SE UTILIZA ESTE METODO. POR LO GENERAL ESTO OCURRE CUANDO LOS METODOS TRABAJAN DE MANERA INTERNA CON LOS CAMPOS "METODOS DE INSTANCIA"

THIS.X = INTA;

THIS.Y = INTA;

}

}

## Clases e interfaces

### La clase principal Main

CLASES SON TIPOS POR REFERENCIA, por ello se deben instanciar antes de usar, al instanciarla con NEW se crea un espacio en memoria

class Program

{

static void Main(string[] args)

{

claPrimeraClase ObjMiClase = new claPrimeraClase();

Console.WriteLine(ObjMiClase.NombreCompleto("a", "p"));

Console.Read();

ObjMiClase.MiSaludo();

ACCESO AL INDEX DEL ARRAY DE LA CLASE POR MEDIO DEL INDIZADOR

string strN=ObjMiClase [1];

MANDAR UN ARRAY DE PARAMETROS: NOTA QUE SE PUEDEN MANDAR N CANTIDAD DE PARAMETROS SIN NECESIDAD DE CREAR INICIALMENTE UN ARRAY PARA MANDARLO COMO PARAMETRO, YA QUE LA FUNCION ESTATICA AL INCLUIR PARAMS UNICAMENTE REQUIERE QUE SE MANDEN ARGUMENTOS Y LOS TOMA COMO ARRAY

int Num = claPrimeraClase.sumaParams(1, 2, 3, 4);

Console.WriteLine("la suma de la funcion con array de parametros es : {0} " , Num);

Console.Read();

PARA SABER SI UNA CLASE DETERMINADA IMPLEMENTA UNA INTERFAZ

claAnimal myAnim=new claAnimal();

if (myAnim is intfAnimal )

Console.WriteLine("myAnim es del tipo intfAnimal");

else

Console.WriteLine("myAnim no es del tipo intfAnimal");

Console.Read();

SI DETERMINADA INTERFAZ IMPLEMENTA ALGUN PROCEDIMIENTO, EJECUTARLO

object o=null ;

if (o is intfAnimal)

{

intfAnimal unaPrueba = (intfAnimal)o;

unaPrueba.Alimentarse();

}

REFERENCIAR UN METODO EXPLICITO DE INTERFACE, AQUI NO APARECE EL METODO DESPLAZARSE, SOLO EL ALIMENTARSE PORQUE NO ES EXPLICITO

//claAnimal myAnimExplicito = new claAnimal();

//myAnimExplicito.

PARA ACCESAR A LOS METODOS SE HACE ASI: SE CREA UNA INSTANCIA DE LA INTERFACE QUE IMPLEMENTA EL METODO DESEADO Y A TRAVES DE ELLA SE LLAMA AL MISMO

claAnimal myAnimExplicito = new claAnimal();

intfAnimal DespAnimal = myAnimExplicito;

DespAnimal.Desplazar();

UTILIZACION DE LA CLASE PARCIAL, SU UTILIZACION ES NORMAL, COMO CUALQUIER OTRA CLASE

claMiClaseParcial myCP=new claMiClaseParcial() ;

myCP.mensaje();

CLASES ABSTRACTAS, NO SE PUEDEN INSTANCIAR

//absClaseAbstracta myCa = new absClaseAbstracta();

METODOS VIRTUALES: IMPLEMENTA SU METODO VIRTUAL PARA SER SOBRECARGADO

claVirtual miV = new claVirtual();

Console.WriteLine(miV.Nombre());

//devuelve su propio nombre porque sobrecargo el metodo virtual

claDerivada1 miD1 = new claDerivada1();

Console.WriteLine(miD1.Nombre());

//devuelve el nombre de la base porque no sobrecargo el metodo virtual

claDerivada2 miD2 = new claDerivada2();

Console.WriteLine(miD2.Nombre());

Console.Read();

METODO DE EXTENSION : VER QUE ESTAMOS EXTENDIENDO LA "CLASE O TIPO" INT32, PARA QUE DEVUELVA UN NEGATIVO, LA CONDICION ES QUE SEA UNA CLASE ESTATICA EN EL AMBITO DE LA APLICACION, LA CUAL ESTA ARRIBA. NOTA QUE NO NECESITAMOS HEREDAR (PORQUE ADEMAS INT32 ES UNA ESTRUCTURA Y NO SE PUEDE HEREDAR), Y TAMPOCO NECESITAMOS CREAR UNA INSTANCIA, SOLO INVOCAR.

int Ext = 10;

Console.WriteLine (Ext.Negativo());

Console.Read();

O TAMBIEN

Console.WriteLine(CLAExtendida.Negativo(Ext));

LOGICA DE ASIGNACION DE CLASES, EJEMPLO

//claMamifero

//claCaballo: claMamifero

//claBallena: claMamifero

//claCaballo miCab=new claCaballo();

LEGAL PORQUE TODO CABALLO ES MAMIFERO

//claMamifero miMam=miCAb

ILEGAL, NO TODOS LOS MAMIFEROS SON CABALLOS, A MENOS QUE SE UTILICE UN CAS "AS O IS"

//miCab=miMam

LEGAL, PERO SI EN EL CAST MAMIFERO NO ES UN CABALLO DARA UN NULL

//miCab=miMAm as claCaballo

}

}

### Las clase principal

class claPrimeraClase

{

LOS CAMPOS PODRIAN INICIALIZARSE, PERO NORMALMENTE NO SE HACE

private string strApellido;

private string strNombre;

1. **Propiedades**

PARECIDO A UN METODO PERO SIN PARAMETROS Y CON GET Y SET. SE DEFINE DE LECTURA O ESCRITUTA CON ELIMINAR

CUALQUIERA DE SUS ATRIBUTOS GET O SET. TAMBIEN SE PUEDE MODIFICAR EL AMBITO A LOS ATRIBUTOS, POR EJEMPLO PONIENDO

PRIVATE A CUALQUIERA DE LOS DOS. UNA DE LAS CARACTERISTICAS DE LAS PROPIEDADES ES PROPORCIONAR ACCESO CONTROLADO A

LOS CAMPOS DE LA CLASE.

OJO: HAY OCASIONES EN LAS QUE SE INICIALIZAN CAMPOS POR MEDIO DE CONSTRUCTORES, PERO PODRIA SER NECESARIO ESCRIBIR

UN GRAN NUMERO DE CONSTRUCTORES DEPENDIENDO LAS COMBINACIONES DE CAMPOS. UNA OPCION ES UTILIZAR LAS PROPIEDADES Y

DEFINIR CAMPOS CON VALORES POR DEFECTO. PP 403 LIBRO. EJEMPLO:

//private int intLado = 10;

//public int Lado

//{

// set { this.intLado = value; }

//}

VER MAS DETALLE PARA LAS PROPIEDADES INDEXADORAS, QUE SIRVEN PARA ACCESAR AL INDICE DE COLECCIONES

public string Apellido

{

private get

{

return strApellido;

}

set

{

strApellido = value;

}

}

1. **Metodos que devuelven valor**

SE LE DECLARA UN TIPO Y LLEVA RETURN PORQUE DEVUELVEN UN VALOR, COMO LAS FUNCIONES DE VB POR DEFAUL LOS PARAMETROS SON POR VALOR. PARA DECLARAR PARAMETROS POR REFERENCIA EXISTEN DOS: out Y ref METODO CON DOS PARAMETROS DE AMBITO PUBLICO, O SEA QUE ES ACCESIBLE DESDE TODO EL CODIGO.

SI LA CLASE DERIVADA Y LA BASE TIENE IDENTICO NOMBRE DE METODOS, LA DERIVADA OCULTA AL DE LA CLASE BASE EN ESTE CASO EL COMPILADOR ALERTARA DE ESTA OCULTACION

VER MAS DETALLE DE OUT Y REF \*\*

public string NombreCompleto(string Nom, string Ap)

{

return Nom + " " + Ap;

}

**(2.1.) SOBRECARGA DE METODOS y METODOS VIRTUALES:**

SOBRECARGAMOS EL METODO ANTERIOR CON UN PARAMETRO MAS

string NombreCompleto(string Nom, string Ap, string Ap2)

{

return Nom + " " + Ap + " " + Ap2 ;

}

UN METODO QUE ESTA PENSADO PARA SOBRECARGARLO COMO ES EL CASO DE TOSTRING() SE LLAMA METODO VIRTUAL, Y SE HACE VER MEDIANTE EL MODIFICADOR VIRTUAL, POR EJEMPLO

//"public virtual string tostring(){}".

UNA CLASE PROPIA DEBERIA SOBRECARGAR SU PROPIA VERSION DE TOSTRING, VER EL SIGUIENTE EJEMPLO QUE LLAMA AL MÉTODO TOSTRING DE LA CLASE BASE. UTILIZA OVERRIDE PORQUE EN LA CLASE BASE SE DECLARO COMO VIRTUAL EL METODO, NOTAR QUE ES UN METODO CON SIGNATURA IDENTICA, SI OMITIMOS OVERRIDE ENTONCES NO SOBRECARGAMOS EL EMTODO SINO QUE ESTAMOS CREANDO UNO NUEVO Y OCULTANDO EL DE LA BASE

public override string ToString()

{

return base.ToString();

SI QUEREMOS PERSONALIZAR EL MENSAJE DEBERIAMOS:

//return "son el nombre personalizado de miprimeraclase";

}

1. **Metodos que no devuelven valor**

COMO LOS METODOS SUB DE VB. LLEVA LA INSTRUCCION void. NOTA: QUE NO NECESITA DECLARAR UN TIPO DE DATO, UNICAMENTE void

public void MiSaludo()

{

Console.WriteLine("Hola");

}

1. **Array de parámetros**

POR EJEMPLO, EL METODO WRITELINE DE CONSOLE ESTA SOBRECARGADO VARIAS VECES PARA SOPORTAR VARIOS TIPOS DE PARAMETROS, PERO EN CUESTION DE NUMERO DE PARAMETROS SERIA COMPLICADO SOBRECARGARLO PARA 1 O N, POR LO TANTO SE USAN LOS ARRAYS DE PARAMETROS PARA SOPORTAR UNO O VARIOS DE ELLOS. NOTA: TAMBIEN ACEPTA PARAMETROS TIPO OBJECT

public static int sumaParams(params int[] parametros)

{

int total = 0;

for (int i = 0; i < parametros.Length; i++)

total += parametros[i];

return total;

}

OJO: LAS PROPIEDADES SIRVEN PARA CONTROLAR EL ACCESO A CAMPOS QUE CONTIENEN UN SOLO VALOR, PERO UN INDEXADOR SE UTILIZA

PARA PROPORCIONAR ACCESO A ELEMENTOS TIPO ARRAY.

1. **Indizadores**

SON PROPIEDADES QUE HACEN QUE LA CLASE QUE LAS CONTIEN "ACTUEN COMO ARRAY (???)" Y SE PUEDE ACCESAR A LOS OBJETOS QUE CONTIENE A TRAVES DE SU NOMBRE E INDICE. RECIBEN UN PARAMETRO Y SE DECLARA CON (EL NOMBRE DE LA CLASE QUE LO CONTIENE\*\* OPCIONAL) MAS THIS, POR ELLO SE DICE QUE REEMPLAZAN A LAS PROPIEDADES PREDETERMINADAS QUE YA NO EXISTEN EN NET. SON LAS UNICAS PROPIEDADES QUE PERMITEN DEFINIR PARAMETROS E IMPLEMENTAR BLOQUES GET O SET. SE PUEDEN SOBRECARGAR.

POR EJEMPLO, SI ESTA CLASE TUVIERA UN ARRAY DE CADENAS

private string[] arrNombres = new string[] { "uno", "dos", "tres" };

public string this[int index]

{

get

{

return arrNombres [index];

STRING ES EL TIPO DE DATOS DEL ARRAY AL QUE QUEREMOS ACCESAR Y EL INT PORQUE QUEREMOS ACCESAR A SU INDEX EN ESTE EJEMPLO EL INDICE ESTA DADO POR EL MISMO ARRAY

}

}

OTRO EJEMPLO ES UTILIZANDO LA CLASE SORTEDLIST PARA PODER PERSONALIZAR LOS INDICES, Y ASI EMULAR UNA BASE DE DATOS DE DIRECTORIO TELEFONICO, ESTO A MANERA MAS SENCILLA DEL EJEMPLO DEL LIBRO EL CUAL SE REALIZA MEDIANTE ESTRUCTURAS. DECLARAMOS LO SIGUIENTE:

//SortedList Directorio = new SortedList();

//luego sobrecargamos el constructor de la siguiente manera

//public claPrimeraClase()

//{

// Directorio[1] = "a";

// Directorio[20] = "b";

// Directorio[31] = "c";

//}

Y CREAMOS UN METODO

//public object this[int index]

//{

// get

// {

// return Directorio[index];

// }

//}

FINALMENTE EL LA CLASE CON INDEXADOR SE LLAMA DE LA MISMA MANERA DESDE EL METODO APROPIADO. SI QUISIERAMOS QUE DANDO

EL TEXTO REGRESARA EL NUMERO, SOLAMENTE CAMBIARIAMOS "[STRING INDEX]", LA LLAMADA DESDE EL PROCEDIMIENTO "OBJMICLASE["A"].TOSTRING()" Y EL ORDEN EN COMO SE LLENA EL ARRAY "DIRECTORIO["A"] = 1;"

1. **Parametros opcionales**

SI SE MANDA LOS DOS VALORES LOS TOMA, SI SE MANDA SOLO UNO TOMA EL VALOR DEFINIDO POR DEFAULT PARA EL PARAMETRO OPCIONAL. ESTOS SON UTILIES PARA UTILIZAR COM QUE NO ACEPTA SOBRECARGA, PERO SI PARAMETROS OPCIONALES, TAMBIEN EN CIRCUNSTANCIAS COMO CUANDO UN METODO TIENE MUCHAS SOBRECARAS Y SE PUEDE CONFUNDIR A CUAL DE ELLAS SE ESTA HACIENDO REFERENCIA. OTRA FORMA UTIL ES LLAMAR A LOS ARGUMENTOS POR SU NOMBRE, POR EJEMPLO:

public decimal CalcularComision(decimal value, decimal comision = 4)

{

return (value \* comision) / 100m;

}

1. **Constructor**

ES BUENA PRACTICA QUE SE UTILICE EL CONSTRUCTOR DE LA CLASE PARA INICIALIZAR LOS CAMPOS. EL CONSTRUCTOR LLEVA EL NOMBRE DE LA CLASE QUE LO CONTIENE. PREFERENTEMENTE SE COLOCA ALGUN TIPO DE CODIGO QUE INICIALICE ALGUN TIPO CODIGO. POR EJEMPLO CONEXIONES A BASES DE DATOS. TAMBIEN PUEDE RECIBIR PARAMETROS ASI COMO SOBRECARGARSE CON THIS SE LLAMA DESDE UN CONSTRUCTOR A OTRO, EN CASO DE TENER CODIGO COMPLEMENTARIO. EN LA CLASE DERIVADA DEBEMOS LLAMAR EXPLCITAMENTE AL CONSTRUCTOR DE LA CLASE BASE EN CASO DE HABERLO DEFINIDO MANUALMENTE, YA QUE SI NO INTENTA LLAMAR AL CONSTRUCTOR POR DEFECTO Y OCASIONARA UN ERROR.

public claPrimeraClase()

{

}

public claPrimeraClase(string NumSeg): this()

{

}

PARA UTILIZAR UNA CLASE CON REFERENCIA COMPARTIDA UTILIZANDO STATIC, SOLO SE NECESITA COMPARTIR LOS CAMPOS Y LAS PROPIEDADES, NO ES NECESARIO PONERLE STATIC CLASS XXX, SI SE HICIERA ESTO POR DEFAULT TODOS LOS OBJETOS INTERNOS DE LA CLASE SERIAN STATIC. SU FINALIDAD ES CREAR OBJETOS DE "CONVIVENCIA" OJO. CUANDO SE DECLARAN ESTE TIPO DE CLASES, NO TIENE SENTIDO SOBRECARGAR UN CONSTRUCTOR PORQUE NUNCA SE UTILIZA EL NEW QUE ES LO QUE LO DESENCADENA.

UNA FUNCIONALIDAD INTERESATE –UTILIZADA EN LINQ- ES LA POSIBILIDAD DE INICIALIZAR LAS PROPIEDADES AL MISMO TIEMPO QUE SE CREA LA INSTANCIA DE LA CLASE. LO CUAL TAMBIEN SE PUEDE HACER MEDIANTE EL CONSTRUCTOR, SOLO QUE EL PRIMER METODO NO REQUIERE ESCRIBIR TANTO CODIGO, solo requiere escribir entre llaves los valores deseados para inicializar las propiedades.

claseBase MiClase=new ClaseBase() {prop1=x, prop2=y};

}

### Las clases auxiliares

**Metodos de extension**

LOS METODOS DE EXTENSION SON UTILES CUANDO QUEREMOS AMPLIAR LA FUNCIONALIDAD DE UNA CLASE, PERO SIN MODIFICAR SU CODIGO EXISTENTE

static class CLAExtendida

{

LA CLAVE ESTA EN QUE AL DARLE THIS INT, EL COMPILADOR SABE QUE ESTA AMPLIANDO A INT (INT32)

public static int Negativo(this int i)

{

return -i;

}

}

**Metodos virtuales**

LA CLASE BASE DECLARA UN METODO VIRTUAL

class claVirtual

{

public virtual string Nombre()

{

return "soy el nombre de la base";

}

}

class claDerivada1 : claVirtual

{

public override string Nombre()

{

return "soy nombre sobrecargado y sobreescrito, derivada 1";

}

}

class claDerivada2 : claVirtual

{

ESTA OTRA CLASE NO SOBRECARGA NI SOBREESCRIBE EL METODO

}

**Interface animal**

HEREDAR ES PODEROSO, PERO EL MAXIMO POTENCIAL SE LOGRA CUANDO SE HEREDA DE UNA INTERFAZ, LAS INTERFACES SON EL COMPLEMENTO DE LA CAPACIDAD DE HERENCIA DE LAS CLASES. LAS INTERFACES SE OCUPAN DE QUE, EL COMO NO ES ASUNTO DE SU AMBITO

INTERFACES: SON MUY UTILIZADAS PORQUE SON EL MEDIO DE IMPLEMENTAR POLIMORFISMO. SON CLASES ESPECIALES QUE SOLO DEFINENE LOS MIEMBROS (LA FIRMA), NO CONTIENEN CODIGO EJECUTABLE. CUANDO UNA CLASE HEREDA DE LA INTERFACE ESTA OBLIGADA A IMPLEMENTAR SUS COMPONENTES. NO PUEDE DEFINIR CAMPOS Y LOS EVENTOS DEBEN IMPLEMENTARSE MEDIANTE UN DELEGADO !!!!!

UNA DE LAS UTILIDADES ES CUANDO TENEMOS VARIAS CLASES QUE IMPLEMENTAN METODOS COMUNES, YA QUE AL IMPLEMENTAR INTERFACES AHORRAMOS EL ESCRIBIR CODIGO. VER NOTAS AL FINAL

RESUMIENDO: PODEMOS IMPLEMENTAR VARIAS INTERFACES Y TAMBIÉN PODEMOS DECIDIR SI QUEREMOS QUE ESAS IMPLEMENTACIONES SEAN VISIBLES A TRAVÉS DE LA PROPIA CLASE O POR MEDIO DE OBJETOS DEL TIPO DE LA INTERFAZ IMPLEMENTADA.

RESTRICCIONES: NO SE DEFINEN CAMPOS, CONSTRUCTORES, NO SE DEFINEN MODIFICADORES DE ACCESO, NO HEREDA DE UNA CLASE O

ESTRUCTURA, AUNQUE SI DE OTRA INTERFACE

NO ES NECESARIO INDICAR LA ACCESIBILIDAD, POR DEFECTOS SON PUBLIC

public interface intfAnimal

{

string Especie { get; set; }

int Edad { get; }

void Desplazar();

void Alimentarse();

}

NOTAS SOBRE LA IMPLEMENTACION DE INTERFACES

4. ¿CUANDO USAR INTERFACES?

EN GENERAL SIEMPRE QUE TENGAS, O PREVEAS QUE PUEDES TENER MÁS DE UNA CLASE PARA HACER LO MISMO: USA INTERFACES. ES MEJOR

PECAR DE EXCESO QUE DE DEFECTO EN ESTE CASO. NO TE PREOCUPES POR PENALIZACIONES DE RENDIMIENTO EN TU APLICACIÓN PORQUE NO LAS HAY

NO DIGO QUE TODA CLASE DEBA IMPLEMENTAR UNA INTERFAZ OBLIGATORIAMENTE, MUCHAS CLASES INTERNAS NO LO IMPLEMENTARÁN, PERO

EN EL CASO DE LAS CLASES PÚBLICAS (VISIBLES DESDE EL EXTERIOR) DEBERÍAS PENSARLO BIEN. ADEMÁS PENSAR EN LA INTERFAZ ANTES QUE EN LA CLASE EN SÍ, ES PENSAR EN LO QUE DEBE HACERSE EN LUGAR DE PENSAR EN COMO DEBE HACERSE.

USAR INTERFACES PERMITE A POSTERIORI CAMBIAR UNA CLASE POR OTRA QUE IMPLEMENTE LA MISMA INTERFAZ Y PODER INTEGRAR LA NUEVA CLASE DE FORMA MUCHO MÁS FÁCIL (SÓLO DEBEMOS MODIFICAR DONDE INSTANCIAMOS LOS OBJETOS PERO EL RESTO DE CÓDIGO QUEDA IGUAL).

5. SEGREGACIÓN DE INTERFACES

IMAGINA QUE TENEMOS UN SISTEMA QUE DEBE TRABAJAR CON VARIOS VEHÍCULOS, ENTRE ELLOS AVIONES Y COCHES, ASÍ QUE DECLARAMOS

LA SIGUIENTE INTERFAZ:

interface IVehiculo{

void Acelerar(int kmh);

void Frenar();

void Girar(int angulos);

void Despegar();

void Aterrizar();

}

LUEGO IMPLEMENTAMOS LA CLASE AVIÓN:

class Avion : IVehiculo{

public void Acelerar(int kmh) { ... }

public void Frenar() { ... }

public void Girar (int angulos) { ... }

public void Despegar() { ... }

public void Aterrizar() { ... }

}

Y LUEGO VAMOS A POR LA CLASE COCHE… Y AQUÍ SURGE EL PROBLEMA:

class Coche : IVehiculo{

public void Acelerar(int kmh) { ... }

public void Frenar() { ... }

public void Girar (int angulos) { ... }

public void Despegar() {throw new NotImplementedException("Coches no vuelan"); }

public void Aterrizar(){throw new NotImplementedException("Coches no vuelan"); }}

LA INTERFAZ IVEHICULO TIENE DEMASIADOS MÉTODOS Y NO DEFINE EL COMPORTAMIENTO DE TODOS LOS VEHÍCULOS, DADO QUE NO

TODOS LOS VEHÍCULOS DESPEGAN Y ATERRIZAN. EN ESTE CASO ES MEJOR DIVIDIR LA INTERFAZ EN DOS:

interface IVehiculo{

void Acelerar(int kmh);

void Frenar();

void Girar (int angulos);

}

interface IVehiculoVolador : IVehiculo{

void Despegar();

void Aterrizar();

}

FÍJATE ADEMÁS QUE IVEHICULOVOLADOR DERIVA DE IVEHICULO (EN ORIENTACIÓN A OBJETOS DECIMOS QUE HAY UNA RELACIÓN DE

HERENCIA ENTRE IVEHICULOVOLADOR Y IVEHICULO), ESO SIGNIFICA QUE UNA CLASE QUE IMPLEMENTE IVEHICULOVOLADOR DEBE IMPLEMENTAR TAMBIÉN IVEHICULO FORZOSAMENTE. POR LO TANTO PODEMOS AFIRMAR QUE TODOS LOS VEHÍCULOS VOLADORES SON TAMBIÉN VEHÍCULOS.

