



# VISUAL J#2005

## Express Edition

**Primera Edicion**

**Juan Ma. Osuna**

Microsoft  
**Visual J# 2005**  
Express Edition

## INDICE

### **1. UNIDAD I PROGRAMACION VISUAL ELEMENTOS BASICOS**

- 1.1. Introducción
- 1.2. Modelo De Solución
- 1.3. Variables
- 1.4. Tipos De Datos
- 1.5. Operadores Aritméticos
- 1.6. Jerarquía De Operaciones
- 1.7. Conceptos Básicos De Programación Orientada A Objetos
- 1.8. Requisitos Para Programación VISUAL J# 2005
- 1.9. Ambiente Integrado De Desarrollo
- 1.10. Programas, Formas Y Componentes
- 1.11. Aplicación
- 1.12. Otros Componentes
- 1.13. Ventanas En VISUAL J# 2005
- Apéndice: Palabras Reservadas VISUAL J# 2005

### **2. INSTRUCCIONES DE CONTROL DE PROGRAMA**

- 2.1. Introducción
- 2.2. Instrucciones De Control De Programa
- 2.3. Instrucciones Condicionales
- 2.4. Condiciones Simples
- 2.5. Operadores Relacionales
- 2.6. Instrucción If
- 2.7. Condiciones Compuestas
- 2.8. Instrucción Switch
- 2.9. Componente Listbox Y Combobox
- 2.10. Componente Checkbox Y Checkboxlist
- 2.11. Componente Radiobutton
- 2.12. Ciclo For
- 2.13. Ciclo While
- 2.14. Ciclo Do...While
- 2.15. Conclusiones Acerca De Ciclos

### **3. ARREGLOS**

- 3.1. Introducción
- 3.2. Arreglos
- 3.3. Arreglos Tipo Listas
- 3.4. Arreglos Tipo Tabla
- 3.5. Listas Visuales ( Listbox )

### **4. PROCEDIMIENTOS Y FUNCIONES**

- 4.1. Procedimientos
- 4.2. Parámetros
- 4.3. Variables Locales Y Globales
- 4.4. Funciones
- 4.5. Arreglos Como Parámetros

### **5. INT A LAS BASES DE DATOS**

- 5.1. Introducción
- 5.2. Modelos De Almacenamiento De Datos
- 5.3. Tablas

- 5.4. Tablas (Continuación)
- 5.5. Microsoft Access
- 5.6. ADO.NET
- 5.7. Selección O Despliegue
- 5.8. Inserción O Adición De Registros
- 5.9. Búsquedas
- 5.10. Filtros
- 5.11. Operaciones Con Campos
- 5.12. Bajas
- 5.13. Edición De Registros
- 5.14. Graphics O Imágenes



Microsoft  
**Visual J#** 2005  
Express Edition

# ELEMENTOS BASICOS

## Unidad I

### 1.1. INTRODUCCION AL VISUAL J# 2005

Información y Conocimiento son los dos elementos claves del nuevo milenio ninguna sociedad podrá alcanzar ni puede ignorar este nuevo esquema ya las naciones no se miden por su riqueza industrial, ni sus activos físicos, ni por su poder militar, sino por la cantidad de información que produce y consume, así como por la recombinación de información nueva en un conocimiento de grado superior.

Nuevos sistemas de información, tienden a ser cada vez de mayor alcance y complejidad sobre todo cuando se toman en cuenta las nuevas necesidades de información y conocimiento que demandan las nuevas organizaciones.

Nuevos sistemas de información son costosos en tiempos y recursos, la solución moderna de sistemas de información exigen herramientas y metodologías que resuelvan rápida, económica, eficiente y de manera global, problemas de información y conocimiento planteados por las organizaciones.

Además el pleno potencial del hardware tampoco es aprovechado plenamente y existe un considerable retraso con el software y sus aplicaciones generando lo que se conoce como "crisis del software".

Actualmente el paradigma de programación se ha enfocado a nuevas necesidades de modernos y globales sistemas de información basados en redes y mas aun en la red global de Internet, actualmente es mas importante poder concebir y construir sistemas de información con estas nuevas tecnologías de programación.

**VISUAL J# 2005** es un lenguaje de programación desarrollado por Microsoft muy apropiado para construir sistemas de información basados en red o mejor aun en Internet.

**.NET** es la nueva tecnología desarrollada y ofrecida por Microsoft que permite hacer más fácil la construcción y desarrollo de programas y aplicaciones para Internet.

En programación tradicional, modular o estructurada un programa describe una serie de pasos a ser realizados para la solución de un problema, es decir es un algoritmo.

En programación orientada a objetos ( OOP ) un programa es considerado como un sistema de objetos interactuando entre sí, ambientes de desarrollo visuales facilitan aun más la construcción de programas y solución de problemas, porque permiten abstraer al ingeniero de software de todo el GUI (interfase gráfica) del problema, que constituye más del 60% del código normal de un programa.

Es decir, en programación modular o estructurada un problema sencillo de información es descompuesto en una serie de módulos (llamados procedimientos o funciones) donde cada uno de ellos realiza una tarea específica, por ejemplo uno de ellos captura los datos, otro resuelve operaciones, etc.

En OOP todo problema aun aquellos sencillos de información, se consideran y resuelven como módulos de código gigante (clase) que contiene todo el código necesario (variables, procedimientos, funciones, interfaces, etc.) para solucionar el problema.

En programación visual (que también es heredera de OOP) la interfase con el usuario (pantallas) son generadas por el propio compilador y el ingeniero de software solo se concentra en resolver el problema planteado.

VISUAL J# 2005 de MICROSOFT es un compilador que permite usar cualquiera de los tres enfoques en la solución de problemas de información que puedan y deban ser resueltos empleando el computador y el lenguaje.

Para propósitos de aprendizaje usaremos el tercer enfoque, es decir programación en ambientes visuales y usando el lenguaje de programación VISUAL J# 2005.

## 1.2. MODELO DE SOLUCION.

En general un problema de información es posible entenderlo, analizarlo y descomponerlo en todos sus componentes o partes que de una u otra manera intervienen tanto en su planteamiento como en su solución.

Una herramienta rápida que nos permite descomponer en partes un problema para su solución, es el llamado modelo de solución, este consiste de una pequeña caja que contiene los tres elementos más básicos en que se puede descomponer cualquier problema sencillo de información, estas tres partes son:

- **LA PRIMERA PARTE** son todos los datos que el computador ocupa para resolver el problema, estos datos son almacenados internamente en la memoria del computador en las llamadas variables de entrada.
- **LA SEGUNDA PARTE** son todas las operaciones generalmente algebraicas necesarias para solucionar el problema, generalmente esta parte del modelo es una formula (o igualdad matemática, ej.  $X = y + 5$ ).
- **LA TERCERA PARTE** es el resultado o solución del problema que generalmente se obtiene de la parte de operaciones del modelo y dichos datos están almacenados en las llamadas variables de salida.

En resumen para todo problema sencillo de información es necesario plantearse las siguientes preguntas:

- Que datos ocupa conocer el computador para resolver el problema y en cuales variables de entrada se van a almacenar?
- Que procesos u operaciones debe realizar el computador para resolver el problema planteado?
- Que información o variables de salida se van a desplegar en pantalla para responder al problema planteado originalmente?

Como nota importante no confundir los términos datos, variables e información.

- **Datos** se refiere a información en bruto, no procesada ni catalogada, por ejemplo "Tijuana", "calle primera # 213", "15 años", "\$2,520.00", etc.
- **Variables** es el nombre de una localidad o dirección interna en la memoria del computador donde se almacenan los datos, ejemplo de variables para los casos del inciso anterior, CIUDAD, DIRECCION, EDAD, SUELDO, ETC.
- **Información** son datos ya procesados que resuelven un problema planteado.

### EJEMPLO DE MODELO DE SOLUCION

PROBLEMA 1.- Construir un modelo de solución que resuelva el problema de calcular el área de un triángulo con la formula área igual a base por altura sobre dos.

Variable(s) Entrada	Proceso u operación	Variable(s) salida
BASE ALTURA	$AREA = \frac{BASE * ALTURA}{2}$	AREA

PROBLEMA 2.- convertir la edad en años de una persona a meses.

PROBLEMA 3.- convertir pesos a dólares.

PROBLEMA 4.- calcular el área de un círculo con la formula:

$$AREA = PI * RADIO^2$$

PROBLEMA 5.- EVALUAR LA FUNCION  $Y = 5X^2 - 3X + 2$  PARA CUALQUIER VALOR DE X.

Observar para el caso de constantes fijas o conocidas (PI) no se debe dar como dato de entrada su valor, en cambio colocar directamente su valor dentro de la formula, en la parte de operaciones del problema.

Pero recordar también que existirán problemas sencillos donde:

No se ocupan entradas o no se ocupan operaciones, pero todos ocupan salida.

Una formula grande o muy compleja puede ser más segura y fácil de resolver, si es descompuesta y resuelta en partes, juntando al final los parciales para obtener el resultado final.

Un problema puede tener más de una solución correcta.

El problema no esta suficientemente explicado o enunciado, entonces, estudiarlo, analizarlo y construirlo de manera genérica.

#### TAREAS VISUAL J# 2005:

Construir los modelos de solución de los siguientes problemas:

PROBLEMA 6.- Convertir millas a kilómetros (caso normal).

PROBLEMA 7.- Convertir 125 metros a centímetros (no ocupa entradas).

PROBLEMA 8.- Se calcula que en promedio hay 4.7 nidos en cada árbol en la UABC, también se calcula que en cada nido existen un promedio de 5.8 pájaros, se pide calcular la cantidad total de nidos y de pájaros en los 227 árboles que existen en la UABC. (No ocupa entradas).

PROBLEMA 9.- La gorda Sra. López y sus 8 hijos solo compran una vez al mes su mandado en conocido supermercado, en dicha tienda el kilogramo de frijol cuesta \$8.75, el paquete de tortillas cuesta \$3.55 y el frasco de café vale \$14.25, si solo compran de estos tres productos para su mandado, calcular su gasto total. (Problema no claro).

PROBLEMA 10.- Capturar y desplegar los cinco datos mas importantes de un automóvil (no ocupa operaciones).

PROBLEMA 11.- La distancia Tijuana - Ensenada es de 110 kilómetros. Si un automóvil la recorre a una velocidad constante de 30 millas por hora, cuanto tiempo tarda en llegar. (1 milla = 1.609 Km.) (Dos maneras correctas de resolverlo).

PROBLEMA 12.-Evaluar la función  $y = 3x^2 + 2x - 5$  para cualquier valor de x. (caso normal).

PROBLEMA 13.-Evaluar la función  $y = -5x^3 - 3x^2 + 8$  para cuando x vale 4. (No ocupa entradas).

PROBLEMA 14.- Evaluar el factorial de cualquier numero usando la formula:  $n!=n!-1$ .

PROBLEMA 15.-La distancia que recorre un auto es de 50 Km. y su velocidad es de 30 millas por hora .&quest; Cuanto tiempo tardara en llegar?

PROBLEMA 16.-Encontrar la derivada de x para cualquier valor con la formula  $(d/dx(x)=1)$

PROBLEMA 17.-Calcular l interés que gana un capital de x pesos a una tasa de interés del 15% anual en un periodo de n años.

PROBLEMA 18.-Que aceleración tiene un tren que parte de Tijuana a 10 Km. /HR y pasa por ensenada una hora después a 50 Km. /HR.

PROBLEMA 19.-Calcular el numero de aulas en una escuela que tiene 10 edificios y cada edificio 3 pisos y cada piso 5 aulas, excepto un edificio que solo tiene dos pisos.

PROBLEMA 20.-Si en una escuela hay 30 maestros y 15 son hombres que atienden a 10 alumnos cada uno. Cuantas maestras hay?

PROBLEMA 21.-Calcular la corriente de un circuito con un voltaje de 15v y una resistencia de 6 ohms. Formula ( $I = V/R$ ).

PROBLEMA 22.-Calcular la normal estándar (z) dados los datos por el usuario: X=datos,  $\bar{x}$ =media, d=desviación. Formula ( $Z = (X - \bar{x}) / d$ ).

PROBLEMA 23.-Dado un numero (N) cualesquiera obtener su raíz y potencia cuadrada.

PROBLEMA 24.-Determinar la media de 5 números diferentes.

PROBLEMA 25.-Determinar la velocidad v requerida para recorrer una distancia d en un tiempo t. Formula ( $V = d * t$ )

PROBLEMA 26.-Determinar la pendiente de una recta. Formula ( $y = m x + b$ )

PROBLEMA 27.-Calcular la función de  $y = x^2 + 8x + 3$  para cualquier x

PROBLEMA 28.-Convertir minutos a horas.

PROBLEMA 29.-Aplicar la formula general para a=1, b=2, c=3.

PROBLEMA 30.-Se desea instalar un cable de red, el cliente pide 30 pies, considerando que se venden en metros, cuantos deberá comprar.

PROBLEMA 31.-Un campesino siembra trigo en un área cuadrada de 25 mts., ¿cual es el largo del cerco frontal en CMS.?

PROBLEMA 32.-Resolver  $x^2 + 15x - 8$  para cualquier variable (X).

PROBLEMA 33.-Convertir C a F.

PROBLEMA 34.-Si cada salón de la escuela tiene 40 alumnos y son 30 salones Cuantos alumnos son en toda la escuela?

PROBLEMA 35.-Si Juan trabaja 5 días a la semana y descansa 2 Cuantos días trabajo en 4 años?

PROBLEMA 36.-Si en una oficina se procesan 20 facturas cada 10 minutos cuantas se procesaran si se trabajan 5 horas?

PROBLEMA 37.-Si una empresa tiene \_\_\_\_\_ de activo y un pasivo de \_\_\_\_\_ Cual es su capital?. Formula ( $C = A - P$ )

PROBLEMA 38.-Calcule el voltaje de un circuito dada una intensidad I y una resistencia R. Formula ( $V=IR$ )



PROBLEMA 39.-Calcule la frecuencia de una onda que circula con un tiempo  $t$ . Formula ( $F=1/t$ )

PROBLEMA 40.-Calcule la potencia de un circuito con un voltaje  $V$  y una intensidad  $I$ . Formula ( $f = VI$ )

PROBLEMA 41.-Calcule el total que tendra que pagar una persona que va al cine dependiendo del no. de boletos a comprar y el precio.

PROBLEMA 42.-Calcule las anualidades que tendra que pagar una persona que pidió un préstamo. Dependiendo del tiempo que el elija y el interés por año. Formula ( $\text{Anualidad} = (\text{Préstamo}/\text{Tiempo}) + \text{interés}$ ).

PROBLEMA 43.-Determinar cuanto ganara una persona en base a la horas trabajadas. Tomando en cuenta el pago por hora.

PROBLEMA 44.-Convertir horas a segundos.

PROBLEMA 45.-Calcular la fuerza. Formula ( $\text{fuerza} = \text{trabajo} / \text{tiempo}$ )

### 1.3. VARIABLES.

**Identificadores** son conjuntos de letras y/o números que se utilizan para simbolizar todos los elementos que en un programa, son definibles por el usuario (programador o ingeniero de software) del mismo, como son las variables donde se almacenan datos, funciones (pequeños módulos con código), etiquetas, clases, objetos, etc.

En VISUAL J# 2005 un identificador es una palabra compuesta de letras y/o números de hasta 32 caracteres significativos, empezando siempre con una letra.

Una **variable** se define como un identificador que se utiliza para almacenar todos los datos generados durante la ejecución de un programa.

Existen ciertas reglas en cuanto a variables:

- Claras y con referencia directa al problema.
- No espacios en blanco, ni símbolos extraños en ellas.
- Se pueden usar abreviaturas, pero solo de carácter general.
- No deben ser palabras reservadas del lenguaje.

Ejemplos de buenas variables:

- Nombre, Edad, SdoDiario, IngMensual, Perímetro, Calif1, etc.

## 1.4. TIPOS DE DATOS.

A toda variable que se use en un programa, se le debe asociar (generalmente al principio del programa) un tipo de dato específico.

Un tipo de dato define todo el posible rango de valores que una variable puede tomar al momento de ejecución del programa y a lo largo de toda la vida útil del propio programa.

Los tipos de datos más comunes en VISUAL J# 2005:

### Primitive Data Types

Keyword	Description	Size/Format
<i>(integers)</i>		
byte	Byte-length integer	8-bit two's complement
short	Short integer	16-bit two's complement
int	Integer	32-bit two's complement
long	Long integer	64-bit two's complement
<i>(real numbers)</i>		
float	Single-precision floating point	32-bit IEEE 754
double	Double-precision floating point	64-bit IEEE 754
<i>(other types)</i>		
char	A single character	16-bit Unicode character
boolean	A boolean value (true or false)	true or false

Como se observa es muy similar a los de c o c++.

Para el caso de strings se debiera usar la Clase String que tiene dos constructores, de momento entenderemos esto último como dos maneras de crearse, Ej.;

- a) String nombre= new String();
- b) String ciudad= new String("Tijuana");

En este último caso se crea la string y se inicializa con un dato o valor.