AHORA SI QUE LA CLASE COCHE PUEDE IMPLEMENTAR IVEHICULO Y LA CLASE AVION PUEDE IMPLEMENTAR IVEHICULOVOLADOR (Y POR

LO TANTO TAMBIÉN IVEHICULO). SI UN MÉTODO FOO() RECIBE UN OBJETO IVEHICULOVOLADOR PUEDE USAR MÉTODOS TANTO DE IVEHICULOVOLADOR COMO DE IVEHICULO:

**La clase que implementa la interface animal**

LA CLASE QUE IMPLEMENTA LA INTEFAZ (CON EL CONTEXTUAL PULSAR IMPLEMENTAR INTERFAZ) SI QUEREMOS IMPLEMENTAR MAS DE UNA INTERFAZ LO HACEMOS SEPARANDOLAS POR COMAS, SI AMBAS TIENEN UN METODO IDÉNTICO AL IMPLEMENTARLA CON EL ASISTENTE SOLO DEJARA UNO DE ELLOS

\*\*OJO\*\* SI SE HEREDA DE UNA CLASE Y UNA INTERFACE SE DEBERA RESPETAR ESE ORDEN: PRIMERO LAS CLASES Y LUEGO LAS INTERFACES Y ESTRUCTURAS

class claAnimal: intfAnimal //, intfOtra

{

public string Especie

{

get

{

throw new NotImplementedException();

}

set

{

throw new NotImplementedException();

}

}

public int Edad

{

get { throw new NotImplementedException(); }

}

IMPLEMENTACION EXPLICITA DE PROCEDIMIENTOS DE INTERFACE, PERA EVITAR CONFUSION ENCASO DE QUE SE IMPLEMENTEN MAS DE UNA INTERFACE O CLASE CON METODOS IGUALES

\*\*\*\*OJO\*\*\* CON LA IMPLEMENTACION EXPLICITA NO PODEMOS ESPECIFICAR GRADO DE PROTECCION DE LOS METODOS, O SEA NO IMPORTA QUE LE PONGAMOS PUBLIC, PORQUE NO SE VERAN DE POR SI EN UNA CLASE DEL TIPO CLAANIMAL, ESTO PORQUE SI FUERAN PUBLICOS IGUAL HABRIA CONFUSION SOBRE A QUE INTERFACE LE CORRESPONDE QUE METODO, VER ARRIBA COMO SE ACCESA

void intfAnimal.Desplazar()

{

throw new NotImplementedException();

}

public void Alimentarse()

{

throw new NotImplementedException();

}

}

**Interface que implementa otra interface**

UNA INTERFAZ DE PUEDE IMPLEMENTAR DESDE UNA CLASE, UNA ESTRUCTURA U OTRA INTERFAZ

public interface intfAnimal1:intfAnimal

{

//CODIGO

}

**Clase que implementa la interface iComparable**

public class claX : IComparable

{

public int CompareTo(object obj)

{

throw new NotImplementedException();

}

}

**Clases parciales**

UTIL PARA CLASES MUY GRANDES SEPARADAS EN PARTES PEQUEÑAS PARA SU MEJOR MANEJO. LO UNICO ES QUE HAY QUE AGREGAR LA PALABRA PARTIAL. DEL MISMO MODO SE PUEDEN CREAR INTERFACES Y ESTRUCTURAS PARCIALES

partial class claMiClaseParcial

{

//le declaramos un constructor

public claMiClaseParcial()

{

//codigo para el constructor

}

}

partial class claMiClaseParcial

{

//le declaramos un metodo

public void mensaje()

{

Console.WriteLine("metodo de la clase parcial");

}

}

**Clases abstractas**

A VECES POR ALGUNA RAZON SE COMIENZA A DUPLICAR CODIGO, UNA FORMA DE REFACTORIZAR ES CREAR UNA CLASE ESPECIAL PARA EL CODIGO REPETIDO, ESTO PARECE BUENA SOLUCION, SOLO QUE AUN ES POSIBLE CREAR UNSTANCIAS DE ESTA NUEVA CLASE, LO CUAL OPACA EL PRINCIPIO POR EL CUAL FUE CREADA. PARA ELLO EXISTEN LAS CLASES ABASTRACTAS, LAS CUALES NO SE PUEDEN INSTANCIAR, SU IMPLEMENTACION RADICA EN HEREDAR

abstract class absClaseAbstracta

{

METODO ABSTRACTO, PARECIDO A UN METOD VIRTUAL, YA QUE ES QUIEN IMPLEMENTA EL QUE DEBE PROPORCIONAR SU FUNCIONALIDAD Y TAMBIEN ESTA OBLIGADO A SOBRECARGARLO

//abstract void metodo();

}

**Clases selladas**

CUANDO DECIDE QUE UNA CLASE NO DEBE SER USADA COMO CLASE BASE

sealed class claSellada

{

ES POSIBLE CREAR METODOS SELLADOS, AUN EN CLASES NO SELLADAS, ESTO CON EL FIN DE EVITAR QUE SEAN SOBRECARGADOS

//sealed void Proc()

//{

//}

}

**Otros temas de herencia**

HERENCIA VISUAL. ESTE TIPO DE HERENCIA ES MUY UTIL CUANDO DECIDIMOS USAR UN FORMULARIO BASE, HEREDARLO Y EXTENDER SU FUNCIONALIDAD. PARA ELLO ES NECESARIO COLOCARLO EN UN NUEVO PROYECTO DE LIBRERÍA DE CLASES Y LUEGO CREAR FORMULARIOS QUE HEREDEN DE EL. LOS CONTROLES QUE SERAN MANIPULABLES SE DEBEN PONER COMO PROTECTED EN SU MODIFICADOR.

## Eventos y delegados

### Delegados tipo EventHandler

DELEGADOS: UN DELEGADO ES UN PUNTERO HACIA UN METODO, POR LO TANTO ES POSIBLE LLAMAR a DIVERSOS METODOS DESDE UN DELEGADO, SIEMPRE Y CUANDO TENGAN LA MISMA FIRMA.

SENDER ES EL OBJETO QUE REPRESENTA EL CONTROL QUE HA PRODUCIDO EL EVENTO E ES DE UNA CLASE DERIVADA DE EVENTHANDLER QUE EN ALGUNOS CASOS NOS PUEDE PROPORCIONAR INFORMACIÓN EXTRA SOBRE EL EVENTO QUE SE HA PRODUCIDO EN LA INSTRUCCION QUE VINCULA : THIS.BUTTON1.CLICK += NEW SYSTEM.EVENTHANDLER(THIS.BUTTON1\_CLICK);

SUPONGAMOS QUE TENEMOS UNA CLASE NOMBRE Y QUEREMOS QUE NOTIFIQUE CUANDO ALGUIEN CAMBIE LA PROPIEDAD NOMBRE, SE HACEN 4 COSAS

1.- PRIMERO CREAMOS EN NOMBRE DEL EVENTO EN LA CLASE QUE LO IMPLEMENTA

2.- SE CREA EL VINCULO EN EL LUGAR DE LA CLASE DESDE DONDE SE QUIERE QUE SE LLAME AL EVENTO

3.- SE CREA EL CODIGO DESDE DONDE SE LLAMARA AL DELEGADO

4.- SE CREA EL METODO QUE DEFINE EL COMPORTAMIENTO DEL EVENTO

SI POR ALGUNA RAZON SE TUVIERA QUE AGREGAR UN METODO QUE NO TIENE LA MISMA FIRMA QUE LAS DEMAS, SE PODRIA UTILIZAR UN ADAPTADOR. POR EJEMPLO PARA AGREGAR UN METODO QUE RECIBE UN PARAMETRO TIPO ENTERO SE PODRIA CREAR UN PROCEDIMIENTO SIN PARAMETROS, Y DENTRO DE ESTE EL PROCEDIMIENTO CON UN PARAMETRO 0 PARA UN TRATAMIENTO MAS AMPLIO DE ESTOS TEMAS EXISTEN LAS EXPRESIONES LAMBDA \*\*PP 446

1. **La clase principal**

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

claMyClaseNombre myC = new claMyClaseNombre();

private void Form1\_Load(object sender, EventArgs e)

{

AQUI IBA LA DECLARACION DE LA VARIABLE, LA SAqUE A NIVEL MODULO PARA PODER MANDAR EL NUEVO NOMBRE COMO

PARAMETRO DEL MESSAGEBOX

myC.NombreCambiado +=new EventHandler(myC\_NombreCambiado);

myC.Nombre = "pepe";

}

EL EVENTO EN SI QUE QUEREMOS QUE SE EJECUTE

private void myC\_NombreCambiado(object sender, EventArgs e)

{

MessageBox.Show("Nombre cambiado por '" + myC.Nombre.ToString() + "', actualizar la base de datos!" );

}

}

1. **La clase que implementa el evento**

public class claMyClaseNombre

{

EL DELEGADO DE TIPO EVENTHANDLER

public event EventHandler NombreCambiado;

private string strNombre="";

public string Nombre

{

get

{

return strNombre;

}

set

{

strNombre = value;

if (NombreCambiado !=null )

DESDE AQUI SE DESENCADENA LA LLAMADA AL DELEGADO

NombreCambiado(this, new EventArgs());

}

}

}

### Delegados que invocan a distintos metodos

1. **La clase principal**

public partial class Form1 : Form

{

**2: OTRA** DE LAS FORMAS DE LLAMAR A ESOS METODOS ES A TRAVES DE UN DELEGADO, PORQUE ES UNA REFERENCIA A UN

METODO, SOLO DEBE TENER LA MISMA FIRMA

delegate void delSaludar(string Nom);

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

**3: LLAMANDO** A ESE METODO A TRAVES DE UN DELEGADO, AHORA NO ES NECESARIO REFERENCIAR AL PRIMERO METODO,

SINO SOLAMENTE AL DELEGADO

delSaludar myDelSaludar;

myDelSaludar = new delSaludar(saludar);

**4: PERO ADEMAS**, UN DELEGADO PUEDE SER MULTIMIEMBRO, ASI QUE PUEDO AGREGAR OTRO METODO QUE TAMBIEN TIENE

LA MISMA FIRMA

myDelSaludar += new delSaludar(despedir);

POR EJEMPLO, SI QUISIERA QUITAR UNO DE LOS PROCEDIMIENTOS QUE CONTIENE EL DELEGADO SERIA

//myDelSaludar -= new delSaludar(despedir);

//asi solo mostraria el primero

myDelSaludar("señor");

**5: AL CLICK** DEL BOTON 1 LE AGREGAMOS UN MANEJADOR DE EVENTOS PARA QUE EJECUTE LO QUE TIENE EL BOTON 2

OJO. PODRIAMOS AGREGAR CUALQUIER OTRO METODO DE LA MISMA FORMA, LA UNICA CONDICION ES QUE TENGA LA

MISMA FIRMA

this.button1.Click += new System.EventHandler(this.button2\_Click);

this.button1.Click += new System.EventHandler(this.procCualqProcedimiento);

}

**1: TENGO** UN PAR DE METODOS VOID QUE REALIZAN UNA ACCION, ESTOS PUEDEN SER INVOCADOS DESDE ALGUNA PARTE

DEL PROGRAMA EN FORMA INDIVIDUAL

private void saludar(string Nom)

{

MessageBox.Show("Hola " + Nom);

}

private void despedir(string Nom)

{

MessageBox.Show("Adios " + Nom);

}

ESTE ES EL EJEMPLO DE LIGAR AL CLICK DE UN BOTON CUALQUIER OTRO PROCEDIMIENTO

private void procCualqProcedimiento(object sender, EventArgs e )

{

MessageBox.Show("soy un procedimiento cualquiera ligado al click del boton 1");

}

PROBANDO CLICKS DE UN BOTON DESDE OTRO

private void button1\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

MessageBox.Show("Hola desde el boton 2");

}

}

### Segundo ejemplo de delegados tipo EventHandler

EJEMPLO: SE TIENEN DIVERSAS MAQUINAS EN UNA FABRICA, CADA UNA DE DIFERENTE MARCA Y CON DETALLES PROPIOS, PERO EL METODO PARA APAGARLAS TIENE LA MISMA FIRMA. UNA FORMA SERIA IMPLEMENTAR UN PROCESO VOID CON LOS NOMBRES DE LOS DISTINTOS PROCESOS, PERO ESTO ES RIGIDO, POR EJEMPLO SI LLEGA UNA MAQUIA NUEVA HABRIA QUE MODIFICA REL CODIGO Y AGREGAR SU PROCEDIMIENTO. LA SOLUCION ES UN DELEGADO.

EN ESTE EJEMPLO ES NECESARIO UTILIZAR UN DELEGADO DE TIPO EVENTHANDLER QUE SE IMPLEMENTA JUNTO CON UN EVENTO.

class Program

{

2 SE CREA EL DELEGADO CON LA FIRMA CORRESPONDIENTE

delegate void delApagaMaquinas(string Nom);

static void Main(string[] args)

{

3 ASIGNAMOS LOS PUNTEROS Y LLAMAMOS AL DELEGADO

delApagaMaquinas myDelApagaMaquinas;

myDelApagaMaquinas = new delApagaMaquinas(claM1.ApagaM1);

myDelApagaMaquinas = new delApagaMaquinas(claM2.ShutM2);

myDelApagaMaquinas = new delApagaMaquinas(claM3.OffM3);

4 AQUI LLAMAMOS A LOS METODOS

myDelApagaMaquinas("proc gral");

EN ESTE EJEMPLO SOLO SE USA EL DELEGADO PARA LLAMAR A LOS DISTINTOS PUNTEROS, O SEA A DISTINTOS METODOS. EJEMPLO 2: AGREGANDO, SI AHORA LA CLASE M1 IMPLEMENTARA UNA PROPIEDAD DONDE SE MONITOREA SU TEMPERATURA PARA DETECTAR SI SE SOBRECALIENTA Y APAGARLA.

9 EL LUGAR DONDE SE ESTA TOMANDO LA TEMPERATURA DE LA MAQUINA

claM1 myClaM1 = new claM1();

myClaM1.eveTemperatura += new EventHandler(MideTemperatura);

myClaM1.Temperatura = 80;

}

8 EL PROCEDIMIENTO EN SI QUE SE EJECUTA CON EL EVENTO

private static void MideTemperatura(object sender, EventArgs e)

{

Console.WriteLine("apagado por calentamiento");

Console.Read();

}

}

1. **Las 3 clases de maquinas distintas**

1 SE TIENEN LAS 3 DIFERENTES MAQUINAS CON SU PROCESO DE APAGADO, TODOS CON FIRMA IGUAL. OJO SE DECLARARON STATIC LOS PROCEDIMIENTOS PARA PODERLOS LLAMAR DESDE LA ASIGNACION DEL DELEGADO VER OTRAS FORMAS

class claM1

{

5 SE DECLARA EL DELEGADO TIPO EVENTHANDLER

public event EventHandler eveTemperatura;

//6 el campo y la propiedad

private int intTemperatura = 0;

public int Temperatura

{

get

{

return intTemperatura;

}

set

{ intTemperatura = value;

SE DEBE HACER LA COMPROBACION PORQUE EL CAMPO EVENTO ES IMPLICITAMENTE NULO

if (eveTemperatura != null)

{

if (intTemperatura >= 90)

{

7 EL MOMENTO DONDE SE LLAMA AL EVENTO APAGAR

eveTemperatura(this, new EventArgs());

}

}

}

}

public static void ApagaM1(string Nom)

{

Console.WriteLine("apagada m1 por " +Nom);

//Console.Read();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class claM2

{

public static void ShutM2(string Nom)

{

Console.WriteLine("shutdown m2 por " +Nom);

//Console.Read();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class claM3

{

public static void OffM3(string Nom)

{

Console.WriteLine("off m3 por " + Nom);

//Console.Read();

}

}

OTRA FORMA DE IMPLEMENTAR DELEGADOS QUE ES UN POCO MAS BREVE, SE UTILIZAN LAS MISMAS TRES CLASES DE MAQUINAS CON SUS PROCEDIMIENTO DE APAGADO.

class Program

{

//2 se crea el delegado con la firma correspondiente

delegate void delApagaMaquinas(string Nom);

//para la otra forma de usar delegados se crea una instancia del delegado como campo privado

private delApagaMaquinas delControlApagador;

static void Main(string[] args)

{

//AL INICIAR EL PROGRAMA SE EJECUTA ESTA INSTRUCCION

Program p=new Program();

p.procApaga();

Console.Read();

}

public void procApaga()

{

//SE AGREGAN LOS PROCEDIMIENTOS DE LAS CLASES AL DELEGADO

delControlApagador += claM3.OffM3;

delControlApagador += claM2.ShutM2;

delControlApagador += claM1.ApagaM1;

delControlApagador("APAGADO POR EL CONTROLADOR");

}

}

### Expresiones lambda, delegados y método anónimos

LA IDEA CENTRAL DE LOS DELEGADOS ANONIMOS ES LA SIGUIENTE: A delControlApagador LE QUIERO PASAR UN metodo, PERO NO PRECISAMENTE POR MEDIO DE UNA DIRECCION DE MEMORIA QUE LE INDIQUE DONDE ESTA, SINO QUE LE PUEDO ESCRIBIR IN SITU EL metodo QUE LE VOY A PASAR, CON LO CUAL EL DELEGADO NO REQUERIRIA NI NOMBRE. LA UTILIDAD DE ESTOS ES POR EJEMPLO CUANDO SE REQUERIDA UTILIZAR ESE CODIGO EN UN SOLO LUGAR, POR LO CUAL SOLO SE NECESITA UNA VEZ.

delControlApagador +=delegate(string mensaje)

{

Console.WriteLine("off m3");

};

LOS MÉTODOS ANÓNIMOS SON UNA TÉCNICA ANTERIOR A LAS EXPRESIONES LAMBDA, CUMPLEN EL MISMO PAPEL PERO ESTOS SON MENOS VERSÁTILES QUE LAS SEGUNDAS.

EN EL EJEMPLO DE LAS MAQUINAS Y EL DELEGADO QUE MANEJA SU APAGADO TENDRÍAMOS LA SIGUIENTE FORMA MEDIANTE UN MÉTODO ANÓNIMO. Con esta INSTRUCCIÓN EL DELEGADO AGREGARA AL METODO AUNQUE NO CUENTE CON SU MISMA FIRMA.

delControlApagador += delegate { claM3.OffM3(); };

UNA EXPRESIÓN LAMBDA ES UNA FUNCIÓN SIN NOMBRE QUE CALCULA Y DEVUELVE UN SOLO VALOR. NO REQUIERE NOMBRE PORQUE SE DECLARA IN SITU, Y EN REALIDAD PUEDEN O NO RETORNAR UN VALOR Y RECIBIR O NO PARAMETROS. SE PUEDEN UTILIZAR LAS EXPRESIONES LAMBDA DONDEQUIERA QUE UN TIPO DE DELEGADO SEA VÁLIDO. Por ello, SI POR ALGUNA RAZON SE TUVIERA QUE AGREGAR A UN DELEGADO UN METODO QUE NO TIENE LA MISMA FIRMA QUE LAS DEMAS, LAS EXPRESIONES LAMBDA SON IDEALES.

RESUMIENDO DE FORMA SENCILLA: SON FORMAS DE ESCRIBIR METODOS BREVES EN UNA SOLA LINEA. TIENEN LA SIGUIENTE ESTRUCTURA GENERAL:

Forma general: **parámetros => expresión**, donde:  
- **parámetros**: lista de parámetros separados por comas  
- **"=>"** : separador, SE LEE “VA A”.  
- **expresión**: implementación de las operaciones a realizar

PRIMER EJEMPLO DE EXPRESIONES LAMBDA CON FIRMAS IGUALES: LA UTILIDAD DE ESTO ES SIGINIFICATIVA, PUES YA NO ES NECESARIO ESCRIBIR NOMBRE NI TIPO, PORQUE LA DEFINICION DEL DELEGADO YA LOS IMPLICA, O SEA QUE SOLO SE REQUIERE EL NOMBRE DEL METODO SEGUIDO POR EL SEPARADOR LAMBDA “AHORRO DE CODIGO”.

delControlApagador += (mensaje)=>

{

Console.WriteLine("off m3");

};

INCLUSIVE PODRIA SIMPLIFICARSE AUN MAS EL CODIGO, YA QUE AL TRATARSE DE UNICAMENTE UNA EXPRESION (UNA INSTRUCCIÓN QUE FUNCIONA EN UNA SOLA LINEA DE CODIGO) SE PUEDEN OMITIR PARENTESIS Y LLAVES

delControlApagador += MENSAJE =>

Console.WriteLine("off m3 CON EXPRESIONES LAMBDAA UN MAS SIMPLIFICADA");

## Funciones

### Codificar decodificar caracteres especiales ASP.NET

Toma la cadena con caractereres y los decodifica para interpretarlos correctamente, la Ñ y las tildes

HttpUtility.HtmlDecode(griContratista.Rows[griContratista.SelectedIndex].Cells[2].Text);

HttpUtility.HtmlEncode(griContratista.Rows[griContratista.SelectedIndex].Cells[2].Text);

### Longitud de la cadena

strCadena = texCadena.Text;

texResultado.Text = Convert.ToString ( strCadena.Length );

### IndexOff

SE PUEDE UTILIZAR COMO EL EQUIVALENTE DE INSTR. SI DA -1 QUIERE DECIR QUE EL CRITERIO NO SE CUMPLE.

strCadena = texCadena.Text;

texResultado.Text = Convert.ToString(strCadena.IndexOf(texCriterio .Text ));

### LastIndexOff

strCadena = texCadena.Text;

texResultado.Text = Convert.ToString(strCadena.LastIndexOf (texCriterio.Text));

### ToUpper

strCadena = texCadena.Text;

texResultado.Text = strCadena.ToUpper();

VER EL TOLOWER PARA PASAR A MINUSCULAS

### Substring (Left, Rigth o Mid)

SI COMENZAMOS CON EL INDEX O FUNCIONARA COMO LEFT SI UTILIZAMOS LA LONGITUD DE LA CADENA FUNCIONA COMO RIGHT

strCadena = texCadena.Text;

//texResultado.Text = strCadena.Substring(strCadena.Length -5,5);

texResultado.Text = strCadena.Substring(0,5);

string s3 = "Visual C# Express";

System.Console.WriteLine(s3.Substring(7, 2)); // outputs "C#"

### Remove

strCadena = texCadena.Text;

texResultado.Text = strCadena.Remove(5, 10);

### Replace

strCadena = texCadena.Text;

texResultado.Text = strCadena.Replace("2010","2011");

### Trim

strCadena = texCadena.Text;

texResultado.Text = Convert.ToString(strCadena.Trim());

### Compare

strCadena = texCadena.Text;

int intComp=0;

intComp = string.Compare(strCadena, texCriterio.Text);

texResultado.Text = Convert .ToString ( intComp);

### Insert

strCadena = texCadena.Text;

texResultado.Text =strCadena.Insert(5," XXXXXX ");

### PadLeft o PadRigth

strCadena = texCadena.Text;

//texResultado.Text = strCadena.PadLeft(10, 'c');

texResultado.Text = strCadena.PadRight (10, 'c');

### Startswith

strCadena = texCadena.Text;

texResultado.Text = Convert .ToString ( strCadena.StartsWith(texCriterio .Text ));

### Inputbox

INPUTBOX. EN C# NO EXISTE ESTA INSTRUCCION, SIN EMBARGO, SI SE DESEA UTILIZARLA SE PUEDE LLAMAR A UNA COLABORACION CON VB. AGREGAR LA REFERENCIA Y EL USING

using Microsoft.VisualBasic;

string result = Microsoft.VisualBasic.Interaction.InputBox("kkk", "text", "", 10, 20);

Console.WriteLine(result);

Console.Read();

### El método Format de la clase String

PONE FORMATO DE MILES REDONDEANDO A DOS DECIMALES

//double douCirterio;

//douCirterio =Convert .ToDouble ( texCriterio.Text);

//texResultado.Text = String.Format("{0:0,0.00}", douCirterio);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DateTime datDT;

datDT = Convert.ToDateTime(texCriterio.Text);

//String.Format("{0:t}", dt); // "4:05 PM" ShortTime

//String.Format("{0:d}", dt); // "3/9/2008" ShortDate

//String.Format("{0:T}", dt); // "4:05:07 PM" LongTime

//String.Format("{0:D}", dt); // "Sunday, March 09, 2008" LongDate

//String.Format("{0:f}", dt); // "Sunday, March 09, 2008 4:05 PM" LongDate+ShortTime

//String.Format("{0:F}", dt); // "Sunday, March 09, 2008 4:05:07 PM" FullDateTime

//String.Format("{0:g}", dt); // "3/9/2008 4:05 PM" ShortDate+ShortTime

//String.Format("{0:G}", dt); // "3/9/2008 4:05:07 PM" ShortDate+LongTime

//String.Format("{0:m}", dt); // "March 09" MonthDay

//String.Format("{0:y}", dt); // "March, 2008" YearMonth

//String.Format("{0:r}", dt); // "Sun, 09 Mar 2008 16:05:07 GMT" RFC1123

//String.Format("{0:s}", dt); // "2008-03-09T16:05:07" SortableDateTime

//String.Format("{0:u}", dt); // "2008-03-09 16:05:07Z" UniversalSortableDateTime

//y->AÑO M->MES d->DIA h->HORA H->HORA 12/24 m->MINTO s->SEGUNDO t->AM/PM

//texResultado.Text = String.Format("{0:dd MMM yyyy}", datDT );

texResultado.Text = String.Format("{0:D}", datDT);

### IsNumeric equivalente

ES UNA FUNCION PERSONALIZADA QUE UTILIZA EL ENSAMBLADO using System.Text.RegularExpressions; SOLO SE INVOCA Y SE EVALUA SI DA CIERTO O FALSO.

private bool funEsNumero(string strTextEntry)

{

Regex objNotWholePattern = new Regex("[^0-9]");

return !objNotWholePattern.IsMatch(strTextEntry);

}

OTRA

public static bool IsNumeric(object Expression)

{

bool isNum;

double retNum;

isNum = Double.TryParse(Convert.ToString(Expression), System.Globalization.NumberStyles.Any, System.Globalization.NumberFormatInfo.InvariantInfo, out retNum);

return isNum;

}

## Componentes

### Creacion de un componente (control de usuario)

SE CREA UN PROYECTO DE LA BIBLIOTECA DE CLASES O UN WINDOWS FORM CONTROL LIBRARY, SE AGREGA UN NUEVO ELEMENTO TIPO CONTROL Y A ESTE SE LE AGREGAN LOS CONTROLES QUE SE DESEE, SE COMPILA Y LISTO. EN OTRO PROYECTO SE AGREGA LA REFERENCIA A AL PRIMER PROYECTO, SE UTILIZA USING Y SE CREA EN TIEMPO DE EJECUCION. (TAMBIEN SE AGREGA COMO CONTROL A LA BARRA DE HERRAMIENTAS PARA ARRASTRAR Y SOLTAR)

using c1;

namespace WindowsFormsApplication2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

c1.UserControl1 myC1=new UserControl1();

this.Controls.Add(myC1);

}

}

EXISTEN PROPIEDADES QUE SE PUEDEN EXPONER PARA QUE EL USUARIO QUE UTILICE EL CONTROL LAS MANIPULE, SOLAMENTE SE CREA EL CAMPO PRIVADO Y LA PROPIEDAD PUBLICA. CON ESO YA APARECERA LA PROPIEDAD EN LA BARRA

### Creación de un botón y su evento (EventHandler)

DECLARAMOS EL OBJETO BUTTON.

Button MiControl = new Button();

DECLARAMOS UN NOMBRE AL CONTROL (SI QUEREMOS)

MiControl.Name = "btn1";

CAMBIAMOS ALGUNAS DE SUS PROPIEDADES

MiControl.Text = "Ejemplo de Botón";

MiControl.Top = 10;

MiControl.Left = 10;

MiControl.Width = 110;

MiControl.Height = 25;

AÑADIMOS EL CONTROL AL FORMULARIO

this.Controls.Add(MiControl);

LO VERDADERAMENTE UTIL ES RELACIONARLO CON UN CODIGO MEDIANTE EVENTHANDLER, Y DEFINIR EL EVENTO POR SEPARADO

MiControl.Click += new EventHandler(btn1Click);

CONTRUIMOS EL EVENTO QUE LLAMA EL DELEGADO

private void btn1Click(object Sender, EventArgs e)

{

MOSTRAMOS UN MENSAJE

MessageBox.Show("Soy el Control Button con texto: '" + ((Button)Sender).Text + "'");

}

1. **Modificar las propiedades del control creado dinámicamente**

LA COSAS ESTA ASI: COMO REFERENCIAR AL CONTROL INSERTADO DINAMICAMENTE?SI SE INTENTARA DE LA FORMA NORMAL CAUSARIA UN ERROR DE QUE NO ESTA DECLARADO SI QUISIERAMOS HACERLO DE FORMA NORMAL TENDRIAMOS QUE SACAR A NIVEL FORMULARIO LA DECLARACIÓN DE BUTTON MICONTROL...ENTONCES HACEMOS UN CAST PARA BUSCARL EL OBJETO btn1 EN EL FORMULARIO ACTUAL

MODIFICAMOS LA PROPIEDAD TEXT DEL CONTROL INSERTADO

((Button)this.FindForm().Controls["btn1"]).Text = "Otro texto";

### Creación de un array de controles en t.e.

DECLARAMOS LA VARIABLE CONTADOR DEL BUCLE PARA

byte i;

DECLARAMOS LA VARIABLE CONTADOR DEL NÚMERO DE CONTROLES A CREAR

byte intNumControles = 5;

INICIAMOS EL BUCLE PARA

for (i = 0; i < intNumControles; i++)

{

DECLARAMOS EL OBJETO TEXTBOX

TextBox MiControl = new TextBox();

LE ASIGNAMOS UN NOMBRE AL CONTROL

MiControl.Name = "txt1";

UTILIZAMOS LA PROPIEDAD TAG PARA ALMACENAR AHÍ EL VALOR DEL CONTROL DE LA MATRIZ VIRTUAL

MiControl.Tag = i;

LE ASIGNAMOS UN TAMAÑO EN EL FORMULARIO WINDOWS

MiControl.Size = new Size(100, 20);

LE ASIGNAMOS UNA POSICIÓN EN EL FORMULARIO WINDOWS

MiControl.Location = new Point(50, 22 \* (i + 1));

LE CAMBIAMOS LA PROPIEDAD TEXT

MiControl.Text = MiControl.Name + "(" + i.ToString() + ")";

AÑADIMOS EL CONTROL AL FORMULARIO

this.Controls.Add(MiControl);

AÑADIMOS EL EVENTO CLICK AL CONTROL CREADO DINÁMICAMENTE

MiControl.Click += new EventHandler(txt1Click);

public void txt1Click(object Sender, EventArgs e)

{

MOSTRAMOS UN MENSAJE

MessageBox.Show("Control " + ((TextBox)Sender).Tag.ToString());

}

### Referenciar controles en un contenedor

TENEMOS UN CONTROL GROUPBOX1 EN EL QUE TENEMOS UN LABEL, UN BUTTON Y UN TEXT. Primero BUSCA TODOS LOS OBJETOS TIPO TEXTBOX EN EL CONTENEDOR PRINCIPAL QUE ES EL FORMULARIO, PERO NO ENCONTRARA LOS QUE ESTAN EN OTROS SUB CONTENEDORES, EN ESTE CASO EN EL GROUPBOX1

FOREACH( OBJECT OBJ IN CONTROLS )

IF( OBJ IS TEXTBOX )

HACEMOS UNA CONVERSION EXPLICITA DEL OBJETO A TEXTBOX

((TEXTBOX)OBJ).TEXT = "ENCONTRADO";

1. **Buscar en un contenedor en particular**

foreach (object obj in groupBox1.Controls)

if (obj is TextBox)

{

((TextBox)obj).Text = "textbox encontrado";

}

else if (obj is Label )

{

((Label)obj).Text = "label encontrado";

}

else if (obj is Button )

{

((Button)obj).Text = "button encontrado";

}

EJEMPLO DE COMO BUSCAR EN LAS TABS DE UN TABCONTROL

// for( byte i = 0; i < tabControl1.TabPages.Count; i++ )

// foreach( object obj in tabControl1.TabPages[i].Controls )

// if (obj is TextBox)

// ((TextBox)obj).Text = "";

1. **Buscar en todos los contenedores**

LLAMA AL PROCEDIMIENTO PRIVADO PASANDOLE THIS COMO ARGUMENTO, O EL AMBITO ACTUAL DEL FORMULARIO. BUSCA TODOS LOS CONTROLES TIPO TEXTBOX Y LES PASA UNA CADENA PARA MOSTRAR EN SU TEXT

VaciarTextBox(this);

private void VaciarTextBox(Control Parent)

{

foreach (Control obj in Parent.Controls)

{

if (obj.Controls.Count > 0)

VaciarTextBox(obj);

if (obj is TextBox)

((TextBox)obj).Text = "encontrados todos";

}

}

## Datos (dataset tipados)

## Colecciones o listas

LAS COLECCIONES ALMACENAN Y DEVUELVEN TIPOS OBJECT Y SON TIPOS POR REFERENCIA, O SEA QUE HAY QUE CREAR UNA NUEVA INSTANCIA PARA ACCESARLA. LAS COLECCIONES NO PUEDEN SER DE MAS DE UNA DIMENSION, PERO SUS ELEMENTOS PUEDEN SER ELLOS MISMOS COLECCIONES, LO QUE DA COMO RESULTADO COLECCIONES MULTIDIMENSIONALES

INCONVENIENTES PRINCIPALES DE UNA MATRIZ o ARRAY:

1.- PARA REDIMENSIONAR HAY QUE CREAR UNA NUEVA Y ESTO TIENE IMPLICACIONES COMO PERDER ELEMENTOS

SI LA NUEVA ES MAS PEQUEÑA

2.- PARA ELIMINAR UN ELEMENTO HAY QUE MOVER TODOS LOS ELEMENTOS QUE LE SIGUEN HACIA ARRIBA, ELIMINANDO

EL ULTIMO PARA QUE NO SE REPITA

3.- PARA INSERTAR SE TIENEN QUE MOVER TODOS HACIA ABAJO, PREVINIENDO DE NO PERDER EL ULTIMO

4.- ES FORZOSO UTILIZAR UN INDICE PARA ACCESAR A SUS ELEMENTOS

LAS COLECCIONES SE IMPLEMENTAN EN INTERFACES, PARA HEREDAR CIERTO COMPORTAMIENTO. LOS ARRAYS DERIVAN DE SYSTEM.ARRAY, QUE A SU VEZ DERIVA DE ICOLLECTION Y DE IENUMERABLE. AL DERIVAR DE ESTAS, SON COLECCIONES QUE PROPORRCIONAN FUNCIONALIDADES BASICAS Y CIERTAS CARACTERISTICAS COMO ADD, ITEM, CLEAR, INDEXOF, REMOVE, QUE DEBE INDICARSE SU TAMAÑO

DESDE EL INICIO, QUE NO SE PUEDEN AMPLIAR O DISMINUIR EN CUALQUIER MOMENTO, EL TIPO DE DATOS QUE CONTENDRA, ETC.

COLECCIONES ICOLLECTION: ES COMO LA COLECCION GENERICA DE LAS DEMAS, DERIVA DE IENUMERABLE LO CUAL PERMITE RECORRERLAS CON BUCLES FOREACH. NOS PERMITE LA MANIPULACIÓN DE LOS DATOS DE UNA FORMA BÁSICA O ELEMENTAL, DE FORMA QUE SU USO SEA PARA CASOS MUY CONCRETOS. LAS COLECCIONES QUEUE, STACK Y BITARRAY SON EJEMPLOS QUE COLECCIONES QUE DERIVAN DE ESTA.

COLECCIONES ILIST: SE UTILIZA EN LAS COLECCIONES QUE QUEREMOS ACCEDER MEDIANTE UN INDICE ARRAYLIST: PERMITE ALMACENAR CUALQUIER TIPO DE OBJETO, PERO POR LO MISMO DEBEMOS HACER UN CAST ANTES DE UTILIZAR ALGUNO DE SUS ELEMENTOS.

COLECCIONES IDICTIONARY: SIEMPRE MANTIENEN EL PAR CLAVE/VALOR, YA QUE LA FORMA DE ACCEDER A LOS ELEMENTOS ES MEDIANTE UNA CLAVE ÚNICA. POR TANTO, CADA VEZ QUE AÑADIMOS UN ELEMENTO A UNA COLECCIÓN DE ESTE TIPO TENDREMOS QUE INDICAR UNA CLAVE Y UN VALOR. CADA VALOR ESTARÁ RELACIONADO CON SU CORRESPONDIENTE CLAVE. LAS CLAVES Y VALORES PUEDEN SER DE CUALQUER TIPO

SE UTILIZAN LOS SIGUIENTES ESPACIOS DE NOMBRES:

//using System.Collections;

//using System.Collections.Generic;

SE DICE QUE LAS COLECCIONES QUE DERIAN DIRECTAMENTE DE SYSTEM CLLECTION SON PESADAS PORQUE IMPLICAN UN PROCESO DE BOXING UNBOXING.

### Colecciones que derivan de System.Collections

private void Form1\_Load(object sender, EventArgs e)

{

1. **ArrayList**

ArrayList lista = new ArrayList();

lista.Add("Pepe");

lista.Add("Lola");

lista.Add(9);

//INTRUDUCE 15 EN EL SEGUNDO LUGAR

lista.Insert(1,15);

//ELIMINA EL PRIMER ELEMENTO QUE SEA PEPE

lista.Remove("pepe");

//NOTAR QUE SE UTILIZA COUNT, EN LOS ARRAYS LENGTH

for (int i = 0; i < lista.Count - 1; i++)

{

//IMPLEMENTACION

}

//ELIMINA EL ELEMENTO QUE ESTE EN SEGUNDO LUGAR

lista.RemoveAt(1);

1. **Queue**

//EL PRIMER ELEMENTO EN LLEGAR ES EL PRIMERO EN SALIR

Queue Nums = new Queue();

//INSERTA EN LA COLA

foreach (int N in new int[4] { 9, 3, 7, 2 })

{

//LOS PONE EN COLA ENTRA PRIMERO EL 9

Nums.Enqueue(N);

}

while (Nums.Count > 0)

{

//EL PRIMERO EN SALIR ES EL 9

int N = (int)Nums.Dequeue();

}

1. **Stack**

//EL ULTIMO QUE LLEGA ES EL PRIMERO QUE SALE

Stack NumsS = new Stack();

foreach(int N in new int[4]{9,3,7,2})

{

//ENTRA PRIMERO EL 9

NumsS.Push(N);

}

while (NumsS.Count > 0)

{

//SALE PRIMERO EL 2

int N = (int)NumsS.Pop();

}

1. **Hahstable**

//MANTIENE INTERNAMENTE DOS MATRICES OBJECT, UNA PARA LOS KEYS Y OTRA PARA LOS VALORES. POR ELLO NO PUEDE

//TENER CLAVES DUPLICADAS, OCUPA MEMORIA CONSIDERABLE,

Hashtable hashT = new Hashtable();

hashT.Add("uno", 1);

hashT.Add("dos", 2);

hashT.Add("tres", 3);

//SE PUEDE RECORRER MEDIANTE UN FOR, PERO EXISTE UNA RUTINA ESPECIAL, OJO, EL INDICE NO

//CORRESPONDE CON EL ORDEN EN QUE SE GUARDAN

//SON OBJETOS Y POR ESO SE UTILIZA FOREACH

foreach (DictionaryEntry Ent in hashT)

{

MessageBox.Show(Ent.Key + " " + Ent.Value );

}

1. **Sortedlist**

//PARECIDA A HASHTABLE, PERO CON UN INDICE SIEMPRE ORDENADO, CUANDO AGREGAMOS O ELIMINAMOS

//LA CLASE SE OCUPA DE ESTAR SIEMPRE ORDENADA. IGUAL QUE LA ANTERIOR NO PODEMOS AGREGAR CLAVES

//DUPLICADAS

SortedList Lista = new SortedList();

Lista["d"]=10;

Lista["c"] = 5;

Lista["b"] = 12;

Lista["a"] = 3;

foreach (DictionaryEntry Elem in Lista)

{

//LA SALIDA ES ORDENADA ALFABETICAMENTE POR EL NOMBRE, O ELEM.VALUE

int age = (int)Elem.Value;

//ESTA ES LA PROPIEDAD QUE ORDENA

string Nom = (string)Elem.Key;

MessageBox.Show ("Nombre:" + age + " edad " + Nom);

Console.Read();

}

}

### Clase personalizada Collection Base

LA IDEA ES CREAR UNA CLASE ICOMPARABLE PARA QUE FUNCIONE COMO TIPO Y PERMITA ORDENAR POR APELLIDO (METODO SHORT). LUEGO CREAR OTRA CLASE CON LA QUE SE VA A GUARDAR ELEMENTOS DEL TIPO DE LA PRIMERA. O SEA COLECCIONES TIPADAS CON ELEMENTOS DEFINIDOS POR UNO MISMO, O SEA, QUE ADMITA ELEMENTOS CON TIPOS PERSONALIZADOS, SE PUEDEN AMPLIAR SUS FUNCIONALIDADES, PERO HAY QUE PROGRAMARLO

private void button1\_Click(object sender, EventArgs e)

{

//ClienteIcom claC = new ClienteIcom("j","br");

//MessageBox.Show(claC.ToString());

ClienteColl col = new ClienteColl();

col.Add(new ClienteIcom("Pepe", "López"));

col.Add(new ClienteIcom("Loli", "Pérez"));

col.Add(new ClienteIcom("Eva", "Kelo"));

col.Add(new ClienteIcom("Juan", "Salvador"));

col.Add(new ClienteIcom("Miguel", "Andrade"));

col.Sort();

for (int i = 0; i < col.Count; i++)

Console.WriteLine(col[i].Apellidos);

col.RemoveAt(2);

foreach (ClienteIcom cli in col)

MessageBox .Show (cli.ToString());

}

public class ClienteColl : System.Collections.CollectionBase

{

//METODO PARA AGREGAR OBJETOS DEL TIPO CLIENTEICOM

public int Add(ClienteIcom value)

{

return List.Add(value);

}

public bool Contains(ClienteIcom value)

{

return List.Contains(value);

}

public int IndexOf(ClienteIcom value)

{

return List.IndexOf(value);

}

//METODO PARA AGREGAR OBJETOS DEL TIPO CLIENTEICOM

public void Insert(int index, ClienteIcom value)

{

List.Insert(index, value);

}

public ClienteIcom this[int index]

{

get

{

return (ClienteIcom)List[index];

}

set

{

List[index] = value;

}

}

public void Remove(ClienteIcom value)

{

List.Remove(value);

}

public void Sort()

{

InnerList.Sort();

}

}

public class ClienteIcom : System.IComparable

{

public string Nombre;

public string Apellidos;

//SOBRECARGA DEL CONSTRUCTOR

public ClienteIcom(string nombre)

{

this.Nombre = nombre;

}

//2 SOBRECARGA DEL CONSTRUCTOR

public ClienteIcom(string nombre, string apellidos)

{

this.Nombre = nombre;

this.Apellidos = apellidos;

}

//SOBRE CARGA DEL METODO TOSTRING PARA QUE MANDE NOMBRE Y APELLIDOS

public override string ToString()

{

return Apellidos + ", " + Nombre;

}

//IMPLEMENTACION DEL METODO ORDENANDO POR APELLIDO

public int CompareTo(object obj)

{

if (obj is ClienteIcom)

{

ClienteIcom cli = (ClienteIcom)obj;

return string.Compare(this.Apellidos, cli.Apellidos);

}

else

return 0;

}

}

### Clase personalizada Dictionary Base

private void button2\_Click(object sender, EventArgs e)

{

Articulos col = new Articulos();

col.Add("uno", new Articulo("Uno", "Art. Uno", 10.6M));

col.Add("dos", new Articulo("Dos", "Art. Dos", 22));

col.Add("tres", new Articulo("Tres", "Art. Tres", 45.55M));

foreach (DictionaryEntry de in col)

{

Articulo art;

art = ((Articulo)de.Value);

MessageBox .Show (de.Key + " " + art.Descripcion + " " + art.PVP);

}

Console.WriteLine();

col.Remove("dos");

//

foreach (string s in col.Keys)

MessageBox.Show(s + " " + col[s].Descripcion);

}

CLASE QUE CREA LA COLECCION

public class Articulos : System.Collections.DictionaryBase

{

public Articulo this[string key]

{

get

{

return (Articulo)Dictionary[key];

}

set

{

Dictionary[key] = value;

}

}

public ICollection Keys

{

get

{

return Dictionary.Keys;

}

}

public ICollection Values

{

get

{

return Dictionary.Values;

}

}

public void Add(string key, Articulo value)

{

Dictionary.Add(key, value);

}

public bool Contains(string key)

{

return Dictionary.Contains(key);

}

public void Remove(string key)

{

Dictionary.Remove(key);

}

}

CLASE QUE DETERMINA EL TIPO

public class Articulo

{

public string Codigo;

public string Descripcion;

public decimal PVP;

//SOBRECARGA CONSTRUCTOR

public Articulo(string codigo, string descripcion, decimal precio)

{

this.Codigo = codigo;

this.Descripcion = descripcion;

this.PVP = precio;

}

//SOBRECARGA DEL METODO TOSTRING

public override string ToString()

{

return Codigo + ", " + Descripcion;

}

}

### Collection Base con Colecciones genéricas

POR EL INCONVENIENTE DE RENDIMIENTO DE LAS COLECCIONES QUE HEREDAN DIRECTAMENTE DE SYSTEM.COLLECTIONS, EXISTEN TAMBIEN OTRAS IMPLEMENTACIONES FUERTEMENTE TIPADAS QUE DERIVAN DE SYSTEM.COLLECTIONS.GENERIC, PARA LO CUAL SE LLAMA A ESTE ESPACIO DE NOMBRES.