Uno de los trabajos más frecuentes que se encuentran a lo largo del curso es la conversión de un tipo de dato a otro, especialmente tipos numéricos a tipos strings y viceversa, afortunadamente MICROSOFT proporciona dos mecanismos de conversión:

El primero de ellos es son objetos derivados de la clase CONVERT y sus principales métodos son:

```
System.Convert.ToString(entero);
System.Convert.ToChar(entero);
System.Convert.ToByte(entero);
System.Convert.ToBoolean(entero);
```

```
System.Convert.ToInteger(entero);  
System.Convert.ToDouble(entero);
```

This class returns a type whose value is equivalent to the value of a specified type. The supported base types are Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime and String. Fuente microsoft.net

El segundo método es usar el método PARSE que traen las clases numéricas de las cuales se derivan los tipos de datos de java, estos métodos son:

```
System.Int32.Parse(variable);  
System.Double.Parse(variable);
```

## 1.5. OPERADORES ARITMÉTICOS

Un **operador** es un símbolo especial que indica al compilador que debe efectuar una operación matemática o lógica.

VISUAL J# 2005 reconoce los siguientes operadores:

Operador	Operation
+	SUMA
-	RESTA
*	MULTIPLICACION
/	DIVISION
%	MODULO O RESIDUO

El operador (%) devuelve el residuo entero de una división entre enteros, ejemplo;

```
// Área de declaración
Variable alfa;
// Área de operaciones
Alfa = 23 % 4;
// Área de despliegue
Desplegar alfa; ---> El resultado en pantalla es 3
```

```
Otro ejemplo;
Alfa = 108 % 10;
Desplegar alfa; --> El resultado en pantalla es 8
```

Para resolver los problemas de potencias y raíces se usan ciertas instrucciones especiales que proporciona el lenguaje llamadas funciones matemáticas, en VISUAL J# 2006 existe toda una librería o más correctamente dicho un OBJETO especializado en instrucciones o funciones matemáticas.

Recordar que todas las funciones reciben uno o más datos o valores y regresan siempre un resultado, una de estas funciones matemáticas es:

a) Potencias por ejemplo  $5^2$  se resuelve usando el objeto System.MATH y su método Pow (base, exp). (Ver apéndice final capítulo)

Esta función ocupa dos valores o datos (base y exp.) ambos de tipo doble y regresa un resultado también de tipo doble, ejemplo;

```
Vary pot;
pot = System.Math.Pow(5,2);
disappear pot;
```

b) Raíces solo recordar la ley de exponentes que dice:

$$\sqrt[n]{a^m} = a^{m/n}$$

$\sqrt[3]{5^8}$  = usar System.Math.Pow(5,8/3.0);

$\sqrt{9}$  = usar System.Math.Pow(9,1/2.0) ;

TAREAS VISUAL J# 2005:

Expresar las siguientes funciones en notación algebraica de VISUAL J# 2005

1.  $Y = 5X^2 - 3X + 2$
2.  $y = -5x^3 - 3x^2 + 8$
3.  $y = 3x^3 - \sqrt[3]{x} + 4x^2$

## 1.6. JERARQUIA DE OPERACIONES.

El problema de no tomar en cuenta la jerarquía de los operadores al plantear y resolver una operación casi siempre conduce a resultados muchas veces equivocados como estos:

Ejemplos:

$$2 + 3 * 4 = 20(\text{incorrecto}) = 14(\text{correcto})$$

si  $\text{calif1}=60$  y  $\text{calif2}=80$

En programa se usa

$\text{Promedio} = \text{calif1} + \text{calif2} / 2$

da como resultado promedio = 100

Recordar siempre, que antes de plantear una formula en un programa se deberá evaluar contra el siguiente:

### Orden de operaciones:

- 1.- Paréntesis
- 2.- Potencias y raíces
- 3.- Multiplicaciones y divisiones
- 4.- Sumas y restas
- 5.- Dos o más de la misma jerarquía u orden, entonces resolver de izquierda a derecha

Nota: Si se quiere alterar el orden normal de operaciones, entonces usar paréntesis.

Nota: Tampoco es bueno usar paréntesis de más en una operación, esto solo indica que no se evalúo bien la formula, como en el siguiente ejemplo;

$$\text{area} = (\text{base} * \text{altura}) / 2.0;$$

Aquí los paréntesis están de más, porque por orden de operaciones, multiplicación y división tienen la misma jerarquía y entonces se resuelven de izquierda a derecha, en otras palabras ni que falten paréntesis ni que sobren paréntesis.

## 1.7. CONCEPTOS BÁSICOS DE PROGRAMACION ORIENTADA A OBJETOS.

Para nuestro propósito en general un objeto puede definirse como cualquier cosa, ente o entidad física o lógica de información.

En este sentido todos los elementos materiales o inmateriales pueden clasificarse como objetos.

En particular cualquier objeto considerado presenta los siguientes tres elementos:

- a) **Propiedades:** Son las características propias de un objeto, estas propiedades o atributos son los que permiten diferenciar o individualizar un objeto de otro objeto ya sea de la misma o diferente clase o categoría.

Las propiedades más generales son forma, color, tamaño, peso, etc., pero ya en particular:

Chamarra -> Marca, material, precio, color, tamaño, etc.

Alumno -> Matricula, nombre, edad, domicilio, etc.

Gato -> Raza, nombre, color, edad, etc.

VentanaWindows-->Tamaño, Color, font, etc.

- b) **Métodos:** Son las conductas o comportamientos apropiados a la naturaleza del objeto.

Así como las propiedades son el ser (que es) del objeto, los métodos son el hacer (que hacer) del objeto.

Ejemplo de métodos:

Gato ---> Maullar(), comer(), correr(), saltar(), etc.

Alumno---> Estudiar(), comer(), asistir clase(), pintar()

Cuaderno-->Esescrito(), esrayado(), esborrado(), etc.

VentanaWindows--> Abrir(), cerrar(), maximizar(), etc....

- c) **Eventos:** Es la relación (de varias maneras) que se puede dar entre dos objetos ya sean de la misma o diferente clase.

Un evento se manifiesta como una interacción entre dos objetos, en general al momento de la relación al mismo tiempo se dará una reacción o respuesta por parte de los dos objetos que se manifiestan como una serie, cadena o conjuntos de métodos propios que se activan o disparan, ejemplo:

Evento	Relación	Métodos que se activan
gato detecta gata	detectar	maullar(), correr(), oler()
gato detecta perro	detectar	bufar(), saltar(), correr()
maestro enseña alumno	Enseñar	pasar lista(), preguntar(), etc
Raton click Windows	click	maximizar(), cerrar()
Raton dblclk Windows	dblclk	minimizar(), etc

Un **Programa en VISUAL J# 2005** se puede considerar como un conjunto de una o más formas o ventanas, donde cada una de ellas contiene un conjunto de objetos, componentes o controles.



Un **componente** o propiamente dicho un **control** es un objeto que se especializa en una tarea específica por ejemplo hay controles especializados en desplegar textos o mensajes, otros controles se especializan en desplegar imágenes o vídeos, otros en manipular directorios o archivos en disco, etc.

Pero en general tanto las formas como los controles no dejan de ser objetos en programación y por tanto tienen sus propiedades, métodos y están sujetos a eventos.

### **1.8. REQUISITOS PARA PROGRAMACION.**

Para poder construir programas o aplicaciones visuales con VISUAL J# 2005, se ocupa lo siguiente:

- a) 1.- Una PC con sistema operativo WINDOWS 2000 o WINDOWS XP.
- b) 2.- Dicha Maquina deberá actualizarse con los últimos up dates de Microsoft, solo cargar el Explorer y en la opción TOOLS, usar la opción WINDOWS UPDATE, bajar todos los parches críticos e instalarlos.
- c) 3.- Se deberán bajar e instalar, también de Microsoft el compilador de VISUAL J# 2005 de la siguiente dirección <http://lab.msdn.microsoft.com/express/vJSharp/default.aspx>

Al instalarse va a pedir que se baje e instale también NET FRAMEWORK 2.0 y SQL SERVER 2005, cuando menos deberán bajar e instalar el framework y recordar que son como 100 mb entre las tres aplicaciones, así que hacerlo con paciencia y de preferencia en la noche.

Cuando lo instalen subirse a Microsoft y registrarlo o dentro de 30 días de instalado va a dejar de funcionar.

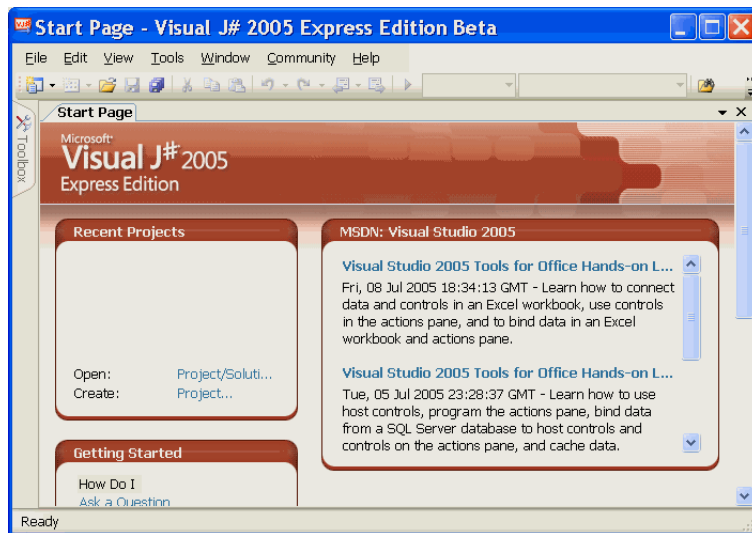
## 1.9. IDE COMPILER AMBIENTE INTEGRADO DE DESARROLLO.

Entradas o capturas de datos y salidas o despliegues de información o resultados son de los procesos más comunes en cualquier tipo de problema de información, estos procesos o instrucciones varían de acuerdo a los lenguajes y ambientes de programación a usar.

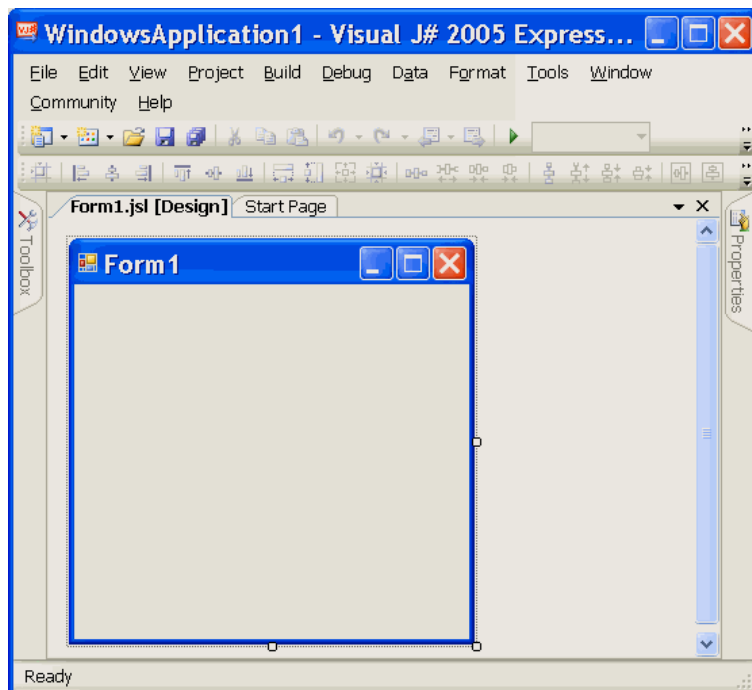
El lenguaje y ambiente de programación a utilizar es de tipo visual, y muchos de los problemas asociados a entradas y salidas se encuentran ya resueltos por el propio compilador.

El ambiente de construcción de programas a usar o ide compiler es el siguiente:

Solo cargarlo en pantalla ejecutando el VISUAL J# 2005 que se encuentra en la barra de start de Windows.



Usar Ahora la opción FILE, NEW PROJECT y seleccionar WINDOWS APLICACION, se tiene ahora la siguiente pantalla que contiene el objeto principal del programa (**FORM1 o VENTANA**).

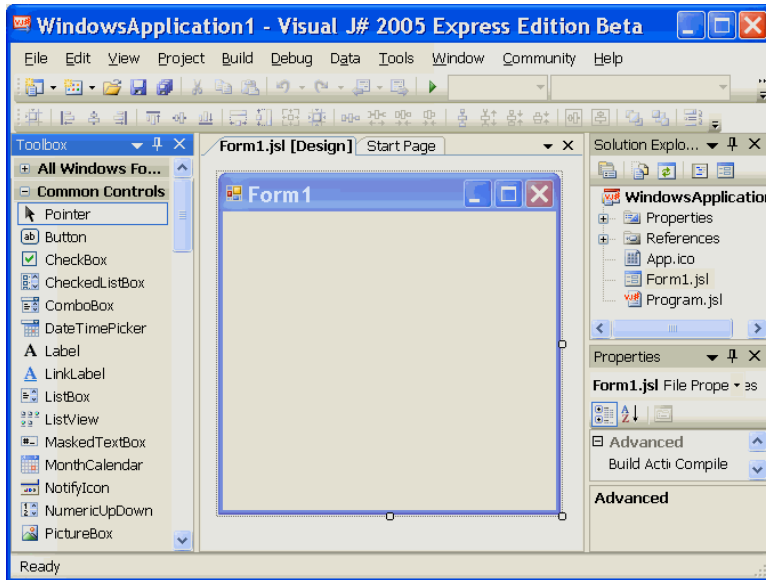


Ahora al IDE le agregamos los tres siguientes elementos, usando las opciones:

- VIEW Properties Windows (ventanilla de propiedades de los objetos).
- VIEW Solution Explorer (Ventanilla de Administración del Proyecto).
- VIEW ToolBox (ventanilla de herramientas).

Observar que existen otras VISTAS (VIEW'S) que se usaran mas adelante en el curso.

Se tiene ahora:



Sus elementos básicos son:

- 1.- La barra de menús (file, edit, etc.).
- 2.- La barra de herramientas (icono de grabar, run, forma, etc.).
- 3.- La barra o paleta de componentes (TOOLBOX).
- 4.- La ventana de Propiedades (Windows Properties)
- 5.- El Administrador de proyectos (Solution Explorer).
- 6.- La forma activa o principal (Form1).



Es sobre esta última forma o ventana donde se construirá el programa y esta forma se convertirá en ventana al momento de ejecutarse el programa.

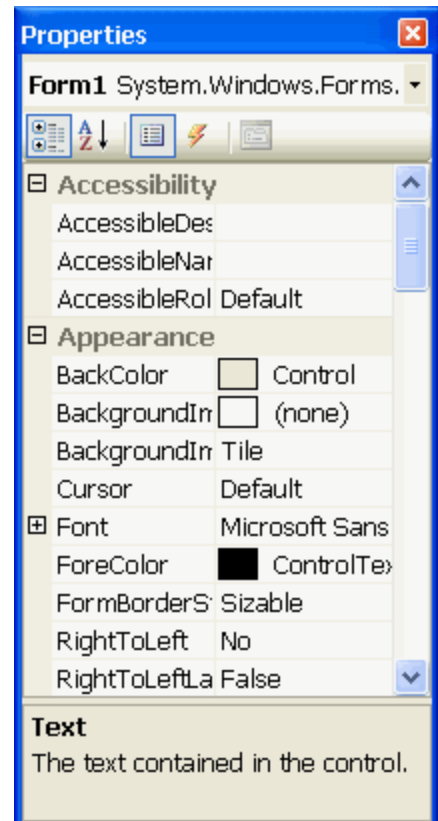
Es decir será la primera ventana que el usuario ve al momento de ejecutarse el programa, su nombre es **Form1**.

Esta forma o ventana es un objeto de VISUAL J# 2005, y como todos los objetos de J# y del universo, la forma o ventana tiene asociados propiedades y eventos.

Propiedades son todas las características particulares que diferencian un objeto de otro objeto, las propiedades o características mas comunes son forma, tamaño, color, etc., para objetos en VISUAL J# 2005, estas propiedades se modifican o individualizan usando la ventana de propiedades, que es la parte del compilador que las contiene.

Recordar que es en esta ventana de propiedades es donde se podrá modificar las propiedades del objeto, en este ejemplo Form1 o VENTANA. Pero recordar también que la ventanilla de propiedades es quien contiene

el icono de EVENTS (usar el icono  para activar las propiedades y usar el icono  para ver los eventos que puede detectar form1), que como ya



se explico en el tema anterior son los EVENTOS quienes contendrán el CÓDIGO DEL PROGRAMA.

También se pueden modificar las propiedades dentro de un programa, usando instrucciones apropiadas, mismas que llevan el siguiente formato:

Nomobjeto.propiedad = nvovalor;

ej.; Form1.BckColor = Pink; <-- y existen muchos colores ver la opciones en el inspector de objetos.