C# PUEDE CREAR COLECCIONES TIPADAS SIN NECESIDAD DE CREAR UN CLASE ESPECÍFICA IMPLEMENTAN LAS CARACTERISTICAS BASICAS DE LAS COLECCIONES QUE HEREDAN, ESTAS SON LAS COLECCIONES GENÉRICAS.

EN GENERAL, ES RECOMENDABLE UTILIZAR COLECCIONES GENÉRICAS PORQUE SE OBTIENE LA VENTAJA INMEDIATA DE LA SEGURIDAD DE TIPOS SIN NECESIDAD DE DERIVAR DE UN TIPO DE COLECCIÓN BASE E IMPLEMENTAR MIEMBROS ESPECÍFICOS DE LOS TIPOS. ADEMÁS, POR LO GENERAL LOS TIPOS DE COLECCIÓN GENÉRICOS TIENEN UN MEJOR RENDIMIENTO QUE LOS TIPOS DE COLECCIÓN NO GENÉRICOS CORRESPONDIENTES (Y MEJOR QUE EL DE LOS TIPOS DERIVADOS DE TIPOS DE COLECCIÓN BASE NO GENÉRICOS) CUANDO LOS ELEMENTOS DE LA COLECCIÓN SON TIPOS DE VALOR, PORQUE CON LOS GENÉRICOS NO HAY NECESIDAD DE REALIZAR CONVERSIONES BOXING DE LOS ELEMENTOS.

\*\*\*\*UTILIZA LA CLASE ClienteIcom DE COLLECTION BASE\*\*\*\*\*\*\*\*\*

private void button3\_Click(object sender, EventArgs e)

{

MessageBox .Show ("Ejemplo usando Generic.List");

List<ClienteIcom> colCli = new List<ClienteIcom>();

colCli.Add(new ClienteIcom("Pepe", "López"));

colCli.Add(new ClienteIcom("Loli", "Pérez"));

colCli.Add(new ClienteIcom("Eva", "Kelo"));

colCli.Add(new ClienteIcom("Juan", "Salvador"));

colCli.Add(new ClienteIcom("Miguel", "Andrade"));

colCli.Sort();

for (int i = 0; i < colCli.Count; i++)

MessageBox.Show(colCli[i].Apellidos);

Console.WriteLine();

// Elimina el elemento de la posición 3

// (pero después de haberlo clasificado)

colCli.RemoveAt(2);

foreach (ClienteIcom cli in colCli)

MessageBox.Show(cli.ToString());

}

### Dictionary Base con Colecciones genéricas

\*\*\*\*UTILIZA LA CLASE Articulo DE DICTIONARY BASE\*\*\*\*\*\*\*\*\*

private void button4\_Click(object sender, EventArgs e)

{

MessageBox .Show ("Ejemplo usando Generic.Dictionary");

Dictionary<string, Articulo> colArt = new Dictionary<string, Articulo>();

colArt.Add("uno", new Articulo("Uno", "Art. Uno", 10.6M));

colArt.Add("dos", new Articulo("Dos", "Art. Dos", 22));

colArt.Add("tres", new Articulo("Tres", "Art. Tres", 45.55M));

foreach (KeyValuePair<string, Articulo> de in colArt)

{

Articulo art = de.Value;

MessageBox.Show( de.Key + " " + art.Descripcion + " " + art.PVP);

}

Console.WriteLine();

colArt.Remove("dos");

foreach (string s in colArt.Keys)

MessageBox.Show(s + " " + colArt[s].Descripcion);

}

### Mas sobre colecciones genéricas

POR EJEMPLO SI HAY UNA CLASE BASE Y DOS QUE HEREDAN DE ELLA, SI SE IMPLEMENTA LA COLECCION GENERICA SE PUEDEN AGREGAR ELEMENTOS DE LOS TIPOS QUE IMPLEMENTAN LA CLASE BASE

Private void button5\_Click(object sender, EventArgs e)

{

//SE PUEDEN AGREGAR TIPOS CLIENTE Y EMPLEADO PORQUE AMBOS HEREDAN DE PERSONA

List<Persona> col = new List<Persona>();

col.Add(new Cliente("Construcciones Pepe"));

col.Add(new Empleado("Juan", "Pérez"));

foreach (Persona p in col)

MessageBox .Show (p.Nombre);

}

//CLASE BASE PERSONA

public class Persona

{

private string nombre;

public string Nombre

{

get

{

return nombre;

}

set

{

nombre = value;

}

}

private string apellidos;

public string Apellidos

{

get

{

return apellidos;

}

set

{

apellidos = value;

}

}

}

//CLIENTE QUE HEREDA DE PERSONA Y UTILIZA NOMBRE

public class Cliente : Persona

{

public decimal Deuda;

public Cliente(string empresa)

{

this.Nombre = empresa;

}

public override string ToString()

{

return Nombre;

}

}

//CLASE EMPLEADO QUE UTILIZA NOMBRE Y APELLIDO DE PERSONA

public class Empleado : Persona

{

public decimal Salario;

public Empleado(string nombre, string apellidos)

{

this.Nombre = nombre;

this.Apellidos = apellidos;

}

public override string ToString()

{

return Apellidos + ", " + Nombre;

}

}

EN EL EJEMPLO SIGUIENTE SE MUESTRA CÓMO USAR LA CLASE [LIST<T>](http://msdn.microsoft.com/es-es/library/6sh2ey19(v=vs.90).aspx). OBSERVE QUE A DIFERENCIA DE LA CLASE [ARRAY](http://msdn.microsoft.com/es-es/library/system.array(v=vs.90).aspx), LOS ELEMENTOS SE PUEDEN INSERTAR EN MEDIO DE LA LISTA. ESTE EJEMPLO RESTRINGE LOS ELEMENTOS DE LA LISTA A FIN DE QUE SEAN CADENAS.

public static void TestList()

{

List<string> sandwich = new List<string>();

sandwich.Add("bacon");

sandwich.Add("tomato");

sandwich.Insert(1, "lettuce");

foreach (string ingredient in sandwich)

{

System.Console.WriteLine(ingredient);

}

}

## Streams

using System.IO ;

### File Stream

1. **Guardar datos**

private void button1\_Click(object sender, EventArgs e)

{

//LLAMA AL LA CLASE FILESTREAM. SI YA EXISTE EL ARCHIVO LO ABRE, SI NO LO CREA

//AGREGANDO LA CADENA AL INICIO DEL MISMO

guardarDatos(@"c:\avance1.txt","hola con stream");

}

static void guardarDatos(string fichero, string cadena)

{

// Abrimos o creamos el fichero, para escribir en él

System.IO.FileStream fs = new System.IO.FileStream(fichero,

System.IO.FileMode.OpenOrCreate,

System.IO.FileAccess.Write);

// Escribimos algunas cadenas,

// el problema es que solo podemos escribir arrays de bytes,

// por tanto debemos convertir la cadena en un array de bytes

byte[] datos;

// pero usando la codificación que creamos conveniente

// de forma predeterminada es UTF-8

System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();

// convertimos la cadena en un array de bytes

datos = enc.GetBytes(cadena);

// lo escribimos en el stream

fs.Write(datos, 0, datos.Length);

// nos aseguramos que se escriben todos los datos

fs.Flush();

// cerramos el stream

fs.Close();

}

1. **Leer datos**

private void button2\_Click(object sender, EventArgs e)

{

this.textBox1 .Text =leerDatos(@"c:\avance1.txt");

}

static string leerDatos(string fichero)

{

// Los bloques leídos los almacenaremos en un StringBuilder

System.Text.StringBuilder res = new System.Text.StringBuilder();

// Abrimos el fichero para leer de él

System.IO.FileStream fs = new System.IO.FileStream(fichero,

System.IO.FileMode.Open,

System.IO.FileAccess.Read);

// los datos se leerán en bloques de 1024 bytes (1 KB)

byte[] datos = new byte[1025];

System.Text.UTF8Encoding enc = new System.Text.UTF8Encoding();

// Para usar la codificación de Windows

//System.Text.Encoding enc = System.Text.Encoding.Default;

// leemos mientras hay algo en el fichero

while (fs.Read(datos, 0, 1024) > 0)

{

// agregamos al stringBuilder los bytes leídos

// (convertidos en una cadena)

res.Append(enc.GetString(datos));

}

// cerramos el buffer

fs.Close();

// devolvemos todo lo leído

return res.ToString();

}

### Stream writer

private void button3\_Click(object sender, EventArgs e)

{

guardarDatosEsp("c:\avance1.txt","con clases especializadas");

}

Static void guardarDatosEsp(string fichero, string cadena)

{

// ABRIMOS EL FICHERO PARA ESCRIBIR, (NO AÑADIR),

// USANDO LA CODIFICACIÓN PREDETERMINADA: UTF-8

System.IO.StreamWriter sw = new System.IO.StreamWriter(fichero, false);

//

//System.IO.StreamWriter sw = new System.IO.StreamWriter(fichero,

// false,

// System.Text.Encoding.Default);

//

// guardamos toda la cadena

sw.WriteLine(cadena);

// Cerramos el fichero

sw.Close();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//OTRO EJEMPLO DE STREARMWRITER DONDE MANDA UNA CADENA MAS ELABORADA, TODO

//DENTRO UNA TRANSACCION USING Y CON UN MENSAJE AL FINAL. ESTE ARCHIVO SE GUARDA EN LA

//CARPETA BIN/DEBUG, PERO PODRIA ESCRIBIRSE LA RUTA COMPLETA

//using (StreamWriter writer = new StreamWriter("Members.txt"))

//{

// writer.WriteLine("First Name: {0}", firstname.Text);

// writer.WriteLine("Last Name: {0}", lastname.Text);

// writer.WriteLine("Tower: {0}", towername.Text);

// writer.WriteLine("Captain: {0}", iscaptain.IsChecked.ToString());

// writer.WriteLine("Member Since: {0}", membersince.Text);

// writer.WriteLine("Methods: ");

// foreach (CheckBox cb in methods.Items)

// {

// if (cb.IsChecked.Value)

// writer.WriteLine(cb.Content.ToString());

// }

// MessageBox.Show("member details salvado", "saved");

//}

}

### Stream reader

private void button4\_Click(object sender, EventArgs e)

{

leerDatosEsp("c:\avance1.txt");

}

static string leerDatosEsp(string fichero)

{

// ABRIMOS EL FICHERO PARA LEER

// USANDO LA CODIFICACIÓN UTF-8 (LA PREDETERMINADA DE .NET)

System.IO.StreamReader sr = new System.IO.StreamReader(fichero, true);

//

// SI QUEREMOS USAR LA PREDETERMINADA DE WINDOWS

//System.IO.StreamReader s=new System.IO.StreamReader(fichero, System.Text.Encoding.Default);

//

// PODEMOS LEER CADA UNA DE LAS LÍNEAS DEL FICHERO O TODO EL CONTENIDO FORMA LARGA:

// SI VAMOS A LEER EL FICHERO LÍNEA POR LÍNEA, MEJOR USAR UN STRINGBUILDER

System.Text.StringBuilder ret = new System.Text.StringBuilder();

// recorremos el fichero hasta que no haya nada que leer

while (sr.Peek() != -1)

{

ret.Append(sr.ReadLine());

}

// cerramos el fichero

sr.Close();

// devolvemos lo leído

return ret.ToString();

//

// // Forma corta:

// // leemos todo el contenido del fichero

// string ret = sr.ReadToEnd();

// // lo cerramos

// sr.Close();

// // devolvemos lo leído

// return ret;

}

### Open file dialog

private void button5\_Click(object sender, EventArgs e)

{

openFileDialog = new OpenFileDialog();

openFileDialog.FileOk += openFileDialogFileOk;

openFileDialog.ShowDialog();

}

private void openFileDialogFileOk(object sender, System.ComponentModel.CancelEventArgs e)

{

string fullPathname = openFileDialog.FileName;

FileInfo src = new FileInfo(fullPathname);

filename.Text = src.Name;

textBox1.Text = "";

TextReader reader = src.OpenText();

string line = reader.ReadLine();

while (line != null)

{

textBox1.Text += line + '\n';

line = reader.ReadLine();

}

reader.Close();

}

### Save file dialog

private void button6\_Click(object sender, EventArgs e)

{

//utilizando la clase save dialog, using Microsoft.Win32;

//utilizando la clase save dialog

SaveFileDialog SD = new SaveFileDialog();

SD.DefaultExt = "txt";

SD.AddExtension = true;

//propone un nombre por default

SD.FileName = "Members";

SD.InitialDirectory = @"c:";

SD.OverwritePrompt = true;

SD.Title = "Campanas";

SD.ValidateNames = true;

SD.ShowDialog();

{

using (StreamWriter writer = new StreamWriter(SD.FileName))

{

writer.WriteLine("First Name: {0}", "");

writer.WriteLine("Last Name: {0}", "");

writer.WriteLine("Tower: {0}", "");

writer.WriteLine("Captain: {0}", "");

writer.WriteLine("Member Since: {0}", "");

writer.WriteLine("Methods: ");

}

}

}

## Genericos

CUANDO SE UTILIZA LA CLASE OBJECTS PARA CREAR OBJETOS ESTO ES MUY FLEXIBLE, PERO DEJA UN HUECO IMPORTANTE QUE DE NO SER BIEN ANALIZADO PUEDE OCASIONAR ERRORES EN TIEMPO DE EJECUCION. CUANDO SE UTILIZAN OBJETOS TIPO INT ADEMAS SE CONSUMEN MAS RECURSOS DEL SISTEMA PORQUE ES NECESARIO REALIZAR BOXING UNBOXING

static void Main(string[] args)

{

//ejemplo: usamos una coleccion queue en la cual guardamos elementos tipo object (clase cualquiera)

Queue myQueue = new Queue();

claClase myClase = new claClase();

//mete en la cola de la coleccion un objeto tipo myClase

myQueue.Enqueue(myClase);

//saca de la cola el elemento, solo que para hacerlo tiene que hacer casting para asegurarse que el

//elemento que va a sacar es del tipo apropiado. esto es un inconveniente de flexibilidad, ai no se

//hiciera daria un error en tiempo de ejecucion.

myClase = (claClase)myQueue.Dequeue();

//GENERICOS

//aqui indicamos que todo lo que este en la cola de myQueue sera del tipo claClase, por lo cual el

//cast sera inecesario. en este caso utilizamos la clase generica Queue

Queue<claClase> myQueueGen = new Queue<claClase>();

claClase MyClaseGen = new claClase();

myQueueGen.Enqueue(MyClaseGen);

MyClaseGen = myQueueGen.Dequeue();

//OJO: se pueden crar clases y metodos genericos \*\*\*

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ESTE APARTADO SE APOYA EN LA CLASE SIGUIENTE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class claClase

{

}

## Recursos locales

PARA ACCESAR A LOS RECURSOS LOCALES DEL PROYECTO HAY QUE DEFINIR UNA RUTA RELATIVA CON PATH, LA CUAL FUNCIONA CON EL Using System.IO. POR EJEMPLO UTILIZAR UNA IMAGEN QUE SE ENCUENTRA EN EL PROYECTO.

Bitmap imagen = new Bitmap(Path.Combine(Application.StartupPath, @"\BOTON.jpg"));

button1.BackgroundImage = imagen;

## Conversion de tipos

CUANDO SE REQUIERE POR EJEMPLO MANDAR DATOS TIPO STRING A UNA CLASE VINCULADA QUE RECIBE DATOS ENUM, SE UTILIZA vALUEcONVERSION. PRIMERO SE USA IMPORTS: ver el ejercicio completo en el proyecto tickets, implica un método mas para la conversión inversa.

using System.Windows.Data;

[ValueConversion(typeof(string), typeof(PrivilegeLevel))]

public class PrivilegeLevelConverter: IValueConverter

{

public object Convert(object value, Type targetType, object parameter,

System.Globalization.CultureInfo culture)

{

PrivilegeLevel privilegeLevel = (PrivilegeLevel)value;

string convertedPrivilegeLevel = string.Empty;

switch (privilegeLevel)

case PrivilegeLevel.Standard:

convertedPrivilegeLevel = "Standard";

break;

case PrivilegeLevel.Premium:

convertedPrivilegeLevel = "Premium";

break;

case PrivilegeLevel.Excecutive:

convertedPrivilegeLevel = "Excecutive";

break;

case PrivilegeLevel.PremiumExcecutive:

convertedPrivilegeLevel = "Premium Excecutive";

break;

}

return convertedPrivilegeLevel;

## Capturar el pulsado de teclas

switch (e.Key)

{

case Key.Enter: "hace algo al pulsar enter";

break;

case Key.Insert: "hace algo al pulsar insert";

break;

case Key.Delete: "hace algo al pulsar delete";

break;

}

# Desconectados con componentes

## Componentes configurados en t.d.

ESTA ES UNA FORMA SEGURA DE HACER MODIFICACIONES EN LOS DATOS, PUES AL PASARSE POR PARÁMETROS ES MAS SEGURO QUE EN LA INSTRUCCION SQL DIRECTA, EN LA CUAL PODRIAN INYECTAR SQL MALICIOSO, LA DESVENTAJA ES QUE ES LABORIOSO ESTAR CREANDO LAS INSTRUCCIONES DEL COMMAND PARA LOS DISTINTOS COMANDOS**. LOS COMPONENTES DE DATOS SE AGREGAN Y CONFIGURAN AL PRINCIPIO: SQLCONNECTION, SQLDATAADAPTER, DATASET Y SQLCOMMAND**

private void frmComNet\_Load(object sender, EventArgs e)

{

sqlDataAdapter1.SelectCommand.CommandText = ("select \* from municipios");

POBLAMOS EL DATASET CON LOS RESULTADOS OBTENIDOS DE LA INSTRUCCION SQL

sqlDataAdapter1.Fill(dataSet1, "municipios");

OJO: ES DIFERENTE INVOCAR UN METODO () A []

this.dataGridView1.DataSource = dataSet1.Tables["municipios"];

### Insert

DataRow MiFila;

//CREAMOS UNA NUEVA FILA

MiFila = dataSet1.Tables["municipios"].NewRow();

MiFila["municipio"] = this.textBox2.Text;

//AÑADIMOS LA FILA NUEVA A LA TABLA

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//HASTA AQUI ES UTIL PARA INSERTAR DATOS TEMPORALMENTE EN EL GRID

dataSet1.Tables["municipios"].Rows.Add(MiFila);

MIRAMOS SI EL DATASET HA CAMBIADO

if (dataSet1.HasChanges())

{

//CREAMOS EL COMANDO DE INSERCION

sqlCommand1.CommandText = "INSERT INTO municipios(municipio) VALUES(@municipio)";

sqlCommand1.Connection = sqlConnection1;

sqlDataAdapter1.InsertCommand = sqlCommand1;

//AÑADIMOS LOS PARAMETROS

sqlCommand1.Parameters.Add("@municipio", SqlDbType.NChar, 100, "municipio");

sqlConnection1.Open();

//ACTUALIZAMOS EL DATASET CONTRA LA BASE DE DATOS

sqlDataAdapter1.Update(dataSet1, "municipios");

sqlConnection1.Close();

MessageBox.Show(" Cambios realizados correctamente !");

### Update

private void button4\_Click(object sender, EventArgs e)

{

BUSCAR LA FORMA DE MODIFICAR EL REGISTRO ACTUAL. ES IMPORTANTE MENCIONAR QUE EL GRID DETECTA CAMBIOS EN SUS REGISTROS, ASI QUE SI CAMBIAMOS ALGO DEL GRID FUNCIONA EL UPDATE

dataSet1.Tables["municipios"].Rows[2][1] = "modificado";

//MIRAMOS SI EL DATASET HA CAMBIADO

if (dataSet1.HasChanges())

{

//CREAMOS EL COMANDO DE ACTUALIZACION

sqlCommand1.CommandText = "UPDATE municipios SET municipio=@municipio WHERE id=@id";

sqlCommand1.Connection = sqlConnection1;

sqlDataAdapter1.UpdateCommand = sqlCommand1;

//AÑADIMOS LOS PARAMETROS

sqlCommand1.Parameters.Add("@id", SqlDbType.Int, 10, "id");

sqlCommand1.Parameters.Add("@municipio", SqlDbType.NVarChar, 50, "municipio");

sqlConnection1.Open();

//ACTUALIZAMOS EL DATASET CONTRA LA BASE DE DATOS

sqlDataAdapter1.Update(dataSet1, "municipios");

sqlConnection1.Close();

MessageBox.Show(" Cambios realizados correctamente !");

}

### Delete

private void button5\_Click(object sender, EventArgs e)

{

//MODIFICAMOS LA FILA 1 COLUMNA 4 ( INDICES COMENZANDO EN 0 )

dataSet1.Tables["municipios"].Rows[7].Delete();

//MIRAMOS SI EL DATASET HA CAMBIADO

if (dataSet1.HasChanges())

{

//CREAMOS EL COMANDO DE ELIMINQACION

sqlCommand1.CommandText = "DELETE from municipios WHERE id=@id";

sqlCommand1.Connection = sqlConnection1;

sqlDataAdapter1.DeleteCommand = sqlCommand1;

//AÑADIMOS LOS PARAMETROS

sqlCommand1.Parameters.Add("@id", SqlDbType.Int, 10, "id");

sqlConnection1.Open();

//ACTUALIZAMOS EL DATASET CONTRA LA BASE DE DATOS

sqlDataAdapter1.Update(dataSet1, "municipios");

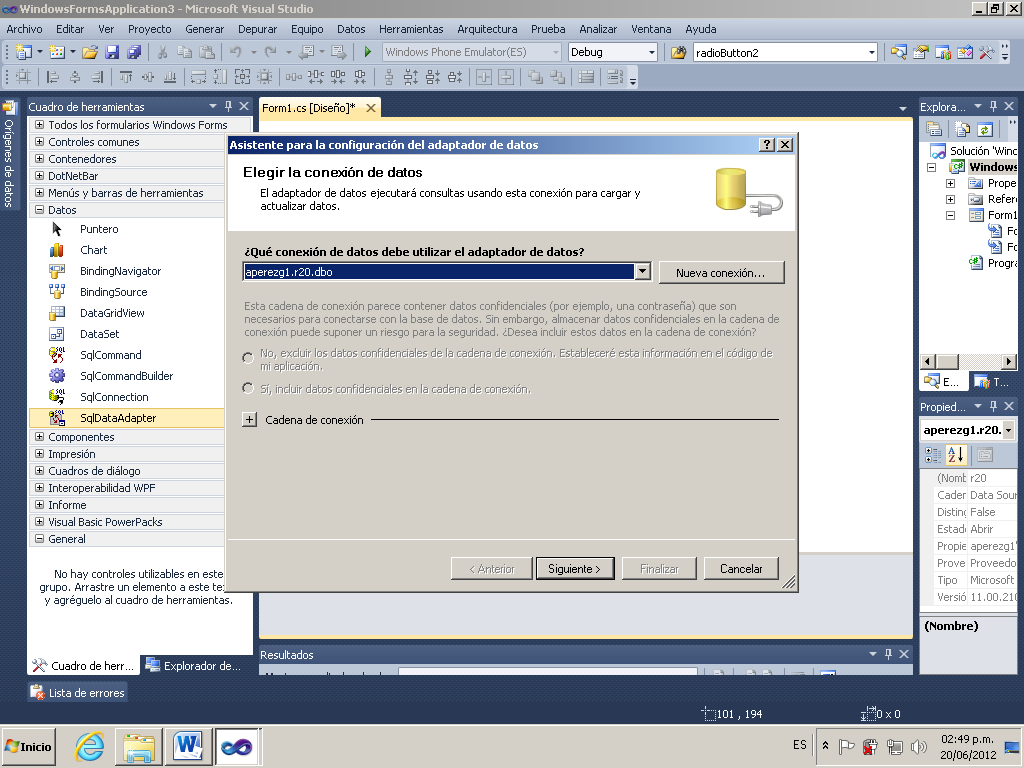
sqlConnection1.Close();

MessageBox.Show(" Cambios realizados correctamente !");

## Automatizacion del data adapter

ESTA TECNICA ES LA MAS PRACTICA EN CUANTO A AHORRO DE CODIGO, AUNQUE SU USO ES PARA TAREAS ESTANDAR QUE NO SEAN UY ELABORARDAS. SU VENTAJA ES QUE AL CONFIGURAR EL ADAPTER CON EL ASISTENTE, SE CREAN AUTOMATICAMENTE LOS COMANDOS DE INSERT, UPDATE..., Y AL MISMO TIEMPO EFICIENTIZA EL CODIGO EN TODO EL SENTIDO DE LA PALABRA

SE AGREGAN LOS OBJETOS DE DATOS SQL A LA BARRA DE HERRAMIENTAS, SE AGREGA EL DATAADAPTER Y SE CONFIGURAN SUS PROPIEDADES: PRIMERO EN SELECTCOMMAND->CONNECTION Y SELECCIONE LA CONEXIÓN EXISTENTE Y ASI PARA CADA UNO DE LOS 4 COMANDOS. TAMBIEN SE PUEDE CONFIGURAR CON EL ASISTENTE QUE APARECE CUANDO SE ARRASTRA AL FORMULARIO.



private void frmAutDatAdap\_Load(object sender, EventArgs e)

{

sqlDataAdapter1.Fill(dataSet1, "municipios");

dataGridView1 .DataSource =dataSet1 .Tables ["municipios"];

### Insert

private void button1\_Click(object sender, EventArgs e)

{

DataRow rowFila = dataSet1.Tables["municipios"].NewRow();

rowFila["municipio"] = textBox1.Text;

dataSet1.Tables["municipios"].Rows.Add(rowFila );

sqlDataAdapter1.Update(dataSet1, "municipios");

MessageBox.Show("la fila se ha insertado correctamente");

}

### Update

private void button2\_Click(object sender, EventArgs e)

{

dataSet1 .Tables ["municipios"].Rows[5][1]="modificado";

sqlDataAdapter1.Update(dataSet1, "municipios");

MessageBox.Show("la fila se ha insertado correctamente");

}

### Delete

private void button3\_Click(object sender, EventArgs e)

{

dataSet1.Tables["municipios"].Rows[5].Delete ();

sqlDataAdapter1.Update(dataSet1, "municipios");

MessageBox.Show("la fila se ha insertado correctamente");

}

# Desconectados con codigo manual

## Acceso a colecciones del dataset

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

string strCom1 = "select \* from localidades";

adaM = new SqlCeDataAdapter(strCom, strConex);

//UN SEGUNDO ADAPTER PARA LA OTRA TABLA

SqlCeDataAdapter adaM1 = new SqlCeDataAdapter();

adaM1= new SqlCeDataAdapter(strCom1, strConex);

adaM.Fill(datM, "municipios");

adaM1.Fill(datM, "localidades");

//COLECCION DE TABLAS

foreach (DataTable dT in datM.Tables)

{

MessageBox.Show("Las tablas en el dataset son :" +dT.ToString ());

}

//COLECCION DE COLUMNAS

DataColumnCollection dCC;

dCC = datM.Tables[0].Columns;

foreach (DataColumn dC in dCC )

{

MessageBox.Show("las columnas de la primera tablas son :" + dC.ColumnName.ToString ());

}

//TABLA MUNICIPIOS

foreach (DataRow dR in datM.Tables[0].Rows)

{

MessageBox.Show("los registros de la primera tabla son :"+ dR[0].ToString () +" " + dR[1].ToString ());

}

//TABLA LOCALIDADES ALTERNATIVAMENTE, LA COLECCION POR SU NOMBRE O POR SU INDEX

foreach (DataRow dR in datM.Tables["localidades"].Rows)

{

MessageBox.Show("los registros de la segunda tabla son :" + dR["idlocalidad"].ToString() +

" " + dR["idmunicipio"].ToString() + " " + dR[2].ToString());

}

## Command builder

LA FORMA NORMAL ES CONFIGURAR LOS CUATRO COMANDOS PARA EL ADAPTER, LO CUAL PROPORCIONA LA FLEXIBILIDAD TOTAL PERO EXISTE UNA CLASE QUE FACILITA ESTA OPERACION MEDIANTE LA CUAL NO ES NECESARIO CREAR LOS CUATRO COMANDOS EL COMMANDBUILDER, SINO SOLAMENTE EL DE SELECCIÓN. AUNQUE SE UTILIZA SOLO PARA ACCIONES BASICAS, SENTENCIAS ESTANDAR, NO PA

USAMOS LA SIGUIENTE CADENA DE CONEXIÓN Y POBLADO DEL DATASET

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strComm = "select idmunicipio, municipio from municipios";

SqlCeDataAdapter adaCeBuil=new SqlCeDataAdapter (strComm,strConex );

SqlCeCommandBuilder builCe = new SqlCeCommandBuilder(adaCeBuil);

DataSet datMun=new DataSet ();

adaCeBuil.Fill(datMun, "municipios");

### insert

//agrega nueva fila

DataRow fil = datMun.Tables["municipios"].NewRow();

//fil["idmunicipio"] = 9;

fil["municipio"] = "mpio\_builder";

datMun.Tables["municipios"].Rows.Add(fil);

//visualiza en grid

griPadre .DataSource =datMun .Tables ["municipios"];

//sincroniza

adaCeBuil .Update(datMun , "municipios");

datMun.Tables["municipios"].AcceptChanges();

MessageBox.Show("registro efectuado");

### Update

//elimina nueva fila

DataRow filFila;

filFila = datMun.Tables["municipios"].Rows[8];

filFila["municipio"] = "mpio\_modificado\_builder";

//visualiza en grid

griPadre.DataSource = datMun.Tables["municipios"];

//sincroniza

adaCeBuil.Update(datMun, "municipios");

aatMun.Tables["municipios"].AcceptChanges();

MessageBox.Show("registro modificado");

### Delete

//elimina nueva fila

DataRow filFila;

filFila = datMun.Tables["municipios"].Rows[9];

filFila.Delete();

//visualiza en grid

griPadre.DataSource = datMun.Tables["municipios"];

//sincroniza

adaCeBuil.Update(datMun, "municipios");

datMun.Tables["municipios"].AcceptChanges();

MessageBox.Show("registro eliminado");

## 4 comandos del data adapter

ESTA ES LA FORMA MAS FLEXIBLE DE ACCESO A DATOS EN EL ORIGEN, PERO IMPLICA MUCHO CODIGO AL TENER QUE CONTRUIR LOS COMANDOS A MANO.

UTILIZAMOS LA SIGUIENTE CADENA DE CONEXION Y POBLADO DEL DATASET

SqlCeCommand commMx=new SqlCeCommand ();

SqlCeDataAdapter adaMx=new SqlCeDataAdapter ();

SqlCeConnection connMx = new SqlCeConnection(@"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False");

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, connMx.ConnectionString );

adaM.Fill(datM, "municipios");

### Insert

//INSERTAR NUEVO REGISTRO

DataRow MiFila;

//Creamos una nueva fila

MiFila = datM.Tables["municipios"].NewRow();

MiFila["municipio"] = this.textBox3.Text;

//Añadimos la fila nueva a la tabla, hasta aqui es util para insertar datos temporalmente en el grid

datM.Tables["municipios"].Rows.Add(MiFila);

//Miramos si el dataSet ha cambiado

if (datM.HasChanges())

{

//Creamos el comando de insercion

commMx.CommandText = "INSERT INTO municipios(municipio) VALUES(@municipio)";

commMx.Connection = connMx;

adaM.InsertCommand = commMx;

//Añadimos los parametros

commMx.Parameters.Add("@municipio", SqlDbType.NChar, 100, "municipio");

connMx.Open();

//Actualizamos el dataSet contra la base de datos

adaM.Update(datM, "municipios");

//connMx.Close();

MessageBox.Show(" Cambios realizados correctamente !");

}

### Update

//BUSCAR LA FORMA DE MODIFICAR EL REGISTRO ACTUAL

datM.Tables["municipios"].Rows[5][1] = textBox3 .Text ;

//MIRAMOS SI EL DATASET HA CAMBIADO

if (datM.HasChanges())

{

//CREAMOS EL COMANDO DE ACTUALIZACION

commMx.CommandText = "UPDATE municipios SET municipio=@municipio WHERE idmunicipio=@idmunicipio";

commMx.Connection = connMx;

adaMx.UpdateCommand = commMx;

//AÑADIMOS LOS PARAMETROS

commMx.Parameters.Add("@idmunicipio", SqlDbType.Int, 10, "idmunicipio");

commMx.Parameters.Add("@municipio", SqlDbType.NVarChar, 50, "municipio");

connMx.Open();

//ACTUALIZAMOS EL DATASET CONTRA LA BASE DE DATOS

adaMx.Update(datM, "municipios");

connMx.Close();

MessageBox.Show(" Cambios realizados correctamente !");

}

### Delete

//MODIFICAMOS LA FILA 1 COLUMNA 4 ( INDICES COMENZANDO EN 0 )

datM.Tables["municipios"].Rows[7].Delete();

//MIRAMOS SI EL DATASET HA CAMBIADO

if (datM.HasChanges())

{

//CREAMOS EL COMANDO DE INSERCION

commMx.CommandText = "DELETE from municipios WHERE idmunicipio=@idmunicipio";

commMx.Connection = connMx ;

adaMx.DeleteCommand = commMx;

//AÑADIMOS LOS PARAMETROS

commMx.Parameters.Add("@idmunicipio", SqlDbType.Int, 10, "idmunicipio");

connMx.Open();

//ACTUALIZAMOS EL DATASET CONTRA LA BASE DE DATOS

adaMx.Update(datM, "municipios");

connMx.Close();

MessageBox.Show(" Cambios realizados correctamente !");

}

## Factoria de datos

LA FACORIAS DE DATOS PERMITEN CREAR OBJETOS GENERICOS QUE DEPENDAN EN UN MINIMO DE PROVEEDORES ESPECIFICOS. ASI, EN CASO DE MIGRACION, SOLAMENTE SE CAMBIARAN ALGUNAS LINEAS. LO UNICO QUE NECESITAMOS ES IMPORTAR SYSTEM.DATA.COMMON

//SOLO HABRIA QUE CAMBIAR ESTA LINEA

DbProviderFactory dpfFactoria = DbProviderFactories.GetFactory("System.Data.SqlClient");

DbConnectionStringBuilder parConexion = dpfFactoria.CreateConnectionStringBuilder();

//Y ESTA LINEA

parConexion.Add("Data Source",@"aperezg1\sqlexpress");

parConexion.Add("Initial Catalog", "bd");

parConexion.Add("Integrated Security", "SSPI");

DbConnection Conexion = dpfFactoria.CreateConnection();

Conexion.ConnectionString = parConexion.ConnectionString;

DbDataAdapter Adapter = dpfFactoria.CreateDataAdapter();

DbCommand Comando = dpfFactoria.CreateCommand();

Comando.CommandText = "select \* from municipios";

Comando.Connection = Conexion;

Adapter.SelectCommand = Comando ;

DataSet Dataset = new DataSet();

Adapter.Fill(Dataset, "municipios");

griPadre .DataSource=Dataset.Tables ["municipios"];

//creando los commandbuilder genericos para modificaciones en la base

//DbCommandBuilder builComando = dpfFactoria.CreateCommandBuilder();

//builComando.DataAdapter = Adapter;

//Adapter.Update(Dataset, "municipios");

# Conectados con codigo manual

PRIMERO CREAMOS EL USING PARA DATA Y SQLDATA

using System.Data;

using System.Data.SqlClient;

## Modo consola

### Conexiones

ES IMPORTANTE MENCIONAR QUE AL FINAL SE DEBE CERRAR LA CONEXIÓN, DE PREFERENCIA EN LA INSTRUCCIÓN FINALLY, YA QUE ASI SE CERRARA AUNQUE OCURRA UN ERROR.UNA ALTERNATIVA ES UTILIZAR UNA INSTRUCCIÓN USING EN LA CUAL SE AGRUPE EL CÓDIGO EN TORNO A LA CREACIÓN DE UNA ESTRUCTURA USING QUE AL FINAL DEL MISMO SE CERRARA.

//using (SqlConnection dataConection = new SqlConnection())

//{

//Todo el codigo siguiente aqui

//}

OTRA ALTERNATIVA INTERESANTE ES UTILIZAR UNA SOBRECARGA DEL DATAREADER LA CUAL ORIGINA QUE CUANDO ESTE SE DEJA DE USAR LA CONEXIÓN SE CIERRA

//SqlDataReader dataReader = dataCommand.ExecuteReader(System.Data.CommandBehavior.CloseConnection);

OJO: EXISTE LA TECNOLOGÍA MARS QUE PERMITE CREAR VARIOS DATAREADER CONTRA LA MISMA CONEXIÓN, PERO HAY QUE HABILITARLO (SQL SERVER2008)

UTILIZANDO STRING BUILDER PARA CREAR LA CADENA DE CONEXIÓN, OJO CON LAS DIFERENCIAS SEGÚN EL SERVIDOR SEA CON NOMBRE O NO:

SqlConnectionStringBuilder strBuilder = new SqlConnectionStringBuilder();

strBuilder.DataSource = @"aperezg1";

strBuilder.InitialCatalog = "Northwind";

strBuilder.IntegratedSecurity = true;

dataConection.ConnectionString = strBuilder.ConnectionString;

dataConection.Open();

### Objetos de datos modo seguro (con parametros)

Console.Write("introduzca el numero de cliente;");

string noCliente = Console.ReadLine();

SqlCommand dataCommand = new SqlCommand();

dataCommand.Connection = dataConection;

dataCommand.CommandType = CommandType.Text;

dataCommand.CommandText = "select \* from orders where customerid=@cusisparam";

SqlParameter param = new SqlParameter("@cusisparam", SqlDbType.Char,5);

param.Value = noCliente;

dataCommand.Parameters.Add (param);

SqlDataReader dataReader = dataCommand.ExecuteReader();

### Acceso a datos

SE PUEDE ACCESAR A LOS ELEMENTOS DESDE UN ELEMENTO EN PARTICULAR O DESDE SU ÍNDICE EN EL MODO COLECCIÓN, SOLO QUE ESTE MÉTODO REQUIERE QUE EL READER UTILICE SU MÉTODO GET MAS EL TIPO DE DATO QUE VA A MANDAR JUNTO CON EL NUMERO DE INDICE

while(dataReader.Read())

{

Console.WriteLine(dataReader["orderid"].ToString());

Console.WriteLine(dataReader.GetInt32(0).ToString());

}

. . .

Finally

{

dataConection.Close();

## Windows forms

### Conexiones

1. **Conexiones a servidor**

//SE CREA LA CADENA DE CONEXION MANDANDO LOS DATOS A SU CONSTRUCTOR, EN UNA SOLA LINEA

SqlConnection conConexion = new SqlConnection(@"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true");

//SE CREA EL COMANDO MANDANDO A SU CONSTRUCTOR LA SENTENCIA SQL

SqlCommand commComando=new SqlCommand ("select \* from municipios",conConexion);

//SE CREA EL READER

SqlDataReader reaReader;

//ABRE LA CONEXION

conConexion.Open();

//EJECUTA EL COMANDO CONTRA EL READER EN SUS DIVERSAS OPCIONES

reaReader = commComando.ExecuteReader();

//commComando.ExecuteScalar();

//commComando.ExecuteNonQuery();

MessageBox.Show(conConex.State.ToString ());

//realiza las acciones sobre el reader, por ejemplo

DataTable tabTabla = new DataTable();

tabTabla.Load(reaReader, LoadOption.OverwriteChanges);

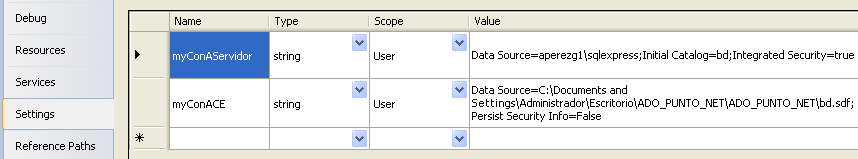
dataGridView1.DataSource = tabTabla ;

1. **Conexiones a servidor registradas en las propiedades del proyecto**

//EN LAS PROPIEDADES DEL PROYECTO, CONFIGURACIONES, AGREGAMOS LA CADENA DE CONEXION, LO CUAL LA REGISTRARA EN //APP.CONFIG. IMPORTAMOS EN ESPACIO DE NOMBRES USING ADO\_PUNTO\_NET.PROPERTIES (O COMO SE LLAME EL PROYECTO), //PARA ASI PODER LLAMAR A LA CONEXION DE FORMA SIMPLIFICADA. ASI TAMBIÉN SE CONFIGURA PARA LAS CONEXIONES A

//SQLSERVER COMPACT EDITION CE, COMO ES EL CASO DE MYCONACE.

using ADO\_PUNTO\_NET.Properties;



string strConex = Settings.Default.myConAServidor;

conConex = new SqlConnection(strConex);

conConex.Open();

**(2.1) Conexiones para CE**

//SE UTILIZA LA REFERENCIA USING SYSTEM.DATA.SQLSERVERCE Y LUEGO EL IMPORTS LUEGO LOS OBJETOS COMMAND, //CONNECTION, ETC. SON CE. SI NO SE ENCUENTRA LA REFERENCIA SE BUSCA EL ARCHIVO SQLSERVERCE.DLL EN EL DISCO //DURO

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\C#\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

conConexCe = new SqlCeConnection();

conConexCe.ConnectionString = strConex;

conConexCe.Open();

**(2.1.1) Datos generales de la conexion**

string strConexCE = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

conConexCe = new SqlCeConnection();

conConexCe.ConnectionString = strConexCE;

conConexCe.Open();

MessageBox.Show(conConexCe.ConnectionString.ToString() +"\n"+

conConexCe.Database.ToString() + "\n" +

conConexCe.ServerVersion.ToString() + "\n" +

conConexCe.DataSource.ToString() + "\n" +

conConexCe.ConnectionTimeout.ToString());

conConexCe.Close();

### Manejo de datos con data reader

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom,conConexCe);

//ojo el reader no tiene constructor SqlCeDataReader reaR;

conConexCe.Open();

reaRCE = commC.ExecuteReader();

while (reaRCE.Read())

{

MessageBox.Show(reaRCE ["municipio"].ToString() );

}

conConexCe.Close();

reaRCE = null;

commC = null;

**Información de la base de datos mediante reader**

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom,conConexCe);

//ojo el reader no tiene constructor SqlCeDataReader reaR;

conConexCe.Open();

reaRCE = commC.ExecuteReader();

comentado para que funcione el segundo bucle

while (reaRCE.Read())

{

MessageBox.Show("la tabla municipios tiene " + reaRCE.FieldCount.ToString () + " campos" );

MessageBox.Show(reaRCE["municipio"].ToString());

}

reaRCE.Read();

for (int i=0; i<=reaRCE.FieldCount -1;i++ )

{

MessageBox.Show ( "el campo " + i + " se llama " + reaRCE.GetName(i) + " y es de tipo " + reaRCE[i].GetType());

}

### Manejo de datos con comandos

1. **Comando con un select**

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select count(municipio) from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom, conConexCe);

conConexCe.Open();

int intCta = 0;

intCta =Convert.ToInt32 (commC.ExecuteScalar());

MessageBox.Show(intCta .ToString ());

conConexCe.Close();

//reaR = null;

commC = null;

1. **Comando con un select con parametro**

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

//de la manera segura mediante parametros a traves del comando

string strCom = "select count(localidad) from localidades inner join municipios " +

"on municipios.idmunicipio=localidades.idmunicipio where municipios.municipio=@municipio";

SqlCeCommand commC = new SqlCeCommand(strCom, conConexCe);

//declara el parametro

commC.Parameters.Add("@municipio", SqlDbType.NVarChar );

//toma el valor del parametro

commC.Parameters["@municipio"].Value = texCriterio.Text;

conConexCe.Open();

int intCta = 0;

intCta = Convert.ToInt32(commC.ExecuteScalar());

MessageBox.Show("el municipio solicitado tiene " + intCta.ToString() + " localidades");

conConexCe.Close();

commC = null;

1. **Comando con un insert con parametro**

if (texCriterio.Text=="")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

//manera segura con parametros

string strCom = "insert into municipios(municipio) values(@municipio)";

SqlCeCommand commC = new SqlCeCommand(strCom, conConexCe);

//declara el parametro

commC.Parameters.Add("@municipio", SqlDbType.NVarChar);

//toma el valor del parametro

commC.Parameters["@municipio"].Value = texCriterio.Text;

conConexCe.Open();

commC.ExecuteNonQuery();

MessageBox.Show("registro agregado");

conConexCe.Close();

//reaR = null;

commC = null;

}

1. **Comando con un delete con parametro**

if (texCriterio.Text == "")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "delete from municipios where municipio=@municipio";

SqlCeCommand commC = new SqlCeCommand(strCom, conConexCe);

//declara el parametro

commC.Parameters.Add("@municipio", SqlDbType.NVarChar);

//toma el valor del parametro

commC.Parameters["@municipio"].Value = texCriterio.Text;

conConexCe.Open();

commC.ExecuteNonQuery();

MessageBox.Show("registro eliminado");

conConexCe.Close();

//reaR = null;

commC = null;

}

1. **Comando con un update con parametro**

if (texCriterio.Text == "")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

//directamente la forma segura, la insegura como las de arriba

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "update municipios set municipio='mod con param' where municipio=@municipio";

SqlCeCommand commC = new SqlCeCommand(strCom, conConexCe);

//declara el parametro

commC.Parameters.Add("@municipio", SqlDbType.NVarChar);

//toma el valor del parametro

commC.Parameters["@municipio"].Value = texCriterio.Text;

conConexCe.Open();

commC.ExecuteNonQuery();

MessageBox.Show("registro modificado");

conConexCe.Close();

//reaR = null;

commC = null;

}

### Manejo de datos con procedimientos almacenados

Esta metodología no es compatible con el modelo Compact Edition, solo están disponibles en modo servidor

utilizar procedimientos almacenados y triggers tiene ventajas sobre las demas tecnicas de acceso a datos porque tienen un rendimiento superior al implementarse en el propio servidor, en un ambiente de equipos de desarrollo tambien es mejor porque bases de datos implementa sus reglas de negocio y programacion no tiene que crear codigo para emular esas reglas. ofrecen seguridad porque los parametros se configuran en tipos, longitud y valores por efault.