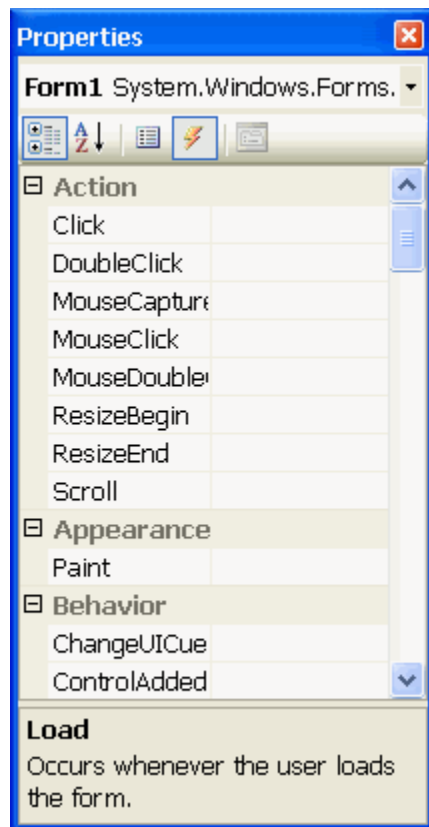
Eventos, son todos aquellos sucesos de carácter externo que afectan o llaman la atención del objeto, para estos casos la forma o ventana:

- Debe tener capacidad de detectar el evento.
- Aun más importante debe tener capacidad de reaccionar y emitir una respuesta, mensaje o conducta apropiada al evento detectado.

Evento es por ejemplo que otro objeto llamado humano, pulse el objeto tecla ESC, o haga click derecho con el objeto ratón en alguna parte de la ventana , etc. , es en estos casos, cuando la ventana detecte un evento de estos, la propia forma deberá responder de manera apropiada.

Esta respuesta no es automática, sino son la serie de instrucciones del lenguaje (o programa) que los ingenieros de software diseñan (o programan), en otras palabras son los eventos quienes contendrán los programas.

Es también la ventana de propiedades quien contiene todos los posibles eventos asociados a la forma.



Para nuestros primeros programas en VISUAL J# 2005 de **Form1** solo se usaran propiedades sencillas como color font etcétera y no se usaran sus eventos.

Un programa o problema de información en VISUAL J# 2005 , no es mas que una o mas formas o ventanas, donde cada una de ellas contiene elementos u objetos especiales llamados objetos o componentes, dichos componentes VISUAL J# 2005 los proporcionara a través de la paleta de componentes (TOOLBOX).

Observar que esta caja de componentes tiene varias categorías con su propio grupo de componentes que se irán usando a lo largo del curso.

Aun más toda la interfase que se quiera manejar con el usuario del programa no consistirá más que de una colección de componentes agrupados en una forma o ventana.

Para incorporar un componente a una forma solo basta seleccionarlo con un click derecho en su icono y luego arrastrarlo hasta el lugar donde quedara dentro de **Form1**.

También estos controles o componentes son objetos de VISUAL J# 2005 y como tales también tienen asociados propiedades y eventos tales como los tiene la forma principal, solo que existen pequeñas variaciones en cuanto a sus propiedades y eventos propios con respecto a **Form1**.

Recordar además, que es la ventana o pagina de propiedades en primera instancia quien permite asociar o modificar propiedades específicas tanto a una forma como a un componente.

Ya en segunda instancia las propiedades de formas y componentes se pueden modificar también directamente dentro de un programa, usando instrucciones como las ya descritas en párrafos muy anteriores. Analizaremos ahora los primeros componentes, que también se usaran para construir o diseñar nuestro primer programa en VISUAL J# 2005 de tipo Visual o de Ventanas.

### 1.10. PROGRAMAS, FORMAS y COMPONENTES.

Recordar que programas visuales, se construyen usando los siguientes elementos:

- 1) Un objeto ventana o **form1**.
- 2) Objetos que permitan capturar y desplegar datos, de momento se usaran los dos objetos de entrada / salida mas comunes:
  - a) Objeto **label** <-- permite desplegar textos o mensajes estáticos
  - b) Objeto **textbox** <-- permite tanto capturar datos, así como desplegar el resultado de operaciones (recordar que **textbox's**) solo capturan o despliegan un dato a la vez, recordar también que todos los datos que entren y salgan de un textbox o un label son de tipo **string**.
- 3) 3.- objeto **button** <-- es el componente principal de la forma, contiene el código principal del programa y su activación por el usuario provoca que se realicen los principales procesos del problema planteado (aquí es donde se capturan datos, se realizan operaciones, etc.).

De este componente se maneja su propiedad **Text** para etiquetarlo con la palabra "OK" o "ACEPTAR" o "EXE" y su evento Click para activarlo, es en dicho evento donde se construirá el código del programa.

Recordar que aunque no es un componente necesario en los programas ya que el código se puede asociar o pegar a cualquier evento de cualquier forma o componente del programa, Microsoft ya acostumbro a todos los usuarios al botón OK, de acuerdo, OK.

Recordar que todos los componentes vistos incluyendo la propia forma y ventana tienen muchas propiedades que se pueden asignar o modificar y también tienen muchos eventos a los cuales les podemos asociar o cargar el código del programa, todos estas propiedades y eventos las podemos acceder dentro de la pagina de propiedades, es decir, por favor vean, analicen y usen la ventana o pagina de propiedades asociado al componente, porque a lo largo de este curso son muy pocas las propiedades que se estudian y también muy pocos los eventos que se cargan y son muchos los que hay que aprender y sobre todo usar.

Regresando a **Form1**, es sobre esta forma donde se construirá el programa y esta forma se convierte en ventana al momento de ejecutarse el programa.

Es decir será la primera ventana que el usuario ve al momento de ejecutarse el programa, su nombre es **Form1**.

Esta forma o ventana es un objeto de VISUAL J# 2005 y como todos los objetos de J# y del universo, la forma o ventana tiene asociados propiedades y eventos.

Propiedades son todas las características particulares que diferencian un objeto de otro objeto, las propiedades o características mas comunes son forma, tamaño, color, etc., para objetos en J# 2005, estas propiedades se modifican o individualizan usando la pagina de propiedades, que es la parte del programa que las contiene.

Recordar que se pueden modificar las propiedades dentro de un programa, usando instrucciones apropiadas, mismas que llevan el siguiente formato:

Nomobjeto.propiedad = nvovalor;

**ej.;** Form2.BackColor=Yellow;

Eventos, son todos aquellos sucesos de carácter externo que afectan o llaman la atención del objeto, para estos casos la forma o ventana u objeto:

Debe tener capacidad de detectar el evento

Aun más importante debe tener capacidad de reaccionar y emitir una respuesta, mensaje o conducta apropiada al evento detectado.

Evento es por ejemplo que otro objeto llamado humano pulse el objeto tecla ESC, o haga click derecho con el objeto ratón en alguna parte de la ventana , etc. , es en estos casos, cuando la ventana u objeto detecte un evento de estos la propia forma u objeto deberá responder de manera apropiada.

Esta respuesta no es automática, sino son la serie de instrucciones del lenguaje (o programa) que los ingenieros de software diseñan(o programan) en otras palabras son los eventos quienes contendrán los programas.

Es también la PAGINA DE PROPIEDDAES, quien contiene todos los posibles eventos asociados a la forma.

Para los primeros programas en VISUAL J# 2005 solo se usaran propiedades sencillas como color, font, etc. de Form1 y los objetos ya mencionados (LABEL, TEXTBOX) y no se usan de momento los eventos que puede detectar Form1.



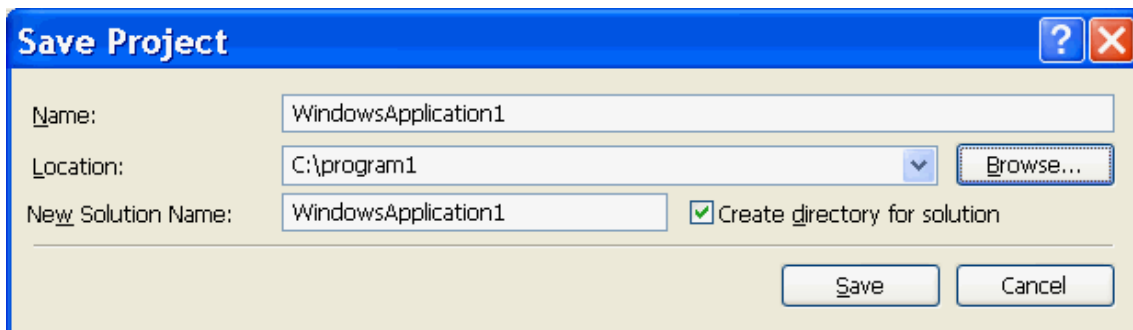
### 1.11. APLICACIÓN.

Resolvemos el problema de calcular el área de un triángulo con la formula  $\text{área} = \text{base} * \text{altura} / 2$   
Para resolver este problema se ocuparan los siguientes objetos.

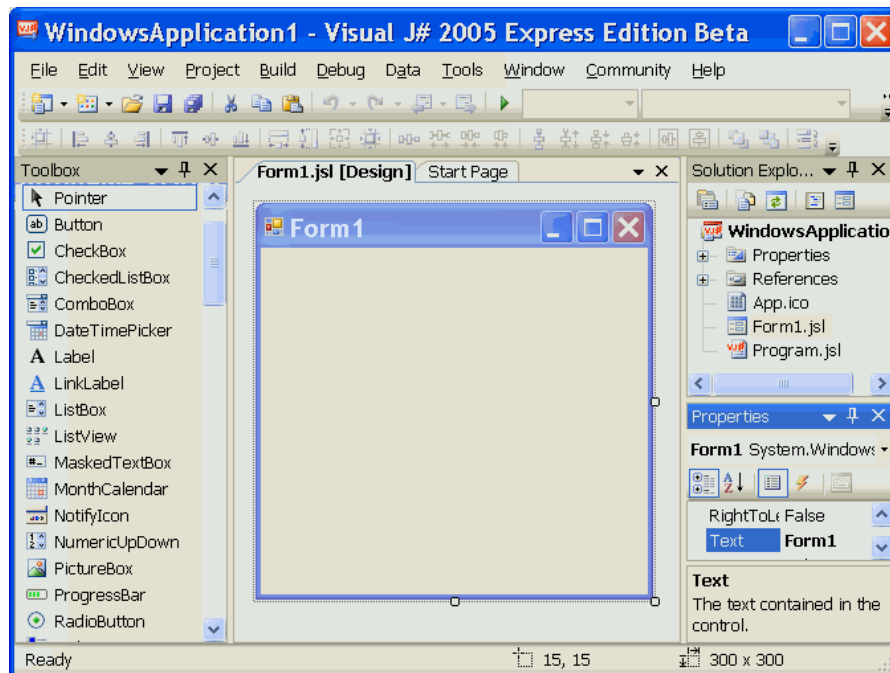
- Una ventana de windows (**Form1**) para contener a todo el resto de componentes.
- Tres label's o etiquetas para desplegar mensajes apropiados al problema.
- Tres componentes **Textbox** para capturar datos (base y altura) y para desplegar el resultado (área).
- Un componente **Button** para que active y contenga el código del problema y/o programa.

#### PROCEDIMIENTO DETALLADO

- 1.- Crear un folder o directorio en su disco duro **C:\** llamado **PROGRAM1**
- 2.- Cargar VISUAL J# 2005 desde la barra de start de windows
- 3.- Usar la opcion **FILE --> NEW PROJECT --> WINDOWS APPLICATION**
- 4.- Usar Ahora la opción **FILE --> SAVE ALL** y en la pantalla de grabación que sale ponerle un nombre al proyecto (dejar el default) y apuntarlo al directorio que se hizo de programas:



- 5.- Queda ahora la pantalla de trabajo o diseño o construcción del programa



Recordar que deberán agregar **Toolbox**, página de propiedades y explorador de soluciones usando la opción VIEW de la barra del menú.

6.- Dentro del programa se estarán cargando todos los objetos que usa el programa y por ejemplo VISUAL J# 2005 a los textbox's que se estarán usando VISUAL J# 2005 los nombra con los DEFAULT de textBox1 y textBox2, etc. con esas mayúsculas y minúsculas así que estará difícil estar recordando su nombre de default.

7.- Arrastrar y acomodar desde la paleta de herramientas (TOOLBOX) a Form1 (ventana principal) 2 (tres) componentes LABEL y cargarles su propiedad TEXT con las palabras BASE, ALTURA y AREA respectivamente, para escribir esto solo hacerlo usando la cajita que esta a un lado de la propiedad dentro de la página de propiedades.

NOTA: si de alguna manera el editor los manda al código del programa, observar que arriba de FORM1 hay tres pestañas ( FORM.jsl <--CONTIENE CÓDIGO; FORM1 DESIGN <--CONTIENE FORM1 VISUAL y START PAGE), solo click en design para regresar al ambiente visual.

8.- Arrastrar y acomodar ahora los componentes textBox de los cuales en su propiedad TEXT, limpiarla y dejarla en blanco y en su propiedad (NAME) cargarlos con las palabras BASE1, ALTURA y AREA respectivamente.

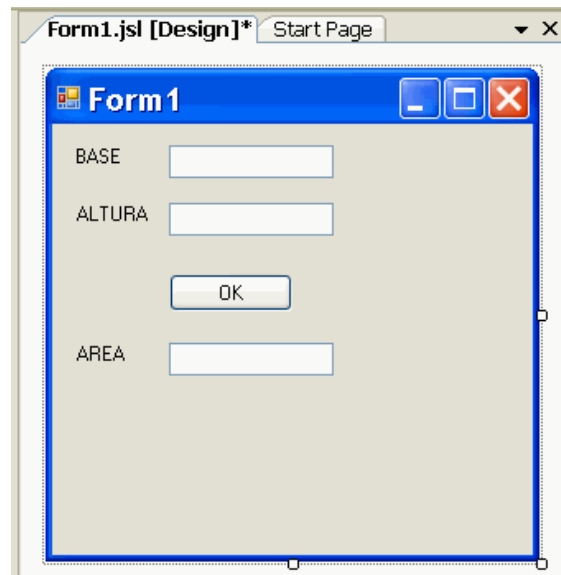
Nota: usamos BASE1 como nombre porque la palabra apropiada BASE es palabra reservada de VISUAL J# 2005 y no puede usarse como nombre de variable o NAME de componente.

LABELS's y BUTTONS solo cargar propiedad TEXT, TEXTBOX's cargar propiedad NAME

Es muy importante recordar que VISUAL J# 2005 es case-sensitive es decir si puede diferenciar entre mayúsculas y minúsculas, es decir si ustedes cargan la propiedad (name) con la palabra base1 en puras mayúsculas dentro del programa se tendra que usar también con puras mayúsculas, están advertidos.

9.- Arrastrar y acomodar ahora un componente o control BUTTON, poner su propiedad text=OK.

La pantalla diseñada es:



10.- ahora colocaremos el siguiente código dentro del evento click de BUTTON (solo hacer un click rápidamente arriba de BUTTON OK)

```
private void button1_Click(Object sender, System.EventArgs e)
```

```

{
// Declaracion variables
double temp;
// Realizando operaciones
temp = ( Double.parseDouble( BASE1.get_Text()) * Double.parseDouble(ALTURA.get_Text()))/2 ;
// cargando y formateando el resultado
AREA.set_Text( String.Format("{0:f}", (System.Double)temp));
}

```

Para escribir este código solo CLIK dentro de BUTTON1 y VISUAL J# 2005 los mandara a la pantalla de código, donde ya estará cargado el evento button1\_clik().

Atención solo se escribe el código que esta adentro de los corchetes del buttonclik (), el evento button1.click(){} ya lo escribe VISUAL J# 2005 por default y respetar todas las mayúsculas y minúsculas dentro del código.


Los dos parámetros que van dentro de button\_click (OBJETO, variable e)--> son para indicarle al compilador que se estarán enviando OBJETOS entre la forma o ventana y el usuario del programa ( LOS TEXTBOXS son los objetos y primero se mandan vacíos al usuario y luego el sistema los recoge con datos desde el usuario), La VARIABLE "e" (environment) es donde el sistema o compilador los va almacenando temporalmente, si ya vieron algo de msdos y conocen el comando set ahí observaran los textbox's con sus datos.