Ejemplo de pa con un parametro y valor por default:

use bd

go

create procedure paMunLoxXMunicipio

@mpio char(50)=''

as

select municipios.municipio, localidades.localidad

from municipios inner join localidades on municipios.idmunicipio

=localidades.idmunicipio where municipios.municipio=@mpio

procedimiento almacenado con transaccion

create procedure miproc

@pid integer, @codigo integer, @pnif char(9)

as

begin

begin transaction;

insert into prestamos values(@pid, @pnif, @codigo);

update libros set disponible='N' where codigo=@codigo;

commit;

end;

1. **Procedimiento almacenado con select (inner)**

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "execute paLocXMunOrdenados1";

SqlCommand commC = new SqlCommand(strCom ,conConex );

conConex.Open();

reaR = commC.ExecuteReader();

MiTabla.Load(reaR, LoadOption.OverwriteChanges);

dataGridView1.DataSource = MiTabla;

conConex.Close();

reaR = null;

commC = null;

1. **Procedimiento almacenado con select (inner y parámetro)**

//deberia implementarse con parametros en el comando, para hacerlo mas seguro

if (texCrit2.Text == "")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "execute paMunLoxXMunicipio '" + texCrit2.Text + "'";

SqlCommand commC = new SqlCommand(strCom, conConex);

conConex.Open();

reaR = commC.ExecuteReader();

MiTabla.Load(reaR, LoadOption.OverwriteChanges);

dataGridView1.DataSource = MiTabla;

conConex.Close();

reaR = null;

commC = null

}

1. **Procedimiento almacenado con un escalar**

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "execute paCuenMunicipios";

SqlCommand commC = new SqlCommand(strCom, conConex);

conConex.Open();

int intCta = 0;

intCta =Convert .ToInt32 ( commC.ExecuteScalar());

MessageBox.Show("los municipios son: "+ intCta .ToString ());

conConex.Close();

reaR = null;

commC = null;

}

1. **Procedimiento almacenado con insert, update y delele**

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "execute paNueMun '" + texCrit2 .Text +"'";

SqlCommand commC = new SqlCommand(strCom, conConex);

conConex.Open();

commC.ExecuteNonQuery();

MessageBox.Show("registro efectuado");

conConex.Close();

reaR = null;

commC = null;

### Manejo de datos con procedimientos triggers

EN REALIDAD CON LOS TRIGGERS NO SE REQUIERE TRABAJO A NIVEL PROGRAMACION, YA QUE SON TAREAS PLANIFICADAS A NIVEL SERVIDOR QUE POR LO REGULAR LAS DESENCADENA UN PA

1. **Trigger que crea un nuevo campo**

POR EJEMPLO ESTA ACCION, PODRIAMOS PLANTEAR QUE ANTE LA CREACION DE UN NUEVO MUNICIPIO ES NECESARIO CREAR UNA LOCALIDAD CABECERA CON EL MISMO NOMBRE

EJEMPLO DE TRIGGER QUE AFECTA OTRA TABLA AL INSERT

use bd

go

create trigger trigNueMunicipio

on municipios

for insert

as

declare @idmunicipio numeric

declare @cabecera varchar(60)

select @idmunicipio=(select idmunicipio from inserted)

select @cabecera=(select municipio from inserted)

insert into localidades(idmunicipio,localidad) values(@idmunicipio,@cabecera)

SE SUPONE QUE DEBE MEDIAR UN PA PARA INSERTAR UN NUEVO REGISTRO EN LA TABLA MUNICIPIOS LO CUAL DESENCADENARA EL TRIGGER. COMO EJEMPLO LO HACEMOS DIRECTAMENTE

if (texCrit2.Text == "")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "insert into municipios (municipio) values ('"+texCrit2 .Text +"')";

SqlCommand commC = new SqlCommand(strCom, conConex);

conConex.Open();

commC.ExecuteNonQuery ();

MessageBox.Show("registro insertado");

conConex.Close();

reaR = null;

commC = null;

}

1. **Trigger que elimina un campo**

UN EJEMPLO TIPICO ES CUANDO SE ELIMINA UN MUNICIPIOS, ES NECESARIO ELIMINAR SUS LOCALIDADES PARA QUE NO QUEDEN REGISTROS HUERFANOS

use bd

go

create trigger trigElimMunicipio

on municipios

for delete

as

declare @idmunicipio numeric

select @idmunicipio=(select idmunicipio from deleted)

delete from localidades where idmunicipio=@idmunicipio

if (texCrit2.Text == "")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "delete from municipios where idmunicipio=" + Convert.ToInt32(texCrit2.Text);

SqlCommand commC = new SqlCommand(strCom, conConex);

conConex.Open();

commC.ExecuteNonQuery();

MessageBox.Show("registro eliminado");

conConex.Close();

reaR = null;

commC = null;

}

1. **Trigger que modifica un campo**

EL TRIGGER SIGUIENTE ACTUALIZA LA COLUMNA IMPORTE DE LOCALIDADES CUANDO SE MODIFIQUE LA COLUMNA MUNICIPIO DE MUNICIPIOS PARA EL ID MODIFICADO (\*\*\*VER QUE AUNQUE SE TRATE DE UN UPDATE SE UTILIZA FROM INSERTED\*\*\*\*)

create trigger triUp

on municipios

after

update

as

declare @idmunicipio numeric

select @idmunicipio=(select idmunicipio from inserted)

if update(municipio)

update localidades set importe=1.45 where idmunicipio=@idmunicipio

if (texCrit2.Text == "")

{

MessageBox.Show("debe proprocionar un valor");

}

else

{

string strConex = @"Data Source=aperezg1\sqlexpress;Initial Catalog=bd;Integrated Security=true";

conConex = new SqlConnection();

conConex.ConnectionString = strConex;

string strCom = "update municipios set municipio='\_up' where idmunicipio=" + Convert.ToInt32(texCrit2.Text);

SqlCommand commC = new SqlCommand(strCom, conConex);

conConex.Open();

commC.ExecuteNonQuery();

MessageBox.Show("registro modificado");

conConex.Close();

reaR = null;

commC = null;

}

# Datos del origen a controles de usuario

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

### A datagridview

griPadre.DataSource = datM.Tables["municipios"];

LIMPIA UN GRID

this.griTalones .DataSource =new DataTable();

O TAMBIEN LIMPIANDO LA TABLA FUENTE

table.Clear();

QUITA EL REGISTRO SELECCIONADO DEL GRID

dataGridView1.Rows.Remove(dataGridView1.CurrentRow);

PASA EL VALOR DE LA COLUMNA X DE LA FILA ACTUAL

(this.dataGridView1[1, this.dataGridView1.CurrentCell.RowIndex].Value);

CUANDO LAS COLUMNAS SE DESORDENAN EN EL GRID SE PUEDEN ORDENAR INDICANDO SU DISPLAYINDEX

dataGridView1.Columns["descripcion"].DisplayIndex = 0;

PARA OCULTAR CIERTAS COLUMNAS

dataGridView1.Columns["descripcion"].visible=False;

### A gridview (ASP.NET)

IMPLEMENTAR LAS OPERACIONES BASICAS DE BASE DE DATOS DESDE UN GRIDVIEW. PRIMERO QUE NADA HAY QUE MENCIONAR QUE EXISTEN VARIOS ENFOQUES PARA HACER ESTO:

**FORMA MANUAL**: ESTE ENFOQUE ES CAPTURANDO LOS EVENTOS DEL GRIDVIEW Y PASANDOLE LOS VALORES A LA BASE DESDE EL GRID POR MEDIO DE CONTROLES DE USUARIO. Esto implica que debe haber una clase que represente al grid con los procedimientos necesarios para su operatividad.

1.-PRIMERO HABILITAMOS EL AUTOGENERATESELECTBUTTON CON ESTO PERMITOMOS DESENCADENAR EL EVENTO SOBRE EL GRID, LUEGO VAMOS A LOS EVENTOS Y GENERAMOS EL SelectedIndexChanged QUE ES DE DONDE TOMAREMOS VALORES.

2.-al desencadenar desde el evento podemos pasar los valores actuales del gridview

claCliente.Direccion = GridView1.Rows[GridView1.SelectedIndex].Cells[1].Text;

**CON PLANTILLAS**: AQUÍ SE TRATA DE GENERAR UNA PLANTILLA PARA QUE LAS OPERACIONES SEAN “IN SITU” ESTA FORMA TAMBIEN IMPLICA LA CODIFICACION MANUAL.

**CON PROCEDIMIENTOS ALMACENADOS**: APARTE DE QUE ES UNA FORMA FACIL POR MEDIO DEL DISEÑADOR, ES LA FORMA OPTIMA DESDE EL PUNTO DE VISTA DEL PERFORMANCE.

**FORMATO:** SI EL GV AUTO GENERA LAS COLUMNAS HAY QUE DESHABILITARLO. LUEGO SE DEBEN GENERAR LAS COLUMNAS QUE SE DESEAN MOSTRAR EN LA SIGUIENTE JERARQUIA:

<asp:GridView ID="griFolios" runat="server" AutoGenerateSelectButton="True" CellPadding="4" ForeColor="#333333" GridLines="None" Width="100%" AutoGenerateColumns="False">

<Columns>

<asp:BoundField DataField="folio" HeaderText="Folio" ItemStyle-HorizontalAlign="center" HeaderStyle-Font-Size="small"

HeaderStyle-HorizontalAlign="Center"/>

<asp:BoundField DataField="no\_contrato" HeaderText="Contrato" ItemStyle-Font-Size="Small" ItemStyle-HorizontalAlign="Center"

HeaderStyle-Font-Size="small" HeaderStyle-HorizontalAlign="Center"/>

<asp:BoundField DataField="alcance\_liquido" HeaderText="A. líquido" DataFormatString="{0:C}" ItemStyle-Font-Size="Small"

ItemStyle-HorizontalAlign="Right" HeaderStyle-Font-Size="small" HeaderStyle-HorizontalAlign="Center"/>

Una tarea común es seleccionar una fila del gridview y pasar su valor a algún otro objeto, esto normalmente se hace desde el selectionchange(), pero hay otra forma que aparentemente detecta el tipo de acción que se ha generado desde el gridview, el el ejemplo siguiente el caso de un select a la fila:

protected void GridView1\_RowCommand(object sender, GridViewCommandEventArgs e)  
{  
     if(e.CommandName == “Select”)  
     {  
        //Seleccionamos la Fila Actual  
        GridView1.SelectedIndex = Convert.ToInt32(e.CommandArgument.ToString());

         //Recuperamos el valor de la segunda celda  
        TextBox1.Text = GridView1.SelectedRow.Cells[1].Text;  
     }  
}

UNA VEZ HECHO ESTO YA SE LE PUEDEN DAN FORMATO A LAS COLUMNAS. RECORDEMOS QUE SI EL GV ESTA LIGADO A UN DATASOURCE, ESTA JERARQUIA DE COLUMNAS SE DECLARA SOLA, ASI QUE SERA MAS FACIL MANIPULARLA.

### A Chart

EL EJEMPLO MUESTRA LA VINCULACION A UNA MATRIZ, PERO PODRIA FACILMENTE CAMBIARSE A UNA TABLA DE UNA BASE DE DATOS

private void Form1\_Load(object sender, EventArgs e)

{

double [] douS1={55,45,36,98};

double [] douS2={84,54,63,84};

this.chart1.Series.Add("Series2");

this.chart1.Series.Add("Series1");

for (int i = 0; i <= douS1.Length - 1; i++)

{

this.chart1.Series["Series1"].Points.Add(douS1[i]);

}

for (int ii = 0; ii <= douS1.Length - 1; ii++)

{

this.chart1.Series["Series2"].Points.Add(douS2[ii]);

}

//this.chart1.Series["Series2"].Points.Add(10, 10);

…

//this.chart1.Series["Series1"].Points.Add(1,10);

…

}

### A combobox

**DESDE UN DATASET**

this.comboBox1.ValueMember = Convert.ToString(datM.Tables["municipios"].Columns["idmunicipio"]);

this.comboBox1.DisplayMember = Convert.ToString(datM.Tables["municipios"].Columns["municipio"]);

this.comboBox1.DataSource = datM.Tables["municipios"];

**DESDE UN READER**

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom,conConexCe);

//ojo el reader no tiene constructor SqlCeDataReader reaR;

conConexCe.Open();

reaRCE = commC.ExecuteReader();

MiTabla.Load(reaRCE, LoadOption.OverwriteChanges);

this.comboBox1.ValueMember = Convert.ToString(MiTabla.Columns["idmunicipio"]);

this.comboBox1.DisplayMember = Convert.ToString(MiTabla.Columns["municipio"]);

This.comboBox1.DataSource = MiTabla;

**HACER REFERENCIA A LOS ELEMENTOS DE UN COMBO**

retorna el valor que se ha cargado en la propiedad value, genralmente es un indice

this.comboBox1.SelectedValue.ToString ();

desde una tabla

this.comStatAlumno.SelectedValue= tabStatus.Rows[0]["clave\_status"].ToString();

**WEB, DESCARGAR DATOS A DROPDOWNLIST**

comCliente.DataTextField =Convert.ToString(tabCliente.Columns["nombre"]);

comCliente.DataValueField = Convert.ToString(tabCliente.Columns["id"]);

comCliente.DataSource = tabCliente;

comCliente.DataBind();

ojo: para limpiar los datos de un dropDown hay que ver la forma en que estan llenados. Si se llena desde su propiedad collection, se puede vaciar com drpXXX.Text=””;. Si se llena desde una tabla o datatable drpXXX.Items.Clear(); (si es que hay un elemento vacio en la lista correspondiente).

### A Listbox

DESDE UN DATASET

foreach (DataRow dR in datM.Tables["municipios"].Rows)

{

this.listBox1.Items.Add(dR[0].ToString() + " " + dR[1].ToString());

}

DESDE UN READER

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom,conConexCe);

OJO EL READER NO TIENE CONSTRUCTOR SQLCEDATAREADER REAR;

conConexCe.Open();

reaRCE = commC.ExecuteReader();

while (reaRCE.Read())

{

this.listBox1.Items.Add (reaRCE["municipio"].ToString());

}

### A textbox

PARA ACCESAR AL VALOR DE REGISTRO DE UNA COLUMNA HAY QUE DETERMINAR LA FILA, EN ESTE CASO LA FILA 0 SE SUPONE QUE LA TABLA TIENE UNA O MAS FILAS RETORNADAS

DESDE UN DATASET

this.textBox4.Text = datM.Tables["municipios"].Rows[0]["municipio"].ToString();

FILAS DE LA TABLA ALQUILERES DEL DATASET

foreach (DataRow Fila in dataSet1.Tables[0].Rows)

textBox1.Text += Fila["id"].ToString() + " " + Fila["municipio"].ToString() + Environment.NewLine;

DESDE UN READER

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom,conConexCe);

//ojo el reader no tiene constructor SqlCeDataReader reaR;

conConexCe.Open();

reaRCE = commC.ExecuteReader();

ACCESO AL VALOR DEL CAMPO

reaR.Read ();

this.textBox1.Text =reaR["municipio"].ToString ();

PARA PONER LOS VALORES DEL READ EN UN TEXTBOX A MODO DE LISTA

while (reaReader.Read())

{

textBox1.Text += reaReader["municipio"].ToString() + Environment.NewLine;

}

VINCULACION CON BINDING

ES UN METODO PARA EL LLENADO DE CONTROLES MANTENIENDO LA VINCULACION CON LA BASE DE DATOS, SU VIRTUD ES MANTENER LA VISUALIZACION SIN NECESIDAD DE MUCHO CODIGO

textBox1.DataBindings.Add("Text", datDataset, "municipios.municipio");

### A tabla

DESDE UN READER

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom,conConexCe);

OJO EL READER NO TIENE CONSTRUCTOR SQLCEDATAREADER REAR;

conConexCe.Open();

reaRCE = commC.ExecuteReader();

MiTabla.Load(reaRCE, LoadOption.OverwriteChanges);

dataGridView1.DataSource = MiTabla;

conConexCe.Close();

reaRCE = null;

commC = null;

ITERAR SOBRE LOS REGISTROS DE UNA TABLA

foreach(DataRow rowF in table.Rows)

{

douSumTalones +=Convert.ToDecimal(rowF["Importe"]);

this.labSuma .Text =string.Format ("{0:0,0.00}",douSumTalones);

}

ACCESO AL VALOR DE UNA COLUMNA EN UNA FILA 0 O UNICA. ESTO ES PRODUCTO DE UNA CONSULTA QUE REGRESA FILA UNICA PRODUCTO DE SUMAS EN LOS CAMPOS.

VARIABLE=tabExis.Rows[0]["existencia"]

### Control de navegacion

string strConex = @"Data Source=C:\Users\adan\Desktop\proyectos\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

this.textBox1.DataBindings.Add("text", datM, "municipios.idmunicipio");

this.textBox2.DataBindings.Add("text", datM, "municipios.municipio");

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SE VALE DE LOS SIGUIENTES EVENTOS RELACIONADOS:

1. **Atrás**

CurrencyManager cm = (CurrencyManager)this.BindingContext[datM, "municipios"];

if (cm.Position > 0)

{

cm.Position--;

}

1. **Adelante**

CurrencyManager cm = (CurrencyManager)this.BindingContext[datM, "municipios"];

if (cm.Position < cm.Count - 1)

{

cm.Position++;

}

1. **Primero**

CurrencyManager cm = (CurrencyManager)this.BindingContext[datM, "municipios"];

cm.Position = 0;

1. **Ultimo**

CurrencyManager cm = (CurrencyManager)this.BindingContext[datM, "municipios"];

cm.Position = cm.Count -1 ;

### Control de navegacion mediante una tabla

//una opcion es volcar una reader a una tabla y hacer la navegacion desde la misma recordemos que un reader es //solo de recorrido hacia delante

string strConex = Settings.Default.myConACE;

conConexCe = new SqlCeConnection(strConex);

string strCom = "select \* from municipios";

SqlCeCommand commC = new SqlCeCommand(strCom, conConexCe);

conConexCe.Open();

reaRCE = commC.ExecuteReader();

MiTabla.Load(reaRCE, LoadOption.OverwriteChanges);

conConexCe.Close();

reaRCE = null;

commC = null;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

el complemento desde butAdelante\_Click

if (i < MiTabla.Rows.Count)

{

textBox1.Text = MiTabla.Rows[i]["municipio"].ToString();

i++;

}

esta es la foma de navegar con el reader, solo que unicamente avanza hacia adelante solo hay que quitar el cierre de la conexion, el null al reader y el volcado a la tabla cada vez que se invoca a read avanza uno

reaR.Read();

textBox1.Text = reaR ["municipio"].ToString();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

el complemento desde butAtras\_Click

if (i > 0)

{

i--;

textBox1.Text = MiTabla.Rows[i]["municipio"].ToString();

}

### DateTimePicker

dateTimePicker1.Format = DateTimePickerFormat.Custom;

dateTimePicker1.CustomFormat = "yy/MM/dd";

//quiero que muestre la fecha en ese formato yy/mm/dd, pero la asignación de l valor debe ser de una fecha con formato “regular”

string cadena = "1972/05/09";

DateTime fecha = DateTime.Parse(cadena

dateTimePicker1.Value = fecha;

**CONTROL CALENDAR ASP.NET**

SI SE PRETENDE LIMPIAR EL CONTROL SE PONE:

calFecStatus.SelectedDate =DateTime.Parse( "01/01/0001 12:00:00 a. m.");

### Tabla M-D manual

LA IDEA DE ESTO ES UTILIZAR LOS GRID COMO INTERFACE PARA LA MANIPULACION DE DATOS **(CAPTURA DE DATOS EN LAS DOS TABLAS GUARDANDO RESTRICCIONES DE MAESTRO DETALLE Y CLAVES UNICAS)**, PERO POR EXPERIENCIA SE DEDUCE QUE ESTA NO ES LA FORMA OPTIMA PARA EL USUARIO, YA QUE SE PRESTA AL DEDAZO, BORRADO ACCIDENTAL, ETC, ES MAS ORDENADO TILIZAR CAJAS COMO INTERFACE DEL GRID, PARA LA MANIPULACION DE REGISTROS, Y POR DETRAS DE LAS CAJAS, PA, TRIGGERS O COMANDOS ADEMAS, LA CONFIGURACION DE LAS RESTRICCIONES, CLAVES, RELACIONES, ETC DE LAS TABLAS PARA EL GRID Y LA SINCRONIZACION DE DATOS RESULTA MUY ENGORROSA PARA ESE PROPOSITO, QUE UTILIZANDO LOS OTROS METODOS ES MUCHO MAS FACIL Y EFICIENTE.

//crear las columnas

DataColumn colIdM = new DataColumn ();

DataColumn colIM = new DataColumn ();

//DataColumn colIExp = new DataColumn(); //solo si se quiere crear un campo con expresion o calculado

//las configura \*\*recordar que hay muchas opciones de configuracion: autoincrement, longitud, etc.

colIdM .DataType =System.Type .GetType("System.Int32");

colIdM.ColumnName = "idmunicipio";

colIM.DataType = System.Type.GetType("System.String");

colIM.ColumnName = "municipio";

//colIExp.DataType = System.Type.GetType("System.Double");

//colIExp.ColumnName = "expresion";

//colIExp.Expression ="0.25 \* 200";

MiTabla.Columns.Add(colIdM );

MiTabla.Columns.Add(colIM);

//MiTabla.Columns.Add(colIExp);

//define la llve primaria

DataColumn[] colLlave = new DataColumn[1];

colLlave[0] = colIdM;

MiTabla.PrimaryKey = colLlave;

//-----------------------------------------------------------

//crea la otra tabla

DataColumn colIdL = new DataColumn();

DataColumn colIdM1 = new DataColumn();

DataColumn colL = new DataColumn();

//las configura \*\*recordar que hay muchas opciones de configuracion: autoincrement, longitud, etc.

colIdL.DataType = System.Type.GetType("System.Int32");

colIdL.ColumnName = "idlocalidad";

colIdM1.DataType = System.Type.GetType("System.Int32");

colIdM1.ColumnName = "idmunicipio";

colL.DataType = System.Type.GetType("System.String");

colL.ColumnName = "localidad";

MiTabla2.Columns.Add(colIdL);

MiTabla2.Columns.Add(colIdM1);

MiTabla2.Columns.Add(colL);

//define la llave primaria

DataColumn[] colLlaveL = new DataColumn[1];

colLlaveL[0] = colIdL;

MiTabla2.PrimaryKey = colLlaveL;

//carga tabla al datet

datM.Tables.Add(MiTabla);

//carga tabla al datet

datM.Tables.Add(MiTabla2);

//la relacion

DataRelation relML = new DataRelation("munLoc", colIdM, colIdM1, true );

datM.Relations.Add(relML);

//carga la tabla manual al grid

this.griHijo.DataSource = MiTabla2;

//carga la tabla manual al grid

griPadre.DataSource = MiTabla;

### Tabla M-D manual (agregar registro)

DataRow filM;

filM = MiTabla.NewRow();

filM["idmunicipio"]=1;

filM["municipio"] = "mpio";

MiTabla.Rows.Add(filM);

### Tabla M-D manual (datagrid jerarquico agregar registro)

Esta tecnica es aceptable porque ofrece buena funcionalidad sin tanta confusion para el manejo temporal de datos en DataGrid, solo habria que ver como sincronizar con el origen. Crea la tabla maestro con una fila y cuando se crea el detalle ya la manda con su respectiva clave de maestro, creando restricciones de integridad

//VAriables de objeto para la tabla CLientes

DataColumn colIdMunicipio=new DataColumn();

DataColumn colMunicipio=new DataColumn();

DataTable tabMpios=new DataTable ("municipios");

DataColumn[] LlaveMpios=new DataColumn [1];

DataColumn colIdlocalidad=new DataColumn();

DataColumn colIdMunicipio1=new DataColumn();

DataColumn colLocalidad=new DataColumn();

DataTable tabLocs=new DataTable ();

DataColumn[] LlaveLocs=new DataColumn [1];

//dataset que contendra las tablas

DataSet datML =new DataSet ();

//configuracion de los campos

colIdMunicipio.ColumnName ="idmunicipio";

colIdMunicipio.DataType =System.Type.GetType("System.Int32");

colIdMunicipio.AllowDBNull =false ;

colIdMunicipio.AutoIncrement =true ;

tabMpios .Columns .Add (colIdMunicipio );

colMunicipio.ColumnName ="municipio";

colMunicipio.DataType =System.Type.GetType("System.String");

colMunicipio.AllowDBNull =false ;

tabMpios .Columns .Add (colMunicipio );

colIdlocalidad.ColumnName ="idlocalidad";

colIdlocalidad.DataType =System.Type.GetType("System.Int32");

colIdlocalidad.AllowDBNull =false ;

colIdlocalidad.AutoIncrement =true ;

tabLocs .Columns .Add (colIdlocalidad );

colIdMunicipio1.ColumnName ="idmunicipio";

colIdMunicipio1.DataType =System.Type.GetType("System.Int32");

colIdMunicipio1.AllowDBNull =false ;

tabLocs.Columns.Add(colIdMunicipio1 );

colLocalidad.ColumnName ="localidad";

colLocalidad.DataType =System.Type.GetType("System.String");

colLocalidad.AllowDBNull =false ;

tabLocs .Columns .Add (colLocalidad );

//Establece la clave primaria de la tablas.

LlaveMpios[0]=colIdMunicipio;

tabMpios.PrimaryKey=LlaveMpios;

LlaveLocs[0]=colIdlocalidad;

tabLocs.PrimaryKey=LlaveLocs;

//la nueva fila \*\*\*no se pondra

DataRow rowFil;

//Ahora va a agregar una fila a la tablas.

rowFil = tabMpios.NewRow();

rowFil[colIdMunicipio ] = 1;

rowFil[colMunicipio ] = "municipio1";

tabMpios.Rows.Add(rowFil);

rowFil = tabMpios.NewRow();

//agrega las tablas al dataset

datML.Tables .Add (tabMpios );

datML.Tables .Add (tabLocs );

//Define la relación que habrá enrtre las tablas.

DataRelation relRel=new DataRelation("munLoc", colIdMunicipio, colIdMunicipio1 ,true );

//Agrega la relación al Dataset.

datML.Relations .Add (relRel );

//Exhibe en controles separados.

this.dataGrid1 .DataSource =datML;

this.dataGrid1 .DataMember = "municipios";

//Clientes.DataSource = DS

//Clientes.DataMember = "Clientes"

**PictureBox**

Image image = Image.FromFile(@"c:\bachiller\fotos\alumnos\" + claAlumno.CURP + ".png");

this.pictureBox1.Image = image;

### Guarda dataset en archivo xml

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

datM .WriteXml(@"c:\MiTabla.xsd",XmlWriteMode.WriteSchema );

### Grid maestro detalle

LA IDEA ES LLEGAR EL GRID MAESTRO Y AL DAR DOBLE CLICK EN ALGUNO DE SUS REGISTROS QUE SE MUESTRE EL DETALLE EN EL OTRO GRID

string strConex = @"Data Source=C:\Users\adan\Desktop\proyectos\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

//grid

griPadre.DataSource = datM.Tables["municipios"];

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

se vale del evento click o doble click del grid padre para su funcionalidad

int intCve=0;

intCve=Convert .ToInt32 ( this.griPadre [0,this.griPadre.CurrentCell .RowIndex ].Value);

dependiendo, en este caso seria bueno rellenarlo con reader al menos un grid busca los valores del hijo lo ideal seria utilizar un PA para rellenar el hijo y utilizar parametros, pero por facilidad lo haremos con un objeto command

DataSet datMLoc = new DataSet();

string strConex = @"Data Source=C:\Users\adan\Desktop\proyectos\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from localidades where idmunicipio=" +intCve ;

datMLoc.Clear();

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datMLoc, "localidades");

//grid

griHijo.DataSource = datMLoc.Tables["localidades"];

### Búsqueda en el dataset

1. **Por numero de fila( index)**

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

DataRow fil;

//se le debe pasar un numero como argumento, que representa el numero de fila

fil = datM.Tables["municipios"].Rows[3];

this.textBox1.Text =fil["idmunicipio"].ToString() ;

this.textBox2.Text = fil["municipio"].ToString();

**(2) metodo find**

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

LAS BUSQUEDAS SE DEBEN HACER POR MEDIO DE UNA CLAVE PRIMARIA, ESTE ES EL CASO SENCILLO CUANDO LAS CLAVES SON DE MAS DE UN CAMPO SE DEBE DECLARAR UN AMATRIZ

DataColumn[] matLlave = new DataColumn[1];

matLlave[0] = datM.Tables["municipios"].Columns["idmunicipio"];

datM.Tables["municipios"].PrimaryKey = matLlave;

//'busca una fila especifica

DataRow drwFila;

drwFila = datM.Tables["municipios"].Rows.Find(8);

textBox1 .Text =Convert .ToString ( drwFila["idmunicipio"]);

textBox2.Text = Convert.ToString(drwFila["municipio"]);

**(3) MEDIANTE VISTAS**

string strConex = @"Data Source=C:\Documents and Settings\Administrador\Escritorio\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

//'declara una vista para filtar datos

DataView visMpios = new DataView(); ;

visMpios.Table = datM.Tables["municipios"];

//'metodos de la vista similar a una cadena sql, muy flexible pp 475 migracion y ref.

//visMpios.Sort = "idmunicipio desc";

visMpios.RowFilter = "municipio='acajete'";

//'puede mostrar una version del dataset, pg eliminado, sin cambios...

//visMpios.RowStateFilter = DataViewRowState.ModifiedCurrent;

//'quita filtros

//visMpios.RowFilter = "";

griPadre.DataSource = visMpios;

acceso a los datos de una columna del dataview

foreach(DataRowView row in visProd )

{

if (row["codigo"] == this.comCodigo.Text)

{

this.texDescripcion.Text = row["descripcion"].ToString();

}

}

### Datos en la copia

UTILIZA LA SIGUIENTE CADENA DE CONEXIÓN Y POBLADO DEL DATASET

string strConex = @"Data Source=C:\Users\adan\Desktop\proyectos\ADO\_PUNTO\_NET\ADO\_PUNTO\_NET\bd.sdf; Persist Security Info=False";

string strCom = "select \* from municipios";

adaM = new SqlCeDataAdapter(strCom, strConex);

adaM.Fill(datM, "municipios");

//grid

griPadre.DataSource = datM.Tables["municipios"];

UNA VEZ QUE SE LLENA UN GRID, SE REALIZAN LOAS OPERACIONES BASICAS

1. **Delete**

DataRow filFila;

filFila = datM.Tables["municipios"].Rows[5];

filFila.Delete();

//COMPRUEBA LA ELIMINACION, aqui podria tomarse el index de la fila

//actual

griPadre.DataSource = datM.Tables["municipios"];

1. **Insert**

//probar con tecnicas para agregar de forma dinamica

DataRow Fila;

Fila=datM .Tables["municipios"].NewRow ();

Fila["idmunicipio"] = 100;

Fila["municipio"] = "prueba";

datM.Tables["municipios"].Rows.Add(Fila);

MessageBox.Show("fila agregada");

1. **Update**

DataRow Fil;

Fil =datM .Tables ["municipios"].Rows [5];

Fil.BeginEdit();

Fil["municipio"] = "mun\_modificado";

Fil.EndEdit();

MessageBox.Show("fila modificada");

1. **Insert con load data**

NO ES UN METODO MUY EFICIENTE LO IDEAL ES HACER UNA COLECCION PERSONALIZADA PARA LA TABLA MUNICIPIOS //COMO ESTA CARGANDO CON EL METODO LOADDATAROW, ESTA TOMANDO LAS RESTRICCIONES DEL DATASET SI YA EXISTE EL INDICE MODIFICA LA CLAVE PROPORCIONADA, SI NO, AGREGA UNA NUEVA

DataColumn [] Llave=new DataColumn [1];

Llave[0]=datM.Tables["municipios"].Columns["idmunicipio"];

datM.Tables["municipios"].PrimaryKey = Llave;

object[] Mat = { 1, "mpio\_agregardo\_load" };

DataRow Fil;

datM.Tables["municipios"].BeginLoadData() ;

Fil = datM.Tables["municipios"].LoadDataRow(Mat, true);

datM.Tables["municipios"].EndLoadData();

MessageBox.Show("fila agregada");

### Binding source y binding navigator

A BINDING LO PODEMOS CONSIDERAR MIXTO ENTRE MANUAL Y ASISTIDO, YA QUE UNA VEZ CONFIGURADO MANTIENE FACILMENTE LA VISUALIZACION DE LOS DATOS. ADEMAS DA TODA LA IMPLEMENTACION PARA INSERT, UPDATE Y DELETE.

EL EJEMPLO TIENE LA **BARRA DE COMANDOS** ARRIBA Y UNA CAJA DE TEXTO COMO CONTROL PRINCIPAL

private void Form2\_Load(object sender, EventArgs e)

{

SqlConnection MiConexion = new SqlConnection(@"Integrated Security=SSPI;Data Source=APEREZG1;database=r20");

//C:\Users\adan\Desktop\Proyectos\InicioCSharp\proy8-Datos\bd.sdf

string strSql = "SELECT \* FROM municipios";

SqlCommand MiComando = new SqlCommand(strSql, MiConexion);

SqlDataReader MiDataReader;

DataTable MiTabla = new DataTable();

ABRIMOS LA CONEXION

MiConexion.Open();

EJECUTAMOS LA INSTRUCCION SQL

MiDataReader = MiComando.ExecuteReader();

CARGAMOS EN LA TABLA LA LECTURA DEL DATAREADER

MiTabla.Load(MiDataReader, LoadOption.OverwriteChanges);

VOLCAMOS LOS DATOS EN EL CONTROL TEXTOBOX

bindingSource1.DataSource = MiTabla;

bindingNavigator1.BindingSource = bindingSource1;

textBox1.DataBindings.Add("Text", bindingSource1, "municipio", true);

CERRAMOS LA CONEXION

MiConexion.Close();

# Linq to sql

LINQ Y FUNCIONES RELACIONALES es mas flexible porque puede accesar a cualquier estructura jerarquica y sin necesidad de insertar código ajeno a la sintaxis determinada, por ejemplo a la de c#, como es es caso si se mete codigo sql.

Linq crea sus propias colecciones basándose en clases construidas por uno mismo, que mapean las tablas de la base de datos. Estas clases se denominan CLASES DE ENTIDAD. El objetivo de linq es proporcionar una interface para sql server, sin embargo ante la robustez muchos han creado sus proveedores de datos para trabajar con linq.

1. **El primer paso es agregar la referencia a los ensamblados**

using System.Data.Linq;

using System.Data.Linq.Mapping;

using System.Data.SqlClient;

1. **luego se crea la clase entidad que mapea la tabla de la base de datos Orders**

[Table(Name = "Orders")]

public class Order

{

[Column(IsPrimaryKey = true, CanBeNull = false)]

public int OrderId { get; set; }

[Column]

public string CustomerId { get; set; }

[Column]

public DateTime? OrderDate { get; set; }

[Column]

public DateTime? ShippedDate { get; set; }

[Column]

public string ShipName { get; set; }

[Column]

public string ShipAddress { get; set; }

[Column]

public string ShipCity { get; set; }

[Column]

public string ShipCountry { get; set; }

}

1. **Luego se crea una clase que herede de data context, que expone una propiedad table basada en la clase entidad Order**

public class NorthWind : DataContext

{

public Table<Order> Orders;

public NorthWind(string connectionInfo):base(connectionInfo)

{

}

}

1. **Finalmente se implementa la instrucción Linq**

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();

builder.DataSource = "aperezg1";

builder.InitialCatalog = "NorthWind";

builder.IntegratedSecurity =true;

se crea una instancia de la clase NorthWind en la cual se resume la clase entidad y la conexión a la base

NorthWind northWind = new NorthWind(builder.ConnectionString);

try

{

Console.WriteLine("entre numero de cliente");

string customerId = Console.ReadLine();

Se crea la instruccion linq basandose en la clase instanciada

var ordersQuery = from o in northWind.Orders

where string.Equals(o.CustomerId, customerId)

select o;

se muestran los resultados

foreach (var order in ordersQuery)

{

if (order.ShippedDate == null)

{

Console.WriteLine("order {0} not yet shiped \n\n", order.OrderId);

}

else

{

Console.WriteLine("Order: {0}\nPlaced: {1}\nShipped: {2}\n" +

"to Address: {3}\n{4}\n{5}\n{6}\n\n",order.OrderId, order.OrderDate,

order.ShippedDate,order.ShipName,order.ShipAddress, order.ShipCity,

order.ShipCountry);

Console.Read();

}

}

}

catch (SqlException e)

{

Console.WriteLine("Error de acceso: {0}", e.Message);

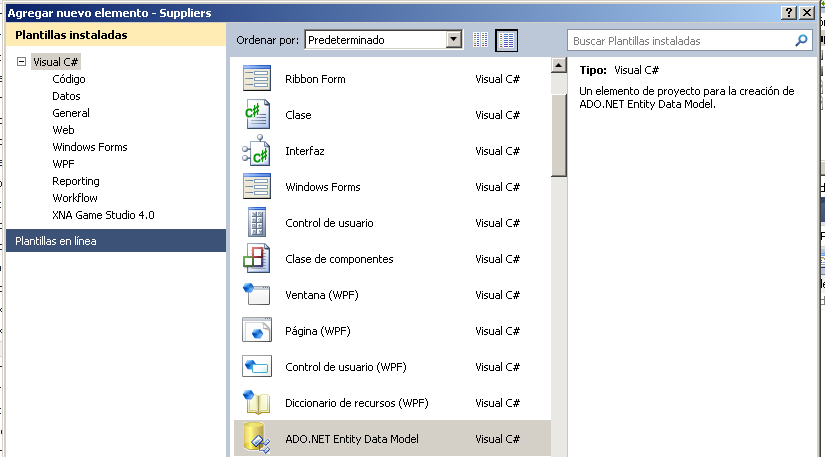
}

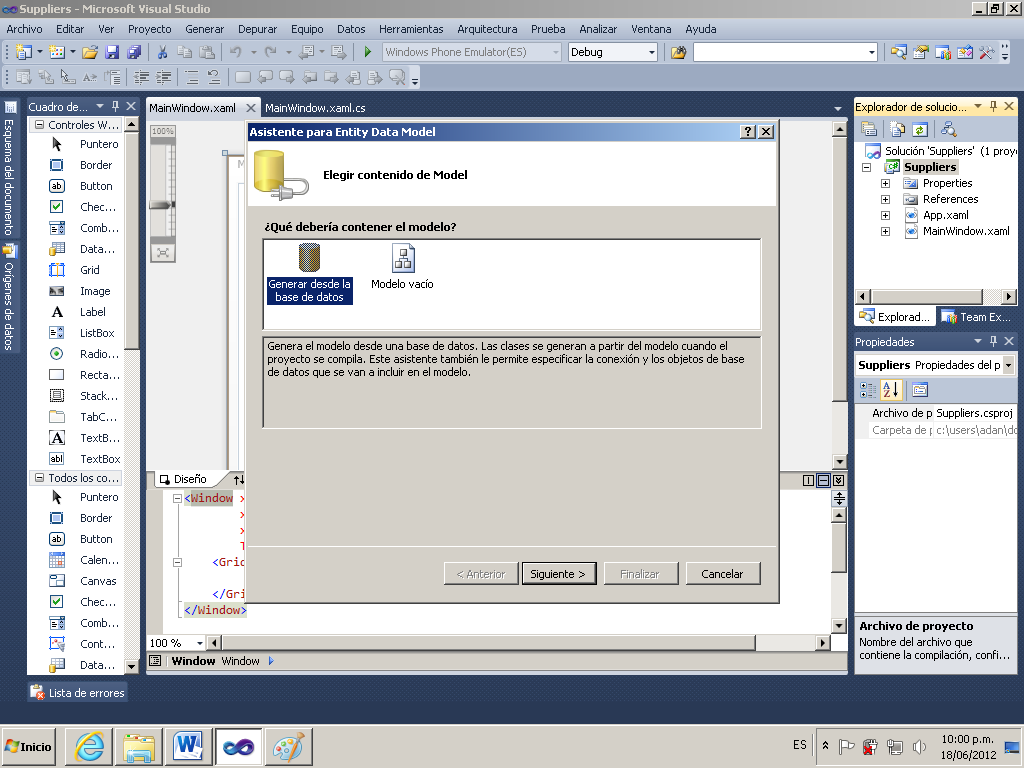
## Entity Framework

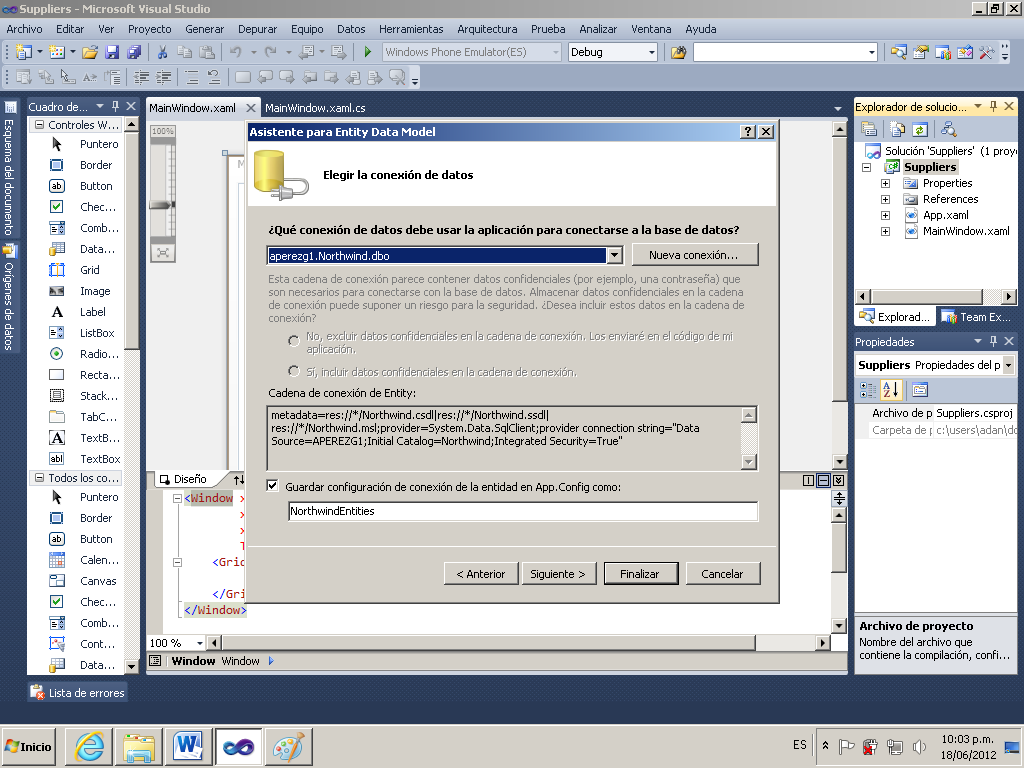
El ejemplo anterior es viable, sin embrago, Microsoft ha desarrollado Entity Framework como un esquema mas integral para trabajo con linq, con lo cual se logra entre otros beneficios: cierta independencia de la tecnología (similar a la factoria) y en general todo un modelo lógico de datos (similar a la validación en wpf en donde un control se liga a una propiedad de una clase). Este modelo especifico se llama Linq to Entity.