Regresando al código:

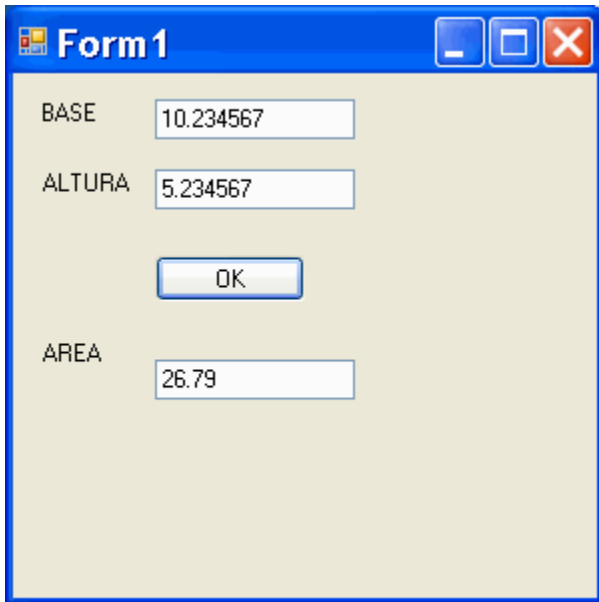
- Son las propiedades TEXT de los TEXTBOX'S quienes contienen los datos tanto capturados como los de salida.
- Observar que estos componentes y muchos otros que se verán mas adelante se lee su valor con el método get\_text() y se carga su valor con el método set\_text(valor)
- Recordar además que cualquier propiedad TEXT de cualquier CONTROL o COMPONENTE solo maneja datos de tipo TEXTO o STRING y por tanto para poder realizar operaciones con los TEXTBOX.TEXT se usaron dos métodos diferentes:
  - CLASENUMERICA.PARSECLASENUMERICA (dato string, var string, componente.text) -->para sacar u obtener su valor numérico, revisar su tema de TIPOS DE DATOS para ver cuales clases numéricas tienen su método PARSE.
  - CLASENUMERICA.ToString(var numérica) --> se uso para convertir el valor numérico a tipo string, para poderlo almacenar en la propiedad TEXT del TEXTBOX.
- Se usa una variable temporal de tipo numérico para procesar datos numéricos, la recomendación es tratar de usar variables temporales en la menor cantidad posible.
- Se esta usando String.Format(mascarilla) para formatear los datos de salida, los símbolos básicos que se pueden usar son:

<b>Format specifier</b>	<b>Name</b>
C or c	Currency
D or d	Decimal
E or e	Scientific (exponential)
F or f	Fixed-point
G or g	General
N or n	Number
P or p	Percent
R or r	Round-trip
X or x	Hexadecimal

Fuente: ayuda VISUAL J# 2005

11.- Ya cargada la forma con sus componente y el evento click con su código respectivo, grabar el programa con la opción FILE->SAVE o usar el icono de grabación (el disquito o disquitos que esta en la barra de iconos arriba) y luego ejecutar el programa usando el icono de RUN  que también se encuentra en la barra de herramientas arriba.

12.- El programa en ejecución:



Si buscan muy bien dentro del folder project1 se encontraran con muchos archivos que creo VISUAL J# 2005, dos de ellos son los interesantes: FORM1.jsl que contiene todo el código fuente, es decir el programa original y windowsapplication1.exe (dentro del folder bin\debug) que es el archivo ejecutable basta darle un click desde el explorer de windows y se ejecutara solo sin necesidad de que este cargado el VISUAL J# 2005.

#### TAREAS VISUAL J# 2005:

1. Construir el programa que convierte la edad en años de una persona a meses.
2. Construir un programa que evalúe la siguiente función  $Y = 5X^2 - 3X + 2$  para cualquier valor de x.
3. Determinar el perímetro de un triangulo rectángulo, dadas las longitudes de la hipotenusa y un cateto.
4. dos problemas cualesquiera de los que vienen en el tema de modelo de solución.

## 1.12. OTROS COMPONENTES.

En este tema analizamos algunas instrucciones de VISUAL J# 2005 que nos permitirán facilitar algunas tareas que están pendientes y también algunos controles y componentes que también nos darán buena ayuda.

### Componentes De Agrupamiento Control Panel

El componente PANEL nos permite agrupar y contener una serie lógica de controles normales en una ventana, es decir una ventana se puede dividir en dos o tres áreas lógicas de la pantalla, cada área es un panel.

Los paneles tienen propiedades interesantes como Visible = true or false; esto nos permite tenerlo oculto hasta el momento que se ocupe.

Los controles normales se manejan de manera normal, ejemplo:

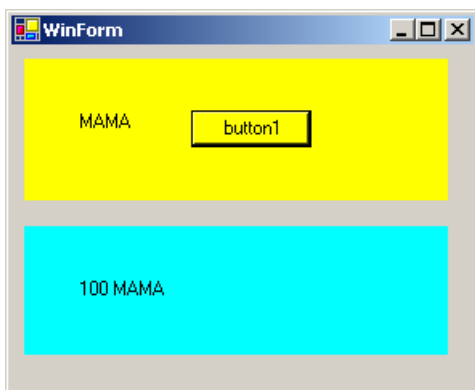
Nota: Poner los dos panels dentro de la forma al segundo panel poner su propiedad visible=false

Coding:

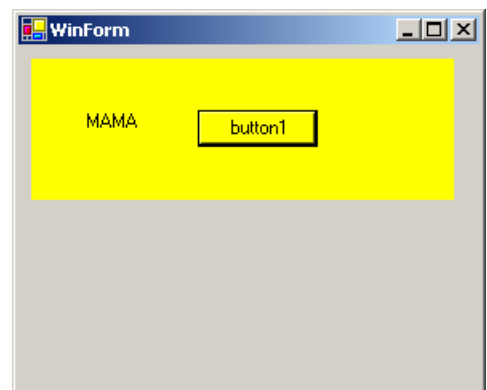
```
-----
private void button1_Click(Object sender, System.EventArgs e)
{
    panel1.set_Visible(true);
    label4.set_Text("100" + " MAMA");
    // observar que dos strings se encadenan usando simbolo +
}
-----
```

Pantallas de corrida:

a)



b)



Observar que existen otros controles de agrupamiento en TOOLBOX de los cuales seria interesante que los usaran, revisaran e implementaran.

### TAREAS VISUAL J# 2005:

Agregar Panel o algún otro componente de agrupamiento a algunos de los problemas ya construidos

### 1.13. VENTANAS EN VISUAL J# 2005.

El siguiente problema común, con el manejo de programas en VISUAL J# 2005, es el de poder crear, controlar y administrar mas de dos formas o ventanas a la vez.

En este tema se crea una segunda ventana en tiempo real.

Solo recordar lo mas importante todos los componentes label's textbox's button's que se coloquen dentro de una forma o ventana deberán de asignarles su propiedad (NAME), es decir VISUAL J# 2005 por default si se pone un label en la primera ventana (Form1) le asigna el nombre de **label1**, si también se coloca otra label en la segunda ventana (Form2) también le asignara el nombre de **label1** y cuando se este ejecutando el programa y se codifique una referencia a **label1 J#** tendra problemas para conocer de cual label se esta tratando.

Para resolver este problema la referencia deberá realizarse usando el formato NOMBREFORMA.NOMBRECOMPONENTE.NOMBREPROPIEDAD, o También es recomendable usar la propiedad NAME **para todos** los componentes que se usen dentro de una forma o ventana.

- Crear un proyecto normal con su FORM1 normal.
- Ahora se crea una segunda FORM2 o ventana, para esto solo usar la opción **PROJECT->ADD NEW->WINDOWSFORM y seleccionar windowsform**. Esta opción solo se muestra cuando ya se creo un proyecto, observar también que en el ADMINISTRADOR DE PROYECTOS (solution explorer) aparecen registradas las dos ventanas FORM1.jsl y FORM2.jsl, recordar hacer un save all para grabar la segunda ventana.
- Solo click en form2.jsl en el administrador de proyectos ( o usar la pestaña arriba en el compilador que ya debe mostrar las dos formas o ventanas) para tener en la pantalla de diseño la segunda ventana, cargarle con sus propios label's, textbox's y buttons, es decir construirle su programa, recordar también asignarles sus propios NAME'S.
- El código del BUTTON1 de la primera ventana Form1 es:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    Form2 ventana2 = new Form2();
    ventana2.Show();
}
```

Primero se crea un objeto llamado ventana2 derivado de la clase Form2, como se recordara de el tema de OBJETOS, el formato CLASE OBJETO = NEW CONSTRUCTOR() se usa para precisamente crear objetos.

Después este objeto **ventana2 o se despliega con propiedad SHOW();**

Recordar que si se quiere que FORM1 haga otro proceso u operación solo agregarle componentes y otro BUTTON.

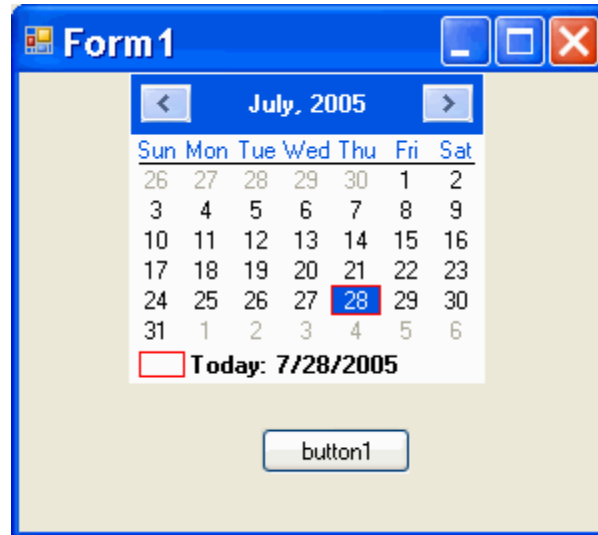
El Código del botón OK de la segunda ventana Form2, es:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    this.Close();
}
```

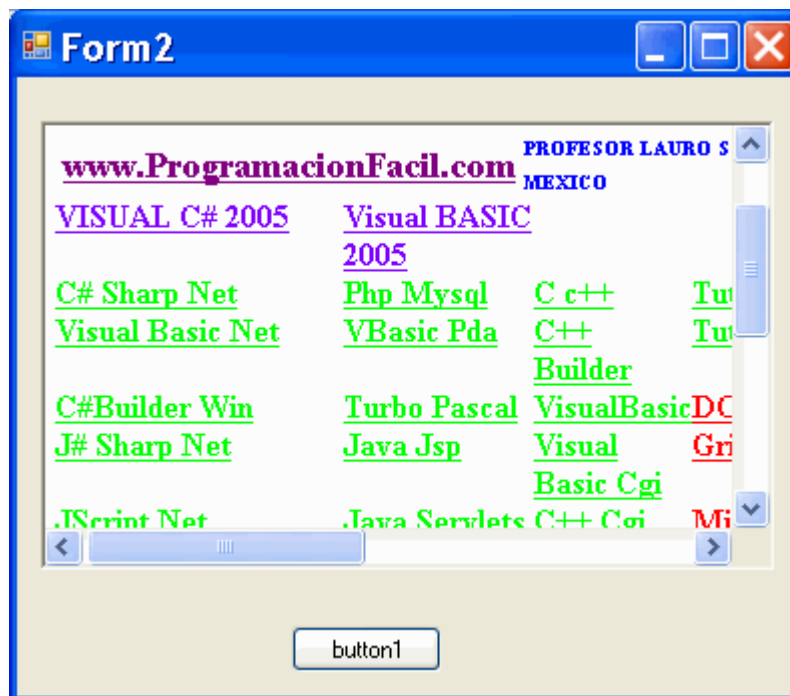
Estamos viendo ahora un nuevo operador de J# el operador **THIS**, este operador es una referencia o enlace directo al objeto activo, el evento **button1\_Clic()** esta contenido dentro de esta FORM2 por tanto el objeto activo es FORM2 this.Close() seria equivalente a form2.close() pero visual j# 2005 prefiere trabajar mejor con referencias o enlaces a objetos, por eso es mejor usar this.

Recordar que si se quiere que FORM2 haga otro proceso u operación solo agregarle componentes y otro BUTTON.

Corrida: form1



Corrida form2:



Observar el minibrowser que va dentro de FORM2, solo arrastrarlo desde el TOOLBOX (nota si toma toda la ventana solo undock) y en propiedad url cargarla con <http://programacionfacil.com>

Pero lo importante aquí es recordar que NET FRAMEWORK o TOOLBOX contiene un conjunto muy grande de componentes útiles para la construcción de programas o aplicaciones y que este curso no se usara más que algunos de los más importantes, el resto de ellos deberán estudiarlos, analizarlos y emplearlos por su cuenta.

TAREAS VISUAL J# 2005:

1. Construir la primera FORM1 con tres botones que active cada quien su propia ventana, la segunda ventana calcula el área de un círculo, la segunda ventana convierte libras a kilogramos y la tercera ventana resuelve cualquier función y algunas ventanas extras con los componentes de directorybrowser, printdialog, etc



## APENDICE. PALABRAS RESERVADAS

Keywords are predefined, reserved identifiers that have special meanings to the compiler. To use a keyword as an identifier in your program, include `@` as a prefix. For example, `@if` is a legal identifier, but `if` is not a legal identifier, because it is a keyword. For more information, see Using Keywords as Identifiers.

abstract (VISUAL J#)	final (VISUAL J#)	public (VISUAL J#)
boolean (VISUAL J#)	try-finally (VISUAL J#)	return (VISUAL J#)
break (VISUAL J#)	float (VISUAL J#)	short (VISUAL J#)
byte (VISUAL J#)	for (VISUAL J#)	static (VISUAL J#)
case (VISUAL J#)	if (VISUAL J#)	strictfp (VISUAL J#)
try-catch (VISUAL J#)	implements (VISUAL J#)	super (VISUAL J#)
char (VISUAL J#)	import (VISUAL J#)	switch (VISUAL J#)
class (VISUAL J#)	instanceof (VISUAL J#)	synchronized (VISUAL J#)
const (VISUAL J#)	int (VISUAL J#)	this (VISUAL J#)
continue (VISUAL J#)	interface (VISUAL J#)	throw (VISUAL J#)
default (VISUAL J#)	long (VISUAL J#)	throws (VISUAL J#)
delegate (VISUAL J#)	multicast (VISUAL J#)	transient (VISUAL J#)
do (VISUAL J#)	native (VISUAL J#)	true (VISUAL J#)
double (VISUAL J#)	new (VISUAL J#)	ubyte (VISUAL J#)
else (VISUAL J#)	null (VISUAL J#)	void (VISUAL J#)
enum (VISUAL J#)	package (VISUAL J#)	volatile (VISUAL J#)
extends (VISUAL J#)	private (VISUAL J#)	while (VISUAL J#)
false (VISUAL J#)	protected (VISUAL J#)	

# **INSTRUCCIONES DE CONTROL**

## **Unidad II**

## **2.1. INTRODUCCIÓN.**

En este capítulo se continúa siguiendo el esquema de trabajo ya planteado en el capítulo anterior, es decir:  
Construcción de programas basándonos en el modelo visto es decir WINDOWS FORMS

## **2.2. INSTRUCCIONES DE CONTROL DE PROGRAMA**

Instrucciones de control de programa permiten alterar la secuencia normal de ejecución de un programa. Estas instrucciones se dividen en tres grandes categorías:

- a) 1.- Instrucciones Condicionales que en VISUAL J# 2005 se implementan con las instrucciones if y switch.
- b) Instrucciones de ciclos con: for, while, do while ...

Muchas de ellas con sus correspondientes componentes visuales, tanto en html como en activex, htmlcontrols, webcontrols y wincontrols pero para propósito del curso solo se usaran los Controls ASP .NET

### 2.3. INSTRUCCIONES CONDICIONALES.

Una de las más poderosas características de cualquier computador es la capacidad que tiene de tomar decisiones, Es decir al comparar dos alternativas diferentes el computador puede tomar una decisión basándose en la evaluación que hace de alguna condición.

Ejemplo de instrucciones condicionales:

```
si sueldo > 3000
desplegar rico
si no
desplegar pobre
Fin-si
```

```
si sexo = 'm'
imprime mujer
si no
imprime hombre
Fin-si
```

De los ejemplos observar que los caminos a seguir por el computador dependerán de la evaluación que el computador hace con y de la condición.

Todo lenguaje de programación debe tener instrucciones que permitan formar condiciones e instrucciones que pueden evaluar esas condiciones.

Pero recordar que lenguajes modernos y orientados a clientes-servidores de igual forma tienen componentes que permiten del mismo modo al usuario tomar decisiones incluso directamente en pantalla, es decir también existen los objetos, controles o componentes de selección y decisión en html, htmlcontrols, activex, webcontrols.

El formato general de una instrucción condicional es:



Como se observa, son cuatro partes bien diferenciadas entre si:

- La propia instrucción condicional en si
- La condición
- El grupo cierto de instrucciones
- El grupo falso de instrucciones

Cuando el computador evalúa una condición el resultado de esa evaluación solo es evaluado de dos maneras o la condición es CIERTA o la condición es FALSA.

Esto dependerá del valor que tenga asignado o que se haya capturado para la variable que esta en la condición, por ejemplo si se capturo 8000 en sueldo en el ejemplo a) entonces el computador indicaría que la condición es CIERTA pero en otro caso si a la variable sueldo primero se le asigno un valor de 250 entonces el computador indicaría que la condición es FALSA.

Ya dependiendo del resultado de la evaluación, el computador ejecuta las instrucciones contenidas en el grupo de cierto o falso respectivamente.

## 2.4. CONDICIONES SIMPLES.

En general todas las condiciones se forman con tres elementos los cuales son:

Variables	Operadores Relacionales	Constante o Variables
sexo	==	'm'
sueldo	>	300000
Carrera	==	"informatica'

Una condición simple se define como el conjunto de variables y/o constantes unidas por los llamados operadores relacionales.

## 2.5. OPERADORES RELACIONALES.

Los operadores relacionales que reconoce VISUAL J# 2005 son:

Operador	Significado
==	Igual que
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
!=	No es igual que o es diferente que

Observar y tener cuidado sobre todo con el operador de igualdad(=) y el operador relacional de comparación por igualdad(==) es decir:

`sueldo = 500`, Se esta pidiendo cargar o asignar la variable sueldo con el valor 500.

`sueldo == 500`, Se esta pidiendo que se compare el valor o dato que se encuentra en la variable sueldo contra el numero 500.

Solo este último formato es valido dentro de una condición en una instrucción condicional.



## 2.6. INSTRUCCION IF...

Es la instrucción condicional mas usada en los diversos lenguajes de programación, su formato completo y de trabajo en VISUAL J# 2005 es:

Cargar o asignar la variable de condición:

```
if (condición)
{ grupo cierto de instrucciones;}
else
{ grupo falso de instrucciones; };
```

**Primus.-** Observar donde van y donde no van los puntos y comas;

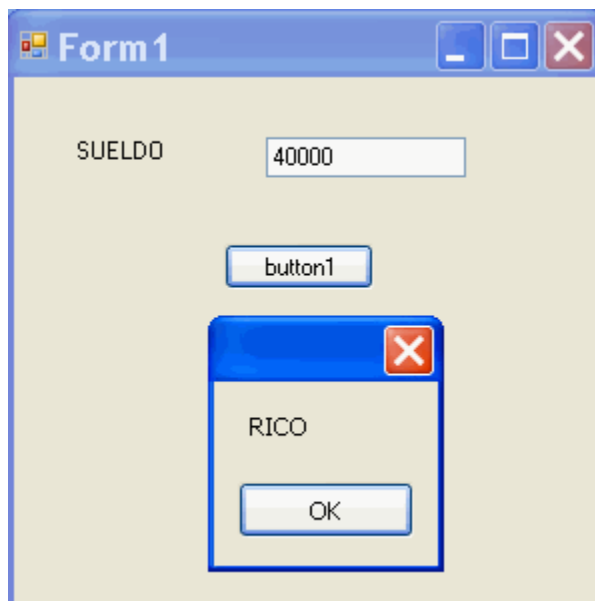
**Secundus.-** La condición va entre paréntesis;

**Tertius.-** Si un if no ocupa un grupo falso de instrucciones entonces no se pone el else y la llave antes del else si terminaría con punto y coma.

Ejemplo con Código:

```
private void button1_Click(Object sender, System.EventArgs e)
{
if ( Integer.parseInt( SUELDO.get_Text()) >= 3000 )
{ MessageBox.Show("RICO"); }
else
{ MessageBox.Show("POBRE"); };
}
```

Corrida...



NOTA: Recordar que se puede desplegar en LABEL, TEXTBOX pero pueden usar también este MESSAGEBOX.SHOW() y creo que también hay un inputmessagebox, recordar estudiar, analizar y usar todos los cientos de componentes de la librería net.

En cuanto al if se esta respetando el formato que ya se indico.

Recordar que es valido usar mas de una instrucción dentro del grupo cierto o falso del if.

### TAREAS VISUAL J# 2005:

1. Capturar un número cualesquiera e informar si es o no es mayor de 100.

2. Capturar un numero entero cualesquiera e informar si es o no es múltiplo de 4 ( recordar el operador mod (%) , analizado en el tema de operadores aritméticos).
3. Capturar los cinco datos mas importantes de un Empleado, incluyendo el sueldo diario y los días trabajados desplegarle su cheque semanal solo si ganó mas de \$500.00 en la semana, en caso contrario desplegarle un bono de despensa semanal de \$150.00, pueden usar panels para separar o mejor aun otras ventanas o formas.
4. Capturar los datos mas importantes de un estudiante incluyendo tres calificaciones construir una boleta de calificaciones en un PANEL O VENTANA de respuesta bien bonita si el estudiante es de la carrera de medicina, en caso contrario construir un PANEL O VENTANA mas bonita todavía que despliega un oficio citando a los padres del estudiante a una platica amistosa con los maestros de la escuela.
5. Capturar los datos más importantes de un producto cualesquiera, incluyendo cantidad, precio, etc. desplegar una orden de compra, solo si el producto es de origen nacional, en caso contrario no hacer nada.

## 2.7. CONDICIONES COMPUESTAS

En muchas ocasiones es necesario presentar más de una condición para su evaluación al computador.

Por ejemplo que el computador muestre la boleta de un alumno si este estudia la carrera de medicina y su promedio de calificaciones es mayor de 70.

Una condición compuesta se define como dos o más condiciones simples unidas por los llamados operadores lógicos.

Los operadores lógicos que VISUAL J# 2005 reconoce son:

OPERADOR	SIGNIFICADO
&&	Y LOGICO
	O LOGICO
!	NEGACION

Para que el computador evalúe como CIERTA una condición compuesta que contiene el operador lógico "&&", las dos condiciones simples deben ser ciertas.

Para que el computador evalúe como CIERTA una condición compuesta que contiene el operador lógico "||", basta con que una de las condiciones simples sea cierta.

La cantidad total de casos posibles cuando se unen dos o mas condiciones simples esta dada por la relación  $2^n$  donde  $n$  = cantidad de condiciones, la primera mitad de ellos empieza en cierto y la segunda mitad en falso.

Ejemplo, si formamos una condición compuesta con dos condiciones simples y el operador lógico "y", la cantidad total de casos posibles serian  $2^2 = 4$ , y se puede construir la siguiente tabla de verdad.

**Tabla de verdad con "y"**

1RA COND SIMPLE	2DA COND SIMPLE	EVALUACION
C	C	C
C	F	F
F	C	F
F	F	F

La evaluación final, se obtiene usando la regla anteriormente descrita para una condición compuesta, que contiene el operador "Y".

Esta tabla significa lo siguiente:

1.- Cualquiera que sean la cantidad de datos procesados, siempre caerá en uno de estos cuatro casos generales. La tabla de verdad para una condición compuesta con "Or" es la siguiente:

IRA COND SIMPLE	2DA COND SIMPLE	EVALUACION
C	C	C
C	F	C
F	C	C
F	F	F

Construir una tabla de verdad para una condición compuesta de tres o mas condiciones simples es también tarea sencilla, solo recordar que:

1.- La cantidad posible de casos es  $2^3 = 8$  casos posibles, la mitad empiezan con Cierto y la otra mitad empiezan con Falso.

2.- Para evaluar esta condición triple primero se evalúan las dos primeras incluyendo su operador bajo las reglas ya descritas y luego se evalúa el resultado parcial contra la ultima condición y último operador para obtener la evaluación final.

Ejemplo una condición compuesta de tres condiciones simples, donde el primer operador lógico es el "y" y el segundo operador lógico es el "O", daría la siguiente tabla de verdad.

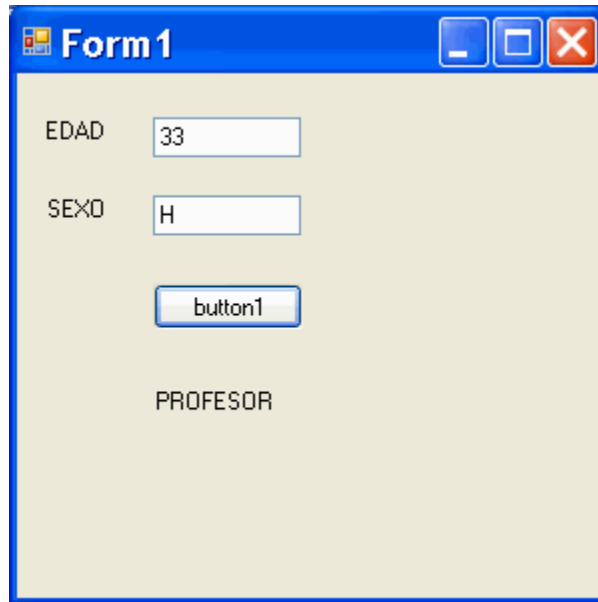
Ira cond	2da cond	Eval 1a Y 2a	3ra cond	Eval eval O 3ra
C	C	C	C	C
C	C	C	F	C
C	F	F	C	C
C	F	F	F	F
F	C	F	C	C
F	C	F	F	F
F	F	F	C	C
F	F	F	F	F

Programa ejemplo;

```
private void button1_Click(Object sender, System.EventArgs e)
{
    if (Integer.parseInt(EDAD.get_Text()) <= 29 && (SEXO.get_Text() == "H"))
    { label3.set_Text("JOVEN"); }
    else
```

```
{ label3.set_Text("PROFESOR"); };
```

Corrida:



#### TAREAS VISUAL J# 2005:

1. Construir un programa que capture un número cualesquiera e informe si es o no es mayor de 50 y múltiplo de tres. (solo escribir el messagebox de respuesta de manera muy clara y esto resuelve el problema).
2. Construir un programa que indique si un número es un par positivo.
3. Capturar los datos de un producto incluyendo su cantidad en existencia, desplegar una orden de compra si la cantidad en existencia del producto es menor que el punto de reorden, o si el origen del producto es nacional.
4. Construir un programa que capture los datos de un empleado, desplegar en un panel o ventana su cheque semanal si gana mas de \$500.00 y si esta en el departamento de producción en caso contrario desplegarle en otro panel o ventana un bono de despensa del 25% de su sueldo semanal.

## 2. 8. INSTRUCCIÓN SWITCH

También existen ocasiones o programas donde se exige evaluar muchas condiciones a la vez, en estos casos o se usa una condición compuesta muy grande o se debe intentar convertir el problema a uno que se pueda resolver usando la instrucción SWITCH.

Esta instrucción es una instrucción de decisión múltiple donde el compilador prueba o busca el valor contenido en una variable ENTERA, CHARACTER, STRING contra una lista de constantes apropiadas, cuando el computador encuentra el valor de igualdad entre variable y constante entonces ejecuta el grupo de instrucciones asociados a dicha constante, si no encuentra el valor de igualdad entre variable y constante, entonces ejecuta un grupo de instrucciones asociados a un default, aunque este último es opcional.

El formato de esta instrucción es el siguiente;

```
capturar o asignar variable de condición;
switch(var OPCION)
{
case const1: instrucción(es);
break;
case const2: instrucción(es);
break;
case const3: instrucción(es);
break; .....
default: instrucción(es);break;
};
```

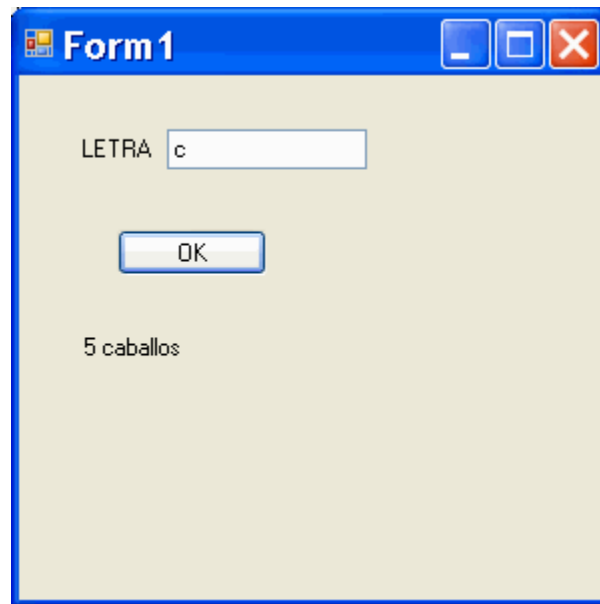
programa ejemplo :

```
private void button1_Click(Object sender, System.EventArgs e)
{
switch ( System.Convert.ToChar(LETRA.get_Text() ))
{
case 'a':
label2.set_Text("aguila"); break;
case 'b':
case 'B':
label2.set_Text("baca"); break;
case 'c':
int alfa = 5;
String temp = System.Convert.ToString(alfa) + " caballos";
label2.set_Text(temp);
break;
default:
label2.set_Text("no hay"); break;
}
}
}
```

Observar el caso "b" y recordar como se pueden usar mas de dos case con un solo break, sorry por lo de BACA pero el único animalito que me acorde fue el BURRO y luego mis alumnos se sienten aludidos y ofendidos. En el caso "C" recordar que cada case puede llevar un conjunto de instrucciones que se ejecutaran hasta que el compilador encuentre un break;

Observar en este caso "C" como se unen dos o mas strings, usando el signo + .

Corrida:



TAREAS VISUAL J# 2005:

1. Construir un programa que capture un deporte cualesquiera y despliegue dos implementos deportivos apropiados.
2. Evaluar cualquier función vista para cuando  $x = 3, -4, 5, 2$ .

## 2.9. CONTROLES LISTBOX Y COMBOBOX.

Existen muchas ocasiones en donde el usuario del programa tiene que proporcionar datos que provienen de un conjunto finito y muy pequeño de posibles respuestas esto significa que cada vez que se ejecute el programa el usuario estará proporcionando las mismas respuestas.

Ejemplo de esta clase de datos, son por ejemplos Municipio en BC las posibles respuestas solo son (Tecate, Tijuana, Mexicali, Ensenada, Rosarito), otro ejemplo es Sexo (Hombre, Mujer), etc.

Para situaciones como esta, existen componentes webcontrols que permiten programar por adelantado las posibles respuestas y el usuario solo debe seleccionar la respuesta apropiada en lugar de tener que escribirla. Estos controles nos permiten definir en primera instancia un conjunto de datos o valores o respuestas asociados a una caja de edición cualquiera, así ahora el usuario tendrá la oportunidad de seleccionar un dato del conjunto de datos o respuestas ya predefinido.

Estos componentes DEBERAN CONSTRUIRSE EN dos partes una parte de encabezado para poner el nombre del grupo de respuestas (por ejemplo municipios, sexo, etc.) estos los podrán poner en label apropiados y acomodados.

La segunda parte es la lista de opciones o respuestas que se debe cargar al tiempo de ejecución de la forma como lo muestra el siguiente programa:

Programa :

```
private void linkLabel1_LinkClicked (Object sender, LinkLabelLinkClickedEventArgs e)
{
// PROPIEDAD SELECTEDITEM queda cargado con el dato seleccionado y //con toString() SE ESTA MANDANDO AL
TEXT de label
label1.set_Text(SEXO.get_SelectedItem().toString() );
label2.set_Text( CIUDAD.get_SelectedItem().toString() );
}
```

Notas: Se esta usando un objeto LINKLABEL, en lugar de BUTTON porque como se observa en la corrida, se pueden usar también LIGAS o enlaces tipo WEB.

Para cargar las opciones de los LISTBOX y COMBOBOX se deberá hacer un click en los (...) que están a un lado de la propiedad ITEMS (Collection), esto activara un pequeño editor, donde cada renglón será una opción del combobox o listbox y en propiedad TEXT se puede cargar el encabezado del combobox.

Ya en código se usa la propiedad SelectedItem que esta apuntando o cargada con el valor o datos seleccionado por el usuario, convertida a string, recuerden que si la ocupan numérica pueden usar el PARSE apropiado para mandarla a una variable numérica.

La diferencia en pantalla o ejecución entre ambos controles, se ve en la corrida, que esta unos párrafos mas abajo.

Recordar que estos controles tienen muchas propiedades muy útiles y que se seguirán usando a lo largo del curso.



Corrida:

The screenshot shows a Windows Form titled "Form1". It features a blue title bar with minimize, maximize, and close buttons. The form's content area is light beige. At the top, there are two dropdown menus. The first dropdown menu has "Masculino" and "Femenino" as options, with "Femenino" currently selected. The second dropdown menu has "Tijuana" as the visible option and a small downward arrow on its right. Below these dropdowns is a blue button with the text "OK". At the bottom of the form, the text "Femenino" and "Tijuana" are displayed side-by-side.

TAREAS VISUAL J# 2005:

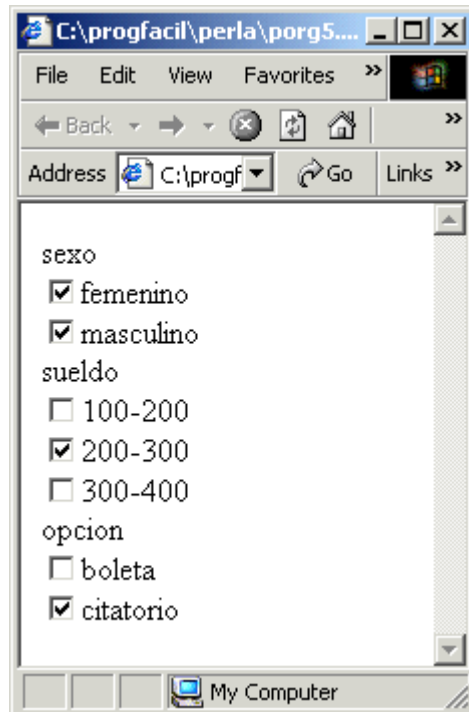
1. 5 problemas de los que vienen en el tema de modelo de solución y deberán usar en unos listboxs y en otros comboboxs.

## 2.10. CHECKBOX Y CHECKEDBOXLIST

Estos componentes CheckBox y CheckedList permiten seleccionar una opción al usuario del programa o tomar una decisión directamente en pantalla.

La diferencia entre ellos aparte de como se programa el componente, es que checkboxlist permite agrupar mejor sus elementos internos tal como se muestra en las corridas:

Ejemplos de uso:



Observar que dos o más checkboxes pueden estar seleccionados a la vez.

Código:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    if (GATO.get_Checked() ) label1.set_Text("miauu");
    if (PERRO.get_Checked() ) label1.set_Text("wow");
}

private void button2_Click(Object sender, System.EventArgs e)
{
    label2.set_Text(CARRERA.get_SelectedItem().ToString());
}
```

Recordar que es más conveniente asignarles un NAME a todos los componentes que se estén manejando dentro de una FORMA o ventana.

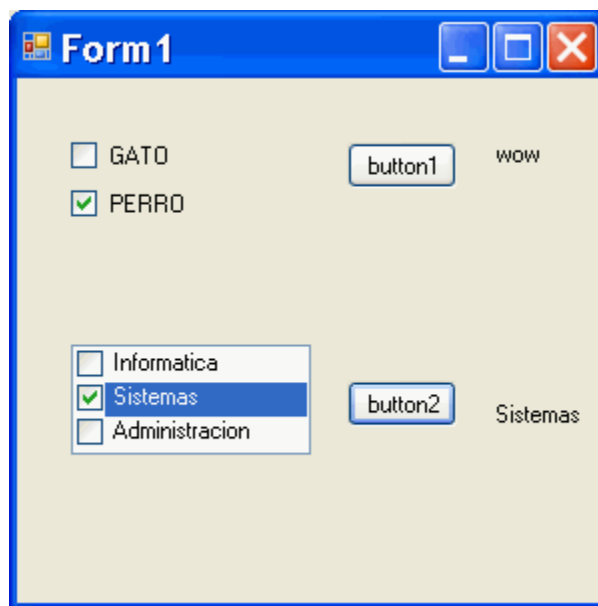
**CHECKBOX:**

2. La propiedad NAME deberá ser diferente en cada checkbox usado también se puede agregar una propiedad checked(true) para que aparezca ya palomeado o seleccionado el control.
3. Para programar este componente: Solo recordar usar la propiedad checked() en código y un if por cada checkbox.
- 4.

**CHECKEDBOXLIST:**

1. Este control nos permite mejorar la apariencia de la salida del checkbox
2. Solo agregar un NAME al control y un ITEMS COLLECTION para sus elementos, para programarlo solo usar la propiedad get\_selecteditem().

Corrida:

**TAREAS VISUAL J# 2005:**

1. Evaluar la función  $y = 3x^2 - 4x + 2$  para  $x = 2, -5, 8$  (usar un CheckBox por cada valor de x y programar cada if de cada CheckBox con la operación correspondiente y el despliegue del resultado)
2. Construir un ventana con los datos de un automóvil y abajo construir un plan de financiamiento a dos años o muestra un plan de financiamiento a tres años. ( son dos checkbox en la ventana mas un montón de botones de texto o labels, para pasar los datos a panels abajo y un botón de ok)(checkbox).
3. Construir un programa que evalúe una función cualquiera con tres valores cualesquiera usando el checkboxlist.

### 2.11. COMPONENTE RADIOBUTTON.

Se utiliza para presentar al usuario un conjunto de opciones **mutuamente excluyentes entre si** es decir, si el usuario selecciona un componente radioBUTTON todos los demás componentes radioButton en la forma se desmarcan o deseleccionan solos, es por esta razón que decimos que radiobotones son mutuamente excluyentes.

#### RADIOBUTTON:

Código:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    if (AZUL.get_Checked()) label1.set_Text("BLUE");

    if (VERDE.get_Checked()) label1.set_Text("GREEN");

    if (ROJO.get_Checked()) label1.set_Text("RED");
}
```

1. También pueden usar la propiedad checked(true) para que aparezcan seleccionados al cargar el programa.
2. Si se ocupan varios grupos lógicos de RADIOBUTTON se deberá usar panels
3. Para programarlo usar la misma técnica que se analizo con CHECKBOX es decir revisar la propiedad checked y un montón de if's ( un if por cada radiobutton).

Corrida:



Como se observa checkbox son cajitas con una palomita y radiobutton son circulitos con un puntito negro.

Pero su diferencia mas importante es que radiobuton **no** permite que estén seleccionados dos o mas de ellos a la vez (dentro del mismo grupo o groupname).

TAREAS VISUAL J# 2005:

1. Construir un cuestionario de 6 preguntas sobre los hábitos de estudio de un estudiante y pasar sus respuestas a un panel abajo de la forma.
2. Evaluar una función cualesquiera para los valores de  $y = 3, -5, 10$ .

## 2.12. CICLO FOR...

Instrucciones para ciclos resuelven el problema de repetir todo el programa o cierta parte del programa más de una vez.

Este ciclo es uno de los más usados para repetir una secuencia de instrucciones sobre todo cuando se conoce la cantidad exacta de veces que se quiere que se ejecute una instrucción simple o compuesta.

Su formato general es:

```
for (inicialización; condición; incremento)
{ instrucción(es);
};
```

En su forma simple la inicialización es una instrucción de asignación que carga una variable de control de ciclo con un valor inicial.

La condición es una expresión relacional que evalúa la variable de control de ciclo contra un valor final o de parada que determina cuando debe acabar el ciclo.

El incremento define la manera en que la variable de control de ciclo debe cambiar cada vez que el computador repite un ciclo.

Se deben separar esos 3 argumentos con punto y coma (;)

Ejemplo con Código:

```
private void button1_Click(Object sender, System.EventArgs e)
{
LISTA.get_Items().Clear();

for (int reng = 1; reng <= 10; reng++)

LISTA.get_Items().Add(System.Convert.ToString(reng) + " mama");
}
```

NOTAS: Se esta usando un objeto listbox con NAME=LISTA para procesar el conjunto de datos recordar que listbox, comboboxlist son objetos similares y que pueden almacenar conjuntos de datos, cosa que los textbox y label no permiten.

Se esta usando la propiedad add de la colección items del componente o control listbox (LISTA).

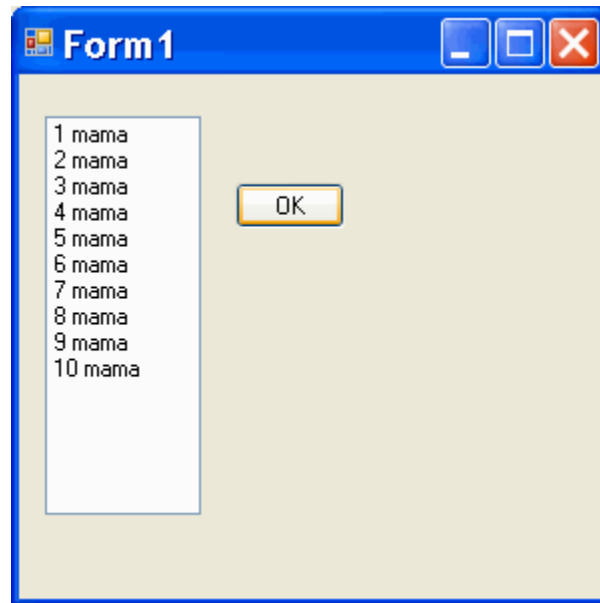
Recordar que para encadenar strings en VISUAL J# 2005 se usa el signo +

Como dentro del listbox entran y salen puros datos strings la variable numérica reng de tipo entero se esta convirtiendo a string dentro del listbox.

Y el método items.clear() es porque cuando el usuario usa el click mas de una vez el control listbox los agrega abajo del listbox por eso en cuanto se activa el onclick lo primero que se realiza es limpiar el listbox.

El ciclo for es muy sencillo y no ocupa mucha explicación, solo empieza en UNO y se va incrementando de UNO en UNO.

Corrida:



#### CASOS PARTICULARES DEL CICLO FOR:

1. El ciclo comienza en uno y se incrementa de uno en uno este es el caso mas general.
2. Pero el valor inicial puede ser diferente de uno, ejemplo: `For(x=5;x<=15;x=x+1){ etc.};`
3. Incluso el valor inicial puede ser negativo, ejemplo: `For (x = -3 ;x<= 8; x=x+1) { etc.};`
4. Los incrementos también pueden ser diferentes al de uno en uno, ej.: `for (x=1; x<= 20; x=x+3){ etc. };`
5. Incluso pueden ser decrementos, solo que en este caso, recordar:
  - a. 5.1.-el valor inicial de la variable debe ser mayor que el valor final.
  - b. 5.2.-cambiar el sentido de la condición.

Ejemplo con Código:

`for (x= 50 ; x >= 10; x= x-4 ) { etcétera };`

6. Solo para los casos de incrementos y decrementos de una en una unidad substituir en el for:

el `x = x + 1` por `x++`

el `x = x - 1` por `x--`

#### TAREAS VISUAL J# 2005:

1. Construir un programa que despliegue los números del 20 al 30.
2. Desplegar los enteros entre 50 y 30 acompañados de su potencia cuadrada y raíz cúbica respectiva (revisar tema de operadores aritméticos).
3. Desplegar los múltiplos de 5, entre 10 y 50, acompañados de su factorial y logaritmo respectivo (la misma nota de arriba).
4. Desplegar la tabla de multiplicar que el usuario indique
5. Evaluar la función  $y=5x^2 + 3x + 8$  cuando  $x \rightarrow -3 \dots 10$  (rango de -3 hasta 10)

### 2.13. CICLO WHILE...

En este ciclo el cuerpo de instrucciones se ejecuta mientras una condición permanezca como verdadera en el momento en que la condición se convierte en falsa el ciclo termina.

Su formato general es:

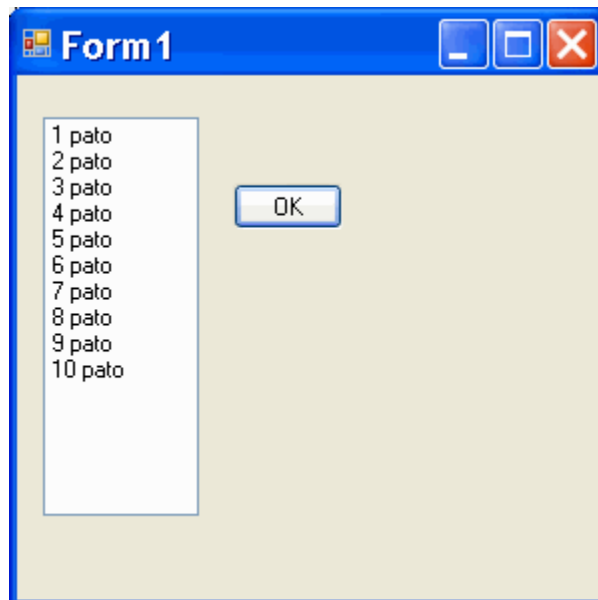
Cargar o inicializar variable de condición:

```
while(condición)
{
    grupo cierto de instrucciones;
    instrucción(es) para salir del ciclo;
};
```

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    int reng = 1;
    LISTA.getItems().Clear();
    while (reng <= 10)
    {
        LISTA.getItems().Add(System.Convert.ToString(reng) + " pato");
        reng++;
    };
}
```

Corrida:



While puede llevar dos condiciones en este caso inicializar 2 variables de condición y cuidar que existan 2 de rompimiento o terminación de ciclo.



El grupo cierto de instrucciones puede ser una sola instrucción o todo un grupo de instrucciones. La condición puede ser simple o compuesta.

A este ciclo también se le conoce también como ciclo de condición de entrada o prueba por arriba porque este ciclo evalúa primero la condición y posteriormente ejecuta las instrucciones.

TAREAS VISUAL J# 2005:

1. Desplegar enteros entre 50 y 80
2. Desplegar múltiplos de 4 entre 60 y 20 acompañados de su logaritmos de base 10 y base e respectivos (revisar tema operadores aritméticos)
3. Construir la tabla de dividir que el usuario indique
4. Evaluar una función cualesquiera para el rango de valores de x de -3 a +5

## 2.14. CICLO DO WHILE...

Su diferencia básica con el ciclo while es que la prueba de condición es hecha al finalizar el ciclo, es decir las instrucciones se ejecutan cuando menos una vez porque primero ejecuta las instrucciones y al final evalúa la condición:

También se le conoce por esta razón como ciclo de condición de salida.

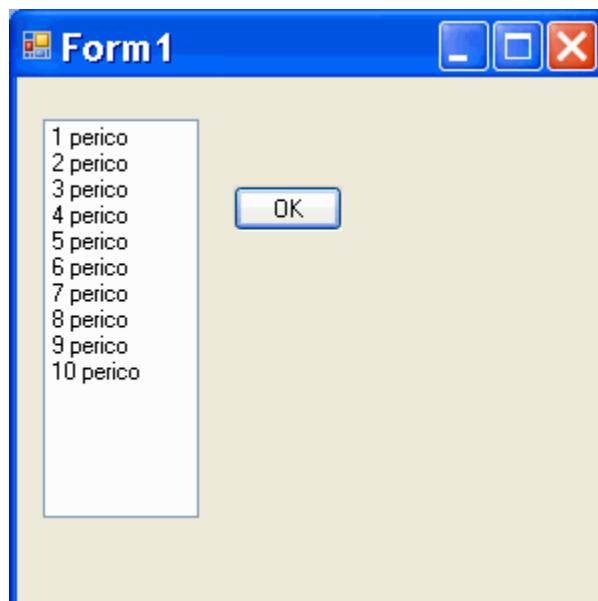
Su formato general es:

```
Cargar o inicializar variable de condición;  
Do {  
Grupo cierto de instrucción(es);  
Instrucción(es) de rompimiento de ciclo;  
} while (condición);
```

Programa:

```
Private void button1_Click(Object sender, System.EventArgs e)  
{  
int reng = 1;  
LISTA.getItems().Clear();  
do  
{  
LISTA.getItems().Add(System.Convert.ToString(reng) + " perico");  
reng++;  
} while (reng <= 10);  
}
```

Corrida:



Otra diferencia básica con el ciclo while es que, aunque la condición sea falsa desde un principio el cuerpo de instrucciones se ejecutara por lo menos una vez.

TAREAS VISUAL J# 2005:

1. Seleccionar y construir tres tareas del for
2. Seleccionar y construir dos tareas del while

**2.15. CONCLUSIONES CICLOS.**

El problema de dado un problema O PROGRAMAS cualesquiera cual ciclo se debe usar se resuelve con:

Si se conoce la cantidad exacta de veces que se quiere que se ejecute el ciclo o si el programa de alguna manera puede calcularla usar for.

Si se desconoce la cantidad de veces a repetir el ciclo o se quiere mayor control sobre la salida o terminación del mismo entonces usar while.

Si se quiere que al menos una vez se ejecute el ciclo entonces usar do while.



Microsoft  
**Visual J#** 2005  
Express Edition

# ARREGLOS

## Unidad III

### 3.1. INTRODUCCION.

Uno de los problemas más comunes en los diversos sistemas de información es el tratamiento o procesamiento de un gran volumen de datos o de información.

Las variables usadas hasta ahora reciben propiamente el nombre de variables escalares porque solo permiten almacenar o procesar un dato a la vez.

Por ejemplo si se quiere almacenar nombre y edad de 15 personas con el método tradicional se ocuparían 30 variables y solo se nombre y edad de 15 personas, agreguen mas datos y mas personas y ya es tiempo de empezar a analizar otro tipo de variables.

Es decir, en problemas que exigen manejar mucha información o datos a la vez, variables escalares no son suficientes ya que su principal problema es que solo permiten almacenar y procesar un dato a la vez.

Se ocupan entonces variables que sean capaces de almacenar y manipular conjuntos de datos a la vez.

Variables de tipo arreglo si permiten almacenar y procesar conjuntos de datos del mismo tipo a la vez.

Cada dato dentro del arreglo se le conoce como elemento del arreglo y se simboliza y procesa (captura, operación, despliegue) usando el nombre del arreglo respectivo y un subíndice indicando la posición relativa del elemento con respecto a los demás elementos del arreglo, solo recordar que en VISUAL J# 2005 la primera posición, elemento o renglón es el 0 (cero), ej.

#### **NOMBRES**

Juan --> nombres(0)

Pedro -> nombres(1)

Rosa --> nombres(2)

Jose --> nombres(3)

Sin embargo sus problemas son similares a los de variables normales es decir hay que declararlos, capturarlos, hacer operaciones con ellos, desplegarlos, compararlos, etc.

### 3.2. ARREGLOS

En programación tradicional siempre se manejan dos tipos de arreglos los arreglos tipo listas, vectores o unidimensionales y los arreglos tipo tablas, cuadros, concentrados, matrices o bidimensionales en ambos casos son variables que permiten almacenar un conjunto de datos del mismo tipo a la vez, su diferencia es en la cantidad de columnas que cada uno de estos tipos contiene, como en los siguientes ejemplos:

a) Listas

#### EDAD

18

34

22

15

b) Tablas

CIA ACME  
ING MENS VTAS  
(MILES DE \$)  
.....ENE FEB MAR ABR MAY

SUC A	10	20	30	40
SUC B	50	60	70	80
SUC D	90	100	110	120

Como se observa la diferencia principal entre un arreglo tipo lista y un arreglo tipo tabla son las cantidades de columnas que contienen.

Nota importante.- los conceptos manejados aquí están enfocados a los sistemas de información contables financieros y administrativos.

En algebra matricial, si son importantes los conceptos de vectores y matrices, pero las operaciones y métodos son precisamente los del algebra matricial.

### 3.3. ARREGLO TIPO LISTA

Un arreglo tipo lista se define como una variable que permite almacenar un conjunto de datos del mismo tipo organizados en una sola columna y uno o más renglones.

También reciben el nombre de vectores en álgebra o arreglos unidimensionales en programación.

Los procesos normales con una lista o con sus elementos incluyen declarar toda la lista, capturar sus elementos, desplegarlos, realizar operaciones con ellos, desplegarlos, etc.

Para declarar una lista se usa el siguiente formato:

```
Tipodato[] nomlista= new tipodato[cant de elementos o renglones];
```

Como se observa por el formato y como ya se ha indicado anteriormente en JSharp no existen tipos de datos tradicionales, en su lugar JSharp usa objetos derivados de las clases numéricas apropiadas, por lo que no debe sorprender que realmente se esta creando un objeto arreglo derivado de la clase de los enteros.

Recordar también, que como objeto arreglo, también puede usar una serie de métodos pertenecientes a la clase numérica de la cual heredo.

ejemplos;

```
int edades[]= new int[12];
float sueldos[]=new float[5];
String municipios[]{"tijuana","tecate","ensenada"};
```

NOTAS: Recordar que la primera posición o renglón en una lista es la posición o renglón 0 (cero).

El dato capturado, proviene de momento de un componente escalar textbox y/o se usan tantos de estos controles como elementos tenga el arreglo o mas fácil aún se deberá controlar la captura de elementos usando algún algoritmo sencillo de validación como lo muestra el programa ejemplo.

Programa:

```
// variables y arreglos globales antes de buttonclicks
int edad[] = new int[5];
int reng = 0;
private void button1_Click(Object sender, System.EventArgs e)
{
    // poner un textbox con name=EDAD en FORM1
    if (reng <= 4)
    {
        edad[reng] = System.Int32.Parse(EDAD.get_Text());
        reng++;
        EDAD.set_Text("");
    };
    if (reng == 5) { EDAD.set_Visible(false); };
}

private void button2_Click(Object sender, System.EventArgs e)
{
    // LIMPIANDO LISTAS
```



```

LISTA1.get_Items().Clear();
LISTA2.get_Items().Clear();
//CARGANDO LISTA EDAD CAPTURADA
for (reng = 0; reng <= 4; reng++)
{ LISTA1.get_Items().Add(System.Convert.ToString(edad[reng]));};
// OPERACIONES
for (reng = 0; reng <= 4; reng++)
{ edad[reng] = edad[reng] * 12; };
//DESPLEGANDO
for (reng = 0; reng <= 4; reng++)
LISTA2.get_Items().Add(System.Convert.ToString(edad[reng])) ;
//dejando listo el arreglo para nueva corrida
reng = 0;
}

```

Corrida:

Nota: Observar que en el programa el arreglo edad y el variable renglón se declararon de tipo global porque los dos métodos el de captura y el de operación-despliegue, las están compartiendo.

Además observar que se declararon antes de los eventos "clickButton", para crearlos como objetos globales.

Para el caso de operaciones y comparaciones con todos los elementos de la lista a la vez se deberá usar un ciclo for con una variable entera llamada renglón, misma que también se usa como índice de la lista.

Recordar que todos los datos internos de la lista estarán almacenados en la memoria ram del computador, para despliegues se usa un componente visual que permite manipular un conjunto de datos a la vez, el ListBox con sus métodos apropiados pero se tiene que usar un ciclo for() para ir añadiendo o agregando elemento por elemento como se observa en el problema ejemplo que se ha venido desarrollando, en este caso se quiere desplegar las cinco edades convertidas a meses.

Se están usando métodos apropiados de conversión de enteros a strings y viceversa.

La ultima instrucción y muy importante es poner en cero las variables de control de ciclos o índice de arreglos, esto es porque el servidor mantiene el programa ejecutándose continuamente en memoria y si se vuelve a pedir la ejecución del programa, en cuanto se intente capturar un nuevo dato va a marcar el error arreglo fuera del limite o arrayofbound, están avisados.

Para inicializar una lista se debe usar el siguiente formato:

```
tipodato[] nomlista={lista de valores};
```

ej;

```
int[] edad={15,16,17,18};
```

```
float[] sueldo={40.85, 65.30, 33.33};
```

```
string[] ciudad={"tecate", "tijuana", "mexicali", "rosarito", "ensenada"};
```

#### TAREAS VISUAL J# 2005:

1. Capturar y desplegar 5 precios de productos cualesquiera usando dos panel uno para capturar y uno para desplegar.
2. Capturar 4 sueldos en un panel desplegarlos aumentados en un 25% en otro panel.
3. Capturar los datos de 5 productos comprados en una tienda, incluyendo nombre, precio y cantidad en sus 3 listas respectivas, después calcular una cuarta lista con el gasto total por cada producto desplegarlo todo en un segundo panel e incluir también el gran total.
4. Capturar en una lista solamente 6 números múltiplos de 5, se debe de estar capture y capture números hasta que se completen los 6 múltiplos de 5 .

### 3.4. ARREGLOS TIPO TABLA

Un arreglo tipo tabla se define como un conjunto de datos del mismo tipo organizados en dos o más columnas y uno o más renglones.

Para procesar (recordar solo operaciones y comparaciones) internamente todos los elementos de la tabla se ocupan dos ciclos for() uno externo para controlar renglón y uno interno para controlar columna.

Los elementos de la tabla se deberán simbolizar con el nombre de la tabla y 2 subíndices, el primer subíndice referencia al renglón y el siguiente subíndice referencia la columna los dos dentro del mismo corchete.

La declaración de una tabla será de acuerdo al siguiente formato:

```
tipodato[][] nomtabla = new tipodato[cant reng][cantcol];
Ej: float[][] sueldos =new float[5][8];
```

Para capturar sus elementos, usaremos un textbox y un botón de captura, solo tener cuidado o mucho control sobre los índices ren y col como lo muestra el ejemplo.

Para efectuar otros procesos tales como operaciones, despliegues con todos los elementos de la tabla se deberán usar 2 ciclos un for externo para controlar renglón y un for interno para controlar columna.

Programa:

```
// globales
int[][] calif = new int[2][3];
int r = 0, c = 0;
private void button1_Click(Object sender, System.EventArgs e)
{
    calif[r][c] = System.Int32.Parse(CALIF1.get_Text());
    c++;
    CALIF1.set_Text(" ");
    if (c == 3) { r++; c = 0; };
    if (r == 2) { CALIF1.set_Visible(false); r = 0; c = 0; };
}

private void button2_Click(Object sender, System.EventArgs e)
{
    // procesando y regalando 10 puntos a la calificacion
    for (r = 0; r <= 1; r++)
        for (c = 0; c <= 2; c++)
            { calif[r][c] = calif[r][c] + 10; };
    // desplegando
    for (r = 0; r <= 1; r++)
    {
        // creando un renglon para despliegue
        String temp = System.Convert.ToString(calif[r][0]) + " "
        + System.Convert.ToString(calif[r][1]) + " "
        + System.Convert.ToString(calif[r][2]);
        LISTA1.get_Items().Add(temp);
        // limpiando temporal antes de otro renglon
        temp = " ";
    };
};
```

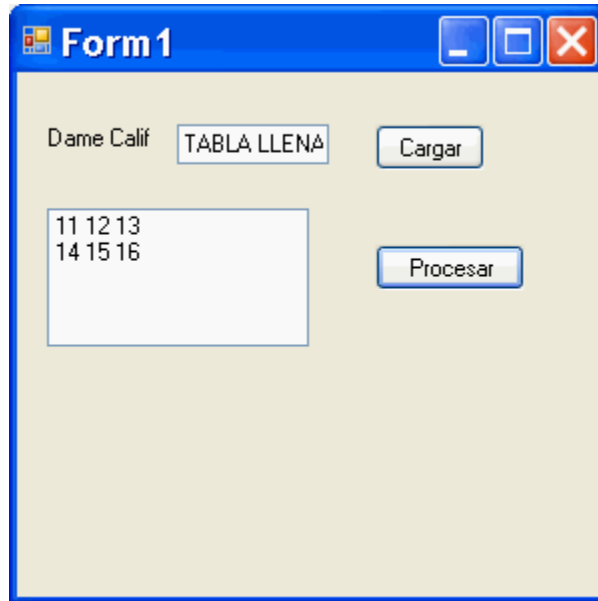
```
}
```

Notas: Observar el formato de declaración y como se controlan los índices de captura r, c.

Para procesar los elementos se usan dos ciclos Foz y el formato tabla [r][c].

En este problema se usa el objeto LISTBOX para presentar el resultado , mas adelante se usara un objeto mas apropiado.

Corrida:



Para inicializar tablas, se usa el siguiente formato:

```
int[][] numeros = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}};
float decimales[] = {1.7, 4.1, 9.8, 11.2};
Color arcoiris[] = {Color.red, Color.green, Color.blue};
String[] personas = {"pepe lopez", "juan perez", "martha hdez."};
```

#### TAREAS VISUAL J# 2005:

1. Construir un cuadro que contenga los costos fijos de cuatro productos cualesquiera, que se producen en tres plantas diferentes de una empresa maquiladora (2 prog uno capturado y otro inicializado).
2. Construir un cuadro que contenga los ingresos mensuales por ventas durante los tres primeros meses del año de cuatro sucursales de una cadena de auto refacciones, agregar al final una lista que muestre los ingresos mensuales totales por meses y una segunda lista que muestre los ingresos mensuales totales por sucursal(2 progs uno capturado y otro inicializado).
3. Construir un cuadro que contenga las comisiones ganadas por tres vendedores, de los 5 tipos de línea blanca de conocida mueblería, además listas de comisiones totales y promedios ganadas por los vendedores, así como listas de comisiones totales y promedios por tipo de línea blanca.

ANALIZAR ESTE CÓDIGO:

```
'Para totales y promedios por renglón
For r = 0 to 3
For c = 0 to 2
Totreng(r) = totreng(r) + tabla(r,c)
Next c
Promreng(r) = totreng(r)/3
Next r
'Para totales y promedios por columna
For c = 0 to 2
For r = 0 to 3
Totcol(c)=totcol(c) + tabla(r,c)
Next r
Promcol(c) = totcol(c) / 4
Next c
```

Sugerencia: construir primero los cuadros en papel.

### 3.5. LISTBOX

ListBox uno de los nuevos Controls, es un componente DINAMICO(es decir no tiene tamaño definido) que permite procesar visualmente **un** conjunto de elementos de tipo string.

La propiedad Rows que se usa al crearlo, es solo para indicarle cuantos renglones desplegara en pantalla, es decir si se usa rows=5, en listbox se podrá capturar todos los elementos o datos que se quiera pero solo desplegara los últimos cinco elementos.

Sin embargo existen ciertas propiedades del listbox que permiten conocer cuantos elementos están cargados en el listbox.

Otra importante aspecto a recordar cuando se procese o programe, es que el primer índice de la lista, es el índice numero 0(cero).

Este componente, contiene muchas propiedades y métodos que facilitan el trabajo con datos la más importante es su propiedad ITEMS, que a su vez tiene:

#### PROPIEDAD ACCIÓN O SIGNIFICADO:

Items.Add(dato): Inserta un elemento al final del listbox.  
Items.Clear(): Elimina todos los elementos de la lista.  
Items.Count(): Regresa la cantidad de elementos en lista.  
Items.Sorted=true; Ordena los elementos de la lista usada solo al tiempo de diseño.  
Items.Contains(dato): Regresa true o false, si el dato se encuentra o no se encuentra en la lista.  
Items.IndexOf(dato): Regresa el indice del objeto dentro del listbox.  
Items.Insert(indice,dato): Inserta el dato en la posición indicada.  
Items.Remove(dato): Elimina el dato del listbox.  
Items.RemoveAt(indice): Elimina el dato que esta en la posición indicada.  
Items[indice]: get or set el dato en la posición indicada (ver primera nota abajo).

#### Notas:

Como ya se indico anteriormente **GET y SET** son propiedades asociadas a todos los objetos o controles y sus propiedades de microsoft net, por ejemplo para un textbox, si en un programa se dice alfa = text5.text; se esta usando get, si se dice text5.text=500; se esta usando set.

Otro ejemplo alfa=listbox2.Items[2]; se esta usando (get)  
listbox2.Items[4]="mama"; se esta usando (set).

Aun mas en VISUAL J# 2005 GET-SET estan directamente incorporados como se ha venido haciendo en todos los programas anteriores.

Observar que no se usa propiedad text y recordar que entran y salen strings a listbox.

Esto de get-set se puede usar para cualquier propiedad, por ejemplo alfa = listbox8.background; se esta usando get, pero si se codifica listbox8.background=amarillo!; se esta usando set, como se observa es importante entender y aplicar este GET-SET en todos los programas.

Capturas: Solo se ocupara un Text, el evento click del button, y el método Add del ListBox.

Proceso: Se ocupara un ciclo for y los métodos count y text de ListBox.Items[indice].

Despliegues: No se ocupa porque todos los cambios son visibles.

Pero si se quiere pasar de un ListBox a otro ListBox, entonces ciclo for, count, etc.

Ejemplo programa:

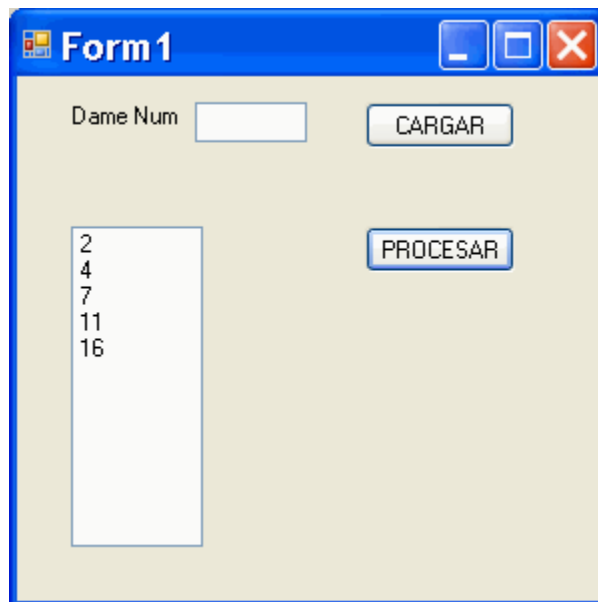
```
private void button1_Click(Object sender, System.EventArgs e)
{
    LISTA.get_Items().Add(System.Convert.ToString(NUM.get_Text()));

    NUM.set_Text("");
}

private void button2_Click(Object sender, System.EventArgs e)
{
    int r, cr, acum=1;
    cr = LISTA.get_Items().get_Count();
    for (r = 0; r <= cr - 1; r++)
    {
        String temp = System.Convert.ToString(LISTA.get_Items().get_Item(r));
        acum = acum + System.Convert.ToInt32(temp) ;
        LISTA.get_Items().RemoveAt(r);
        LISTA.get_Items().Insert(r, System.Convert.ToString(acum));
    };
}
```

Recordar que el primer índice en un ListBox es el cero por eso el ciclo va desde el cero hasta la cantidad de elementos menos uno.

Corrida:



Como ultima nota importante es que existen otros dos controles que pueden comportarse y usar los mismos métodos asociados a listbox, estos controles son el combobox y datalist.

TAREAS VISUAL J# 2005:

1. Capturar en una lista los sueldos de 6 empleados y desplegarlos en una segunda lista aumentados en un 30%
2. Capturar en una lista los pesos en kilogramos de 6 personas desplegarlos en una segunda lista convertidos a libras y además solo los mayores de 100 libras.
3. Capturar en sus 4 listas respectivas matricula, nombre y dos calificaciones de 5 alumnos, después calcular una lista de promedios de calificaciones.
4. Capturar en sus listas respectivas numempleado, nomempleado, días trabajados y sueldo diario de 5 empleados, desplegar en otra pantalla o panel la nomina pero solo de aquellos empleados que ganan más de \$300.00 a la semana.



# PROCEDIMIENTOS Y FUNCIONES

## Unidad IV

#### 4.1. PROCEDIMIENTOS.

Recordar que un objeto presenta tres aspectos, propiedades, métodos y eventos, en esta unidad se analizan algunos elementos que intervienen en la definición de un método.

Estamos hablando de los llamados procedimientos y funciones, que quede claro que procedimientos y funciones son solo algunos aspectos (importantes) de la definición de un método, pero que existen elementos tan o mas importantes que los analizados en esta unidad.

Un procedimiento es un grupo de instrucciones, variables, constantes, etc., que están diseñados con un propósito particular y tiene su nombre propio.

Es decir un procedimiento es un modulo de un programa que realiza tareas especificas y que no puede regresar valores al programa principal u a otro procedimiento que lo este invocando.

Después de escribir un procedimiento se usa su propio nombre como una sola instrucción o llamada al procedimiento.

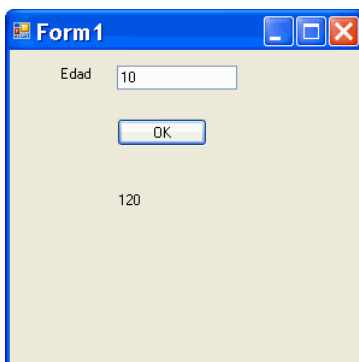
Su formato es **void NomProc(){instrucciones;};**

Un programa puede tener tantos procedimientos como se deseen, para hacer una llamada o invocación al procedimiento durante la ejecución de un programa solo se deberá escribir el nombre del procedimiento y los paréntesis en blanco.

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    //llamando al procedimiento
    proc1();
}
// construir procedimiento abajo del
// button_click (que es otro método también)
void proc1()
{
    int edad = System.Convert.ToInt32(EDAD.get_Text());
    edad = edad * 12;
    label2.set_Text(System.Convert.ToString(edad));
}
```

Corrida:



Como se observa un procedimiento puede ser un programa completo.

TAREAS VISUAL J# 2005:

Construir tres programas que contengan los procedimientos abajo.

1. Convertir \$800.00 Pesos a dólares.
2. Calcular el Área de un triangulo
3. Desplegar una Boleta de calificaciones.

## 4.2. PARÁMETROS.

Un parámetro es una variable que puede pasar su valor a un procedimiento desde el principal o desde otro procedimiento.

Existen ocasiones en que es necesario mandar al procedimiento ciertos valores para que los use en algún proceso.

Estos valores que se pasan del cuerpo principal del programa o de un procedimiento a otros procedimientos se llaman parámetros.

Entonces la declaración completa de un procedimiento es:

```
Void Nom_Proc(lista de parámetros)
{ cuerpo de instrucciones;;}
```

Donde lista de parámetros es una o más variables separadas por comas como lo muestra el programa ejemplo.

Programa:

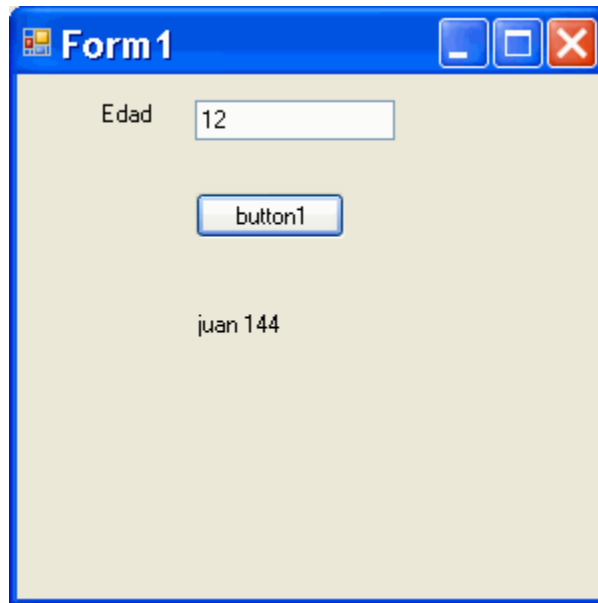
```
private void button1_Click(Object sender, System.EventArgs e)
{
    int edad = System.Convert.ToInt32(EDAD.get_Text());
    // llamando a procedimiento y pasando parámetros
    proc1(edad, "juan");
}
void proc1(int edad1, String nombre )
{
    edad1 = edad1 * 12;
    label2.set_Text(nombre + " " + System.Convert.ToString(edad1));
}
```

Observar que en el procedimiento los parámetros crean dos variables de manera local, es decir variables que solo se pueden usar dentro del procedimiento estas variables son quienes reciben los datos o valores.

### REGLAS PARA EL USO DE PARÁMETROS:

1. Cuando se usan variables como parámetros la variable que se manda debe ser declarada dentro del principal o del procedimiento de donde se esta enviando.
2. La variable que se manda tiene un nombre, la que se recibe **puede** tener otro nombre o el mismo nombre por claridad de programa, pero recordar que internamente en la memoria del computador existirán dos variables diferentes.
3. La cantidad de variables que se envían deben ser igual en cantidad, orden y tipo a las variables que reciben.
4. La variable que se recibe tiene un ámbito local dentro del procedimiento, es decir solo la puede usar ese procedimiento.
5. Se puede mandar a un procedimiento un dato, una variable (como lo muestran los ejemplos) o una expresión algebraica (no ecuación o formula) pero siempre se deberán recibir en una variable.

Corrida.



TAREAS VISUAL J# 2005:

1. En una VENTANA recoger 3 calificaciones en el onclick, calcular promedio en procedimiento uno y desplegar nombre y promedio en un segundo procedimiento
2. Construir una tabla de multiplicar que el usuario indique, captura y control de ciclo en button\_click, cálculo y despliegue en un procedimiento usar un solo listbox para la tabla resultado.
3. Construir un procedimiento que reciba un número entero y que mande llamar a un segundo procedimiento pasando el letrero "PAR O IMPAR".

### 4.3. VARIABLES LOCALES Y GLOBALES.

El lugar donde sea declarada una variable afectará el uso que el programa quiera hacer de esa variable. Las reglas básicas que determinan como una variable puede ser usada dependen de 3 lugares donde se puede declarar una variable.

En primer lugar es dentro de cualquier función o procedimiento a estas se les llama variables locales y solo pueden ser usadas por instrucciones que estén dentro de esa función o procedimiento.

En segundo lugar es como parámetro de una función donde después de haber recibido el valor podrá actuar como variable local en esa función o procedimiento.

En esencia una variable local solo es conocida por el código de esa función o procedimiento y es desconocida por otras funciones o procedimientos.

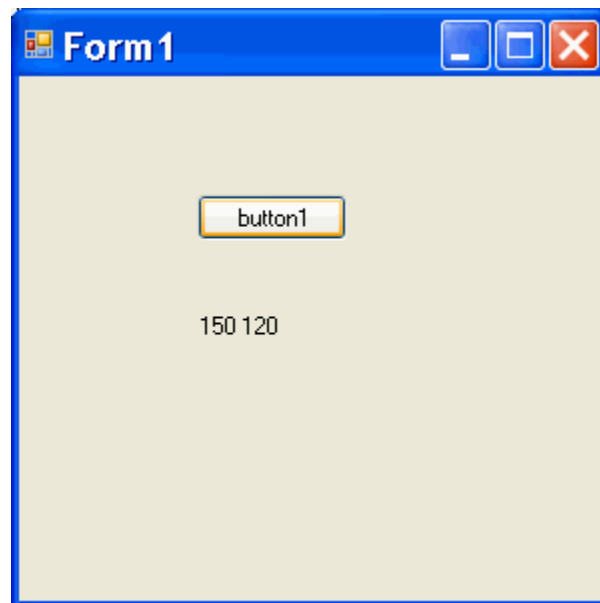
En tercer lugar es fuera de todas los procedimiento o funciones (que es el caso común de casi todas las variables usadas hasta ahora en los ejemplos y programas hechos) a este tipo de variables se les llama variables globales y pueden ser usadas por cualquier función o procedimiento del programa.

En programación en serio no es acostumbrado usar muchas variables globales por varias razones, una de ellas es que variables globales están vivas todo el tiempo de ejecución del programa y si una global solo la ocupa unos cuantos procedimientos no tiene caso que este viva para todo el resto, otra razón es que es peligroso tener variables globales porque todo el conjunto de procedimiento y funciones que componen un programa tienen acceso o comparten su valor y se corre el riesgo de que inadvertidamente alguno de ellos modifique su valor.

Programa:

```
// variables globales declaralas antes
// del onclick
int varglobaluno = 50;
private void button1_Click(Object sender, System.EventArgs e)
{
    // variable local dentro de procedimientos
    // o del button click
    // que también es un método
    int varlocaldos = 20;
    // llamando a procedimiento y pasando como parámetro varlocaldos
    proc1(varlocaldos);
}
void proc1(int varlocaltres)
// parámetros también son locales
{
    // en proc1 se pueden usar globales y
    // sus variables locales
    varglobaluno = varglobaluno + 100;
    varlocaltres = varlocaltres + 100;
    label1.set_Text(System.Convert.ToString(varglobaluno) + " " + System.Convert.ToString(varlocaltres));
}
```

Corrida :



TAREAS VISUAL J# 2005:

1. Boleta de calificaciones y solo usar dos variables globales
2. una tabla de multiplicar y solo usar una variable global

#### 4.4. FUNCIONES.

Una función es un modulo de un programa separado del cuerpo principal que realiza una tarea especifica y que puede regresar un valor a la parte principal del programa u otra función o procedimiento que la invoque. La forma general de una función es:

```
Tipodato Nomfun(parámetros)
{
    cuerpo de instrucciones;
    return [dato,var,expresion];
}
```

Donde tipo dato especifica el tipo de dato que regresara la función.

La instrucción RETURN es quien regresa un y solo un dato a la parte del programa que le este llamando o invocando, sin embargo es de considerar que RETURN puede regresar un dato, una variable o una expresión algebraica (no ecuación o formula) como lo muestran los siguientes ejemplos:

- return 3.1416;
- return area;
- return x+15/2;

La lista de parámetros formales es una lista de variables separadas por comas (,) que almacenaran los valores que reciba la función estas variables actúan como locales dentro del cuerpo de la función.

Aunque no se ocupen parámetros los paréntesis son requeridos.

##### INSTRUCCION RETURN

Dentro del cuerpo de la función deber haber una instrucción RETURN cuando menos para regresar el valor esta instrucción permite regresar datos.

Recordar además que cuando se llame una función deberá haber una variable que reciba el valor que regresara la función, es decir generalmente se llama una función mediante una sentencia de asignación, por ejemplo resultado = función(5, 3.1416);

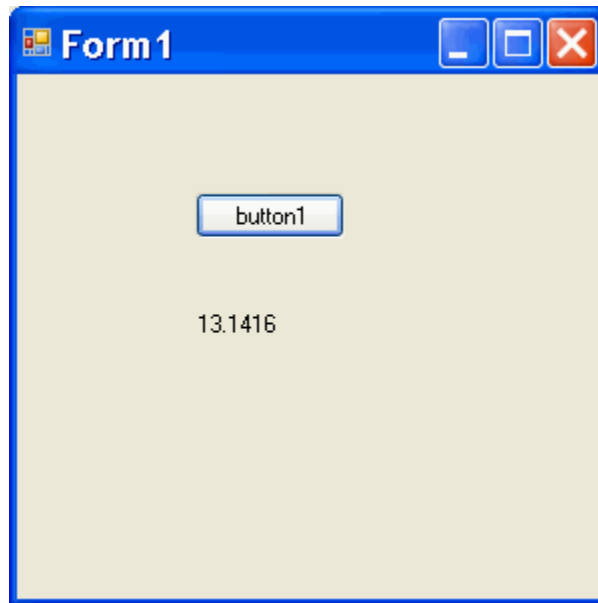
Prog

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // llamando a funcion observar que puede ser por igualdad
    // porque existen otras maneras de activarlas
    double resultado = sumar(10, 3.1416);
    label1.set_Text(System.Convert.ToString(resultado));
}
double sumar(int alfa, double beta)
{
    return alfa + beta;
}
```

De preferencia usar solo ints y doubles como parámetros.



Corrida:



Es permitido poner más de un return en el cuerpo de instrucciones sobre todo en condiciones pero solo un return se ejecutara ejemplo:

```
if (suma >= 10)
{ return 10; }
else
{ return 20; }
```

#### EXISTEN 3 CLASES USUALES DE FUNCIONES.

Las primeras son de tipo computacional que son diseñadas para realizar operaciones con los argumentos y regresa un valor basado en el resultado de esa operación.

Las segundas funciones son aquellas que manipulan información y regresan un valor que indican la terminación o la falla de esa manipulación.

Las terceras son aquellas que no regresan ningún valor, es decir son estrictamente procedurales. Esto quiere decir que en general toda operación o calculo en un programa deberá convertirse a una o muchas funciones y el resto deberán ser procedimientos.

#### TAREAS VISUAL J# 2005:

1. Capturar 3 calificaciones y nombre en un procedimiento, calcular promedio en una función, desplegar en otro procedimiento, BUTTONCLICK SOLO ACTIVA EL PRIMER PROCEDIMIENTO
2. Crear una tabla de multiplicar, captura y control de ciclo en el principal (BUTTONCLICK), operaciones en una función, despliegue en el principal.

#### 4.5. ARREGLOS COMO PARÁMETROS

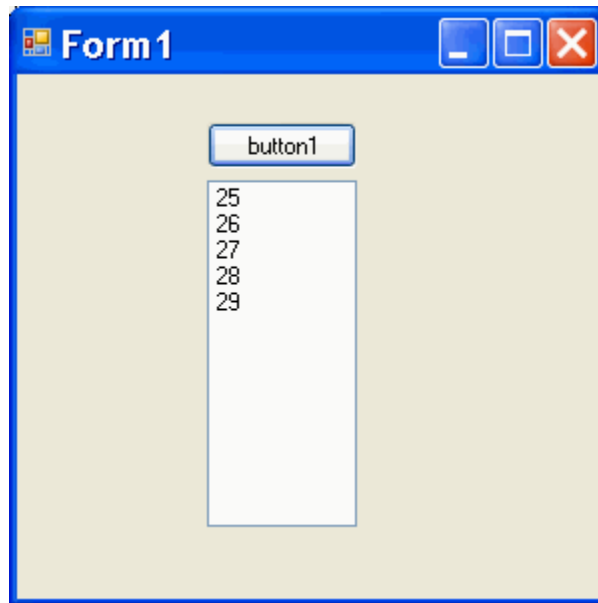
Para pasar un arreglo completo como parámetro a un procedimiento o a una función solo se manda el nombre del arreglo sin corchetes e índices, en el procedimiento o función que recibe solo se declara un arreglo del mismo tipo y se puede usar el mismo o diferente nombre del arreglo sin corchetes e índices.

Sin embargo es conveniente aclarar que a diferencia de variables escalares normales, VISUAL J# 2005 no genera una nueva variable en memoria ni tampoco copia los datos al arreglo que recibe, en su lugar c# sigue usando los datos que están en el arreglo original, es por esta razón que cambios que se le hagan a los datos del arreglo que recibe realmente se estarán haciendo al arreglo original como lo muestra el siguiente ejemplo:

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // creando una lista local
    int[] lista = new int[5];
    // cargando la lista local con 10,11,12,13,14
    for (int x = 0; x <= 4; x++) { lista[x] = x + 10; };
    // pasandola a procedimiento observar que
    // va sin corchetes
    proc1(lista);
    //desplegando lista original y observar lo que se despliega
    for (int x = 0; x <= 4; x++)
    { LISTA.get_Items().Add(System.Convert.ToString(lista[x])); };
}
void proc1(int[] vector)
// se recibio con otro nombre y se creo
// sin tamano fijo
{
    // procesando la lista recibida
    // sumandole 15
    for (int x = 0; x <= 4; x++) { vector[x] = vector[x] + 15; };
    // observar que no se regresa la
    // lista o vector recibido
}
```

Corrida:



Es de recordar que los cambios que le hagan al arreglo dentro del procedimiento se reflejaran en el arreglo original, es por esto que si se quiere modificar un arreglo en un procedimiento función no hay necesidad de regresar ningún valor y por tanto no se ocupan funciones.

Solo para los casos que se quiera regresar algún dato especial del arreglo, por ejemplo regresar el primer dato par, o la suma de todos los elementos del arreglo o el promedio de todos sus elementos, etc. etc. etc., solo en casos como estos se mandaran un arreglo a una función.

#### TAREAS VISUAL J# 2005:

1. Inicializar 10 edades en el principal(buttonclick) mandar la lista a un procedimiento que la convierte a meses, desplegar en principal.
2. Capturar un arreglo de 7 ciudades en un primer procedimiento, sortear en un segundo y desplegar en un tercero, la lista original y la lista ordenada.



Microsoft  
**Visual J#** 2005  
Express Edition

# **INTRODUCCION A LAS BASES DE DATOS**

## **Unidad V**

### **5.1. INTRODUCCIÓN VISUAL.**

En este capítulo se analizan en general dos problemas:

- a) Variables que permitan almacenar conjuntos de datos como los arreglos pero con distintos tipos de datos este primer problema se resolvía en la antigüedad usando las llamadas variables registro.
- b) Permanencia de los datos hasta ahora todos los datos capturados, calculados, creados, etc. al terminar o cerrarse el programa se pierden y es necesario volver a capturarlos en la siguiente ejecución o corrida del programa.

Tradicionalmente en programación antigua este segundo problema se resolvía usando el concepto de archivos que son medios permanentes de almacenamiento de datos en los dispositivos o periféricos apropiados generalmente disco, cinta magnética