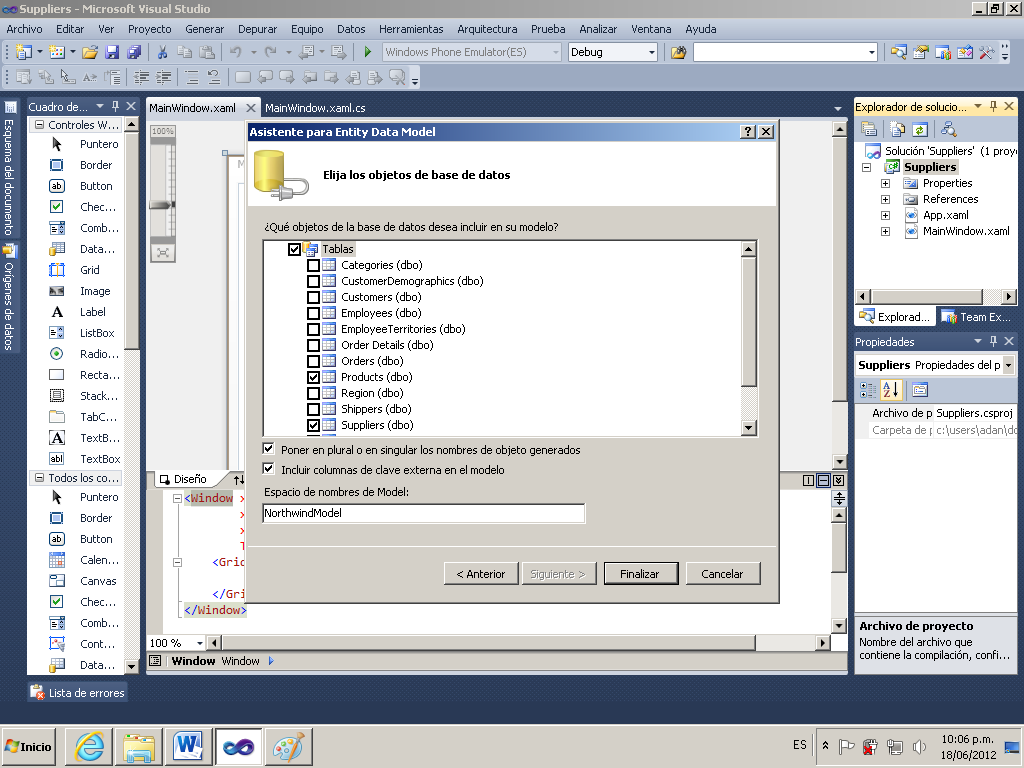
### Configuracion de la aplicacion y acceso a datos

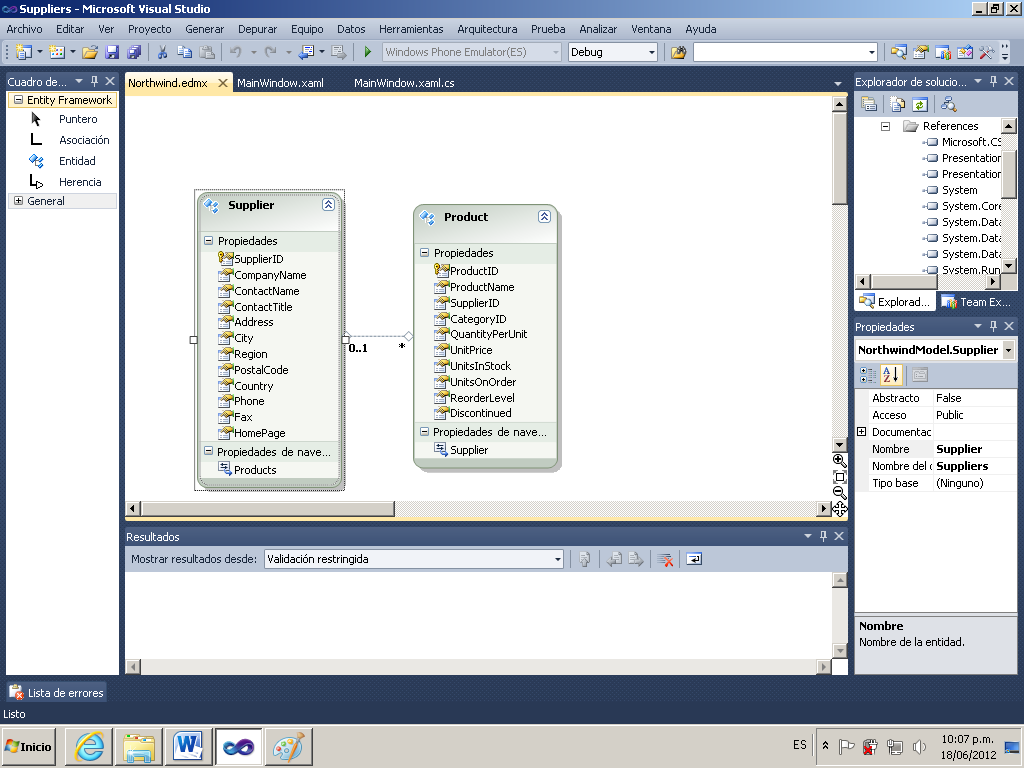
1. **Primero es conveniente utilizar la plantilla del proyecto entity data model**
2. **Cree un proyecto wfp y agregue nuevo elemento, será la plantilla entity data model**

****









Hasta aquí el asistente ha generado dos clases basadas en las tablas seleccionadas, con sus campos y restricciones, muy similar a los dataset tipados, a lo cual se le llama MAPEADO.

El archivo de configuración (app.config) guarda la información de la conexión, para ser modificado en caso de requerirlo y es donde se da cierta independencia,

Ojo : será necesario instalar el archivo de configuraciones application.exe.config (que es el nombre que le pone el compilador a app.config cuando el proyecto se compila) junto con el ejecutable de la aplicación. Para conectarnos a una base distinta solo requerirá modificar conection string desde un editor de texto.

1. Configuración de los controles de usuario

Se cambiaron de nombre los objetos (libro p. 726). Se cambio en (<Application… el StartupUri=”SupplierInfo.xaln”). Se cambio (<Window x:Class=”Suppliers.SupplierInfo”( y el (Title=”Supplier Information”). En la clase (public partial class SupplierInfo:window ) y el constructor (public SupplierInfo()). El código importante es el siguiente:

Primero se crea una plantilla para mostrar los datos del combo que se pone en window resources:

<Window.Resources>

<DataTemplate x:Key="SuppliersTemplate">

<StackPanel Orientation="Horizontal">

<TextBlock Text="{Binding Path=SupplierID}"/>

<TextBlock Text=" : "/>

<TextBlock Text="{Binding Path=CompanyName}"/>

<TextBlock Text=" : "/>

<TextBlock Text="{Binding Path=ContactName}"/>

</StackPanel>

</DataTemplate>

</Window.Resources>

Después se configuran las propiedades de los controles, el combo y el list siguientes:

<ComboBox Name="SupplierList" VerticalAlignment="Top" Width="570" IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding}" ItemTemplate="{StaticResource SuppliersTemplate}"/>

<ListView Height="243" HorizontalAlignment="Left" Margin="12,39,0,0"

Name="saveChanges" VerticalAlignment="Top" Width="568"

IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding}">

<ListView.View>

<GridView>

<GridView.Columns>

<GridViewColumn Width="75" Header="Product ID" DisplayMemberBinding="{Binding Path=ProductID}"/>

<GridViewColumn Width="225" Header="Name" DisplayMemberBinding="{Binding Path=ProductName}"/>

<GridViewColumn Width="135" Header="Quantity per unit" DisplayMemberBinding="{Binding Path=QuantityPerUnit}"/>

<GridViewColumn Width="75" Header="Unit price" DisplayMemberBinding="{Binding Path=UnitPrice}"/>

</GridView.Columns>

</GridView>

</ListView.View>

1. **Recuperacion de datos al formulario**

Añadir las referencias al espacio de nombres:

using System.ComponentModel;

using System.Collections;

**añadir loades a window**

<Window x:Class="Suppliers.SupplierInfo" . . . Loaded="Window\_Loaded">

En la clase SupplierInfo se crean los campos privados, el metodo window\_loaded y el evento selectionchanged del combo. La aplicación es ahora operativa y ante un cambio del combo

public partial class SupplierInfo : Window

{

public SupplierInfo()

{

InitializeComponent();

}

Crea una instancia de NorthWindEntities como se llamo a la plantilla entity para dconectarse con la base suppliers

private NorthwindEntities northWindContext = null;

private Supplier supplier = null;

private IList productsInfo = null;

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

this.northWindContext=new NorthwindEntities();

SupplierList.DataContext = this.northWindContext.Suppliers;

}

private void SupplierList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

this.supplier = SupplierList.SelectedItem as Supplier;

this.northWindContext.LoadProperty<Supplier>(this.supplier, s => s.Products);

this.productsInfo = ((IListSource)supplier.Products).GetList();

productsList.DataContext = this.productsInfo;

}

}

### Modificacion de datos en el origen

# Controles Ajax y html

## FilteredTextBoxExtender

AGREGAR UN CUADRO DE INTERACCION TIPO MSGBOX

‘primero se agrega un ajax script manager

‘luego se agrega un ajax update panel en donde iran los controles

‘luego por ejemplo al boton se le da en addextender y selecciona confirmbuttonextender

‘en el codigo html se le pone el mensaje, en la etiqueta confirmtext

## Marca de agua para textbox

‘arrastrar el TextBoxWaterMarkExtender dentro del cuadro de texto

‘en vista codigo verificar que su propiedad TargetControlId este el Id del TextBox, y que la propiedad WaterMarkText tenga el texto que ‘quiere que se muestre como marca de agua.

## Control calendario

‘arrastrar el CalendarExtender dentro del cuadro de texto

‘en vista codigo verificar que su propiedad TargetControlId este el Id del TextBox.

## Mascara de entrada

‘arrastrar el MaskedEditExtender dentro del cuadro de texto

‘en vista codigo verificar que su propiedad TargetControlId este el Id del TextBox, MaskedType en Date, Mask en 99/99/9999 y ErrorToolTypeEnables a True.

FORMATO DE NUMEROS EN MILES:

Mask="999,999.99" MaskType="Number" InputDirection ="RightToLeft"

## Validar mascara de entrada

‘arrastrar el MaskedEditValidator debajo del cuadro de texto

‘en vista codigo verificar que su propiedad ControlToValidate este el Id del TextBox, ControlExtender al Id de MaskedEditExtender y ‘Display en dynamic. Por ultimo los mensajes de error EmptyValueMessage, InvalidValueMessage y ToolTipMessage.

CARACTERES DE MASCARA

‘9 un numero, L una letra, $ una letra o espacio, ? cualquier carácter,

## FilteredTextBoxExtender

VALIDAR UN TEXTBOX PARA QUE ACEPTE SOLO NUMEROS

<asp:ScriptManager ID="asm" runat="server" />

<asp:TextBox ID="texMonto" runat="server"></asp:TextBox>

<asp:FilteredTextBoxExtender ID="FilteredTextBoxExtender1" runat="server" TargetControlID="texMonto" ValidChars="1234567890."></asp:FilteredTextBoxExtender>

## Textbox a mayúsculas

<asp:TextBox ID="LetraFP" runat="server" style="text-transform :uppercase"

# Sintaxis transact sql

REINICIAR EL AUTONUMERICO DE UNA TABLA SQL SERVER

DBCC CHECKIDENT (nombre\_tabla, RESEED,0)

PASAR DATOS DE UNA TABLA A OTRO

Update TablaA set Campo1=TablaB.Campo1

from TablaA, TablaB

where TablaA.Clave=TablaB.Clave

SELECCIONAR CAMPOS MAS RECIENTES DE UNA TABLA, POR FECHA O POR ID AUTONUMERICO

SI QUEREMOS SOLO EL REGISTRO MAS RECIENTE:

SELECT TOP (100) PERCENT NUMOBRA, MAX(IDAVANCE) AS MáxDeIDAVANCE, MAX(FECHAVISITA) AS MáxDeFECHAVISITA

FROM dbo.avances

GROUP BY NUMOBRA

ORDER BY MáxDeFECHAVISITA DESC

SI QUEREMOS LOS N ULTIMOS REGISTROS

SELECT numobra, idavance

FROM (SELECT numobra, idavance, row\_number() OVER (partition BY numobra

ORDER BY IDAVANCE DESC) AS rn

FROM avances) a

WHERE rn <= 2

OBTENER EL ID DEL REGISTRO INSERTADO

SqlConnection conn = new SqlConnection(@"Server = SERVERSUP\SQLEXPRESS; Database=escuelas; User='" + claUsuario.Usuario + "'; Password='" + claUsuario.Contraseña + "'");

SqlCommand comm1 = new SqlCommand("insert into dicestimaciones(folio,numero\_revision,fecha\_revision,observaciones,reviso) values(" +

intFolio + "," + intNumRevision + ",'" + d.ToString(format) + "','" + strObservaciones + "'," + intReviso + ")", conn);

SqlCommand comm2 = new SqlCommand("SELECT @@IDENTITY AS 'Identity'", conn);

SqlCommand comm3 = new SqlCommand("SELECT MAX(iddictamen) FROM dicestimaciones", conn);

conn.Open();

comm1.ExecuteNonQuery();

comm2.ExecuteNonQuery();

int idDic;

idDic =int.Parse(comm3.ExecuteScalar().ToString());

claGlobal.intLastId = idDic;

conn.Close();

# Campos base

HAY QUE MENCIONAR QUE LOS CAMPOS BASE REALIZAN VALIDACION EN SU IMPLEMENTACION DE LA CLASE, SERIA UTIL COMPLEMENTARLO CON MASCARAS O RESTRICCIONES EN LA INTERFACE DE USUARIO.

//Campos cortos obligados que no permiten caracteres restringidos

//NOMBRE, APELLIDOS

string strCampo;

public string Campo

{

private get

{

return strCampo;

}

set

{

string strExiSigno = "";

if (value == "" || value.Length > 50)

{

throw new ArgumentException("Debe proporcionar un Campo válido");

}

else if (strExiSigno.IndexOf(".") != -1 || strExiSigno.IndexOf("=") != -1 || strExiSigno.IndexOf("%") != -1

|| strExiSigno.IndexOf("\*") != -1 || strExiSigno.IndexOf("'") != -1 || strExiSigno.IndexOf("?") != -1

|| strExiSigno.IndexOf("(") != -1 || strExiSigno.IndexOf(")") != -1)

{

throw new ArgumentException("Ha intentado guardar un Campo con caracteres no permitidos");

}

else

{

strCampo = value.ToUpper();

}

}

}

//Campos largos no obligados que no permiten caracteres restringidos

//NOTAS, DIRECCION

string strCampo1;

public string Campo1

{

private get

{

return strCampo1;

}

set

{

string strExiSigno = "";

if (strExiSigno.IndexOf(".") != -1 || strExiSigno.IndexOf("=") != -1 || strExiSigno.IndexOf("%") != -1

|| strExiSigno.IndexOf("\*") != -1 || strExiSigno.IndexOf("'") != -1 || strExiSigno.IndexOf("?") != -1

|| strExiSigno.IndexOf("(") != -1 || strExiSigno.IndexOf(")") != -1)

{

throw new ArgumentException("Ha intentado guardar un Campo con caracteres no permit

}

idos");

else

{

strCampo1 = value.ToUpper();

}

}

}

//Numero telefonico, campo no obligado, en la conversion rechaza caracteres restringidos

//valida numeros aunque al final los pasa a texto.

object objTelCelular;

public object TelCelular

{

get

{

return objTelCelular;

}

set

{

bool isNum;

double retNum;

string strExisPunto;

strExisPunto = value.ToString();

if (strExisPunto == "")

{

//si no hay numero de telefono

objTelCelular = "";

}

else if (strExisPunto.Length > 10 || double.Parse(strExisPunto) <= 0)

{

throw new ArgumentException("El número no es válido, debe tener un formato correcto");

}

else

{

isNum = Double.TryParse(Convert.ToString(value), System.Globalization.NumberStyles.Any,

System.Globalization.NumberFormatInfo.InvariantInfo, out retNum);

if (isNum & strExisPunto.IndexOf(".") == -1)

{

objTelCelular = Convert.ToString(value);

}

else

{

throw new ArgumentException("El número no es válido, debe tener únicamente diez dígitos");

}

}

}

}

Esta validación es básica, existe otros métodos con expresiones regulare, que sin embargo tienen sus fallas por ser un campo muy diverso

//correo electronico, campo no obligado que no permite caracteres restringidos

string strMail;

public string Mail

{

get

{

return strMail;

}

set

{

string strMail1;

strMail1 = value;

if (value != "")

{

if (strMail1.IndexOf(".") == -1 || strMail1.IndexOf("@") == -1 || strMail1.Length <= 5 ||

strMail1.IndexOf("=") != -1 || strMail1.IndexOf("'") != -1 || strMail1.IndexOf("\*") != -1

|| strMail1.IndexOf("?") != -1 || strMail1.IndexOf("%") != -1 || strMail1.IndexOf("(") != -1

|| strMail1.IndexOf(")") != -1)

{

throw new Exception("La dirección de correo no tiene el formato correcto o contiene caracteres restringidos");

}

else

{

strMail = value;

}

}

else

{

strMail = value;

}

}

}

Este es un método que utiliza expresiones regulares, podría usarse como un método privado en la clase solo para auxiliar la validación del campo.

using System.Text.RegularExpressions;

private Boolean email\_bien\_escrito(String email)

{

String expresion;

expresion = "\\w+([-+.']\\w+)\*@\\w+([-.]\\w+)\*\\.\\w+([-.]\\w+)\*";

if (Regex.IsMatch(email, expresion))

{

if (Regex.Replace(email, expresion, String.Empty).Length == 0)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

//Cantidades GRANDES para uso de calculos GENERALES, por ejemplo inversiones, prestamos, etc

//CANTIDADES POSITIVAS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double douCantidad;

public double Cantidad

{

get

{

return douCantidad;

}

set

{

bool isNum;

double retNum;

isNum = Double.TryParse(Convert.ToString(value), System.Globalization.NumberStyles.Any,

System.Globalization.NumberFormatInfo.InvariantInfo, out retNum);

if (isNum && Convert.ToDouble(value) > 0)

{

douCantidad = value;

}

else

{

throw new ArgumentException("La cantidad no es correcta");

}

}

}

//Cantidades GARNDES para uso de calculos CIENTIFICOS

//CANTIDADES POSITIVAS Y NEGATIVAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double douCantidad1;

public double Cantidad1

{

get

{

return douCantidad1;

}

set

{

bool isNum;

double retNum;

isNum = Double.TryParse(Convert.ToString(value), System.Globalization.NumberStyles.Any,

System.Globalization.NumberFormatInfo.InvariantInfo, out retNum);

if (isNum)

{

douCantidad1 = value;

}

else

{

throw new ArgumentException("La cantidad no es correcta");

}

}

}

//fechas para un rango de edades, por ejemplo para trabajar

DateTime datFecha;

public DateTime Fecha

{

get

{

return datFecha;

}

set

{

DateTime datLimite = new DateTime(1950, 1, 1);

if (datFecha > DateTime.Now)

{

throw new ArgumentException("La fecha no puede ser mayor a la acual");

}

else if (DateTime.Compare(datFecha, datLimite) < 0)

{

throw new ArgumentException("La fecha es menor al límite de edad");

}

else

{

datFecha=value;

}

}

}

# Reportes

## MS Report

### Parámetros

//manualmente mediante redefinicion de adapter y dataset

//datosDataSetTableAdapters.paisesTableAdapter ta = new datosDataSetTableAdapters.paisesTableAdapter();

//datosDataSet.paisesDataTable tabla = new datosDataSet.paisesDataTable();

//ta.FillByGente(tabla, int.Parse(TextBox1.Text));

//ReportViewer1.LocalReport.DataSources.Clear();

//ReportDataSource datasource = new ReportDataSource("DataSet1", (DataTable)tabla);

//ReportViewer1.LocalReport.DataSources.Add(datasource);

//ReportViewer1.LocalReport.RefreshReport();

//con el diseñador, primero modificamos la consulta que se llama por medio del bindingsource, con ello

//aparece el toolstrip, de este solo se agrega el siguiente codigo al boton.

this.alumnosTableAdapter.FillBySemGpo(this.bdDataSet.alumnos, new System.Nullable<byte>(((byte)

(System.Convert.ChangeType(cLAVE\_SEM\_ACTUALToolStripTextBox.Text, typeof(byte))))),

new System.Nullable<byte>(((byte)(System.Convert.ChangeType(cLAVE\_GRU\_ACTUALToolStripTextBox.Text,

typeof(byte))))));

this.reportViewer1.RefreshReport();

### Expresiones

Iff: Su aplicación es similar a Excel, el if anidado va en la parte false de la primera iif.

=iif(Fields!CLAVE\_SEM\_ACTUAL.Value=1,"Primer Semestre",

iif(Fields!CLAVE\_SEM\_ACTUAL.Value=2,"Segundo Semestre",

iif(Fields!CLAVE\_SEM\_ACTUAL.Value=3,"Tercer Semestre",

iif(Fields!CLAVE\_SEM\_ACTUAL.Value=4,"Cuarto Semestre",

iif(Fields!CLAVE\_SEM\_ACTUAL.Value=5,"Quinto Semestre",

iif(Fields!CLAVE\_SEM\_ACTUAL.Value=6,"Sexto Semestre","Error !"))))))

## Clases

### String builder

Es excelente para construir complejas composiciones de texto con excelente rendimiento. Por ejemplo: crea un string con parámetros que toma de tres controles, y luego crea un string builder que agrega al primero a través de su método AppendLine (también existe el Insert y Remove, entre otros). Por ultimo agrega a la cadena cada content del elemento seleccionado dentro de un list y muestra al builder completo en un messagebox.show.

string nameTower = string.Format("member name {0} {1} from the tower at {2} rings the following methods:",

firstname.Text, lastname.Text, towername.Text);

StringBuilder details = new StringBuilder();

details.AppendLine(nameTower);

foreach (CheckBox cb in methods.Items)

{

if (cb.IsChecked.Value)

{

details.AppendLine(cb.Content.ToString());

}

}

MessageBox.Show(details.ToString(), "TITULO");

## Validación \*\* Complicaciones\*\*

Aplica para todos los lenguajes

El modelo de validación a través del LostFocus ofrece problemas porque se complica la lógica cuando se requiere vincular la validación entre controles. En el caso de wpf permite definir reglas de validación como parte del modelo de negocio, definiendo clases plenamente accesibles desde la vista xaml. Algo similar sucede en C# puro y en vb, ya que una opción es crear la validación en la implementación de las propiedades y campos de una clase.

# Implementacion IIS W10

**1.- Si el método de autenticación va a ser para el exterior o por medio de cuentas SQL Server, se debe habilitar únicamente la autenticación anónima. Luego los usuarios iusr e iis\_uisr deben tener permisos de lectura sobre la carpeta de la aplicación.**

**2.- sacar la pagina Default a la ruta raíz del sitio, para que iis la encuentre.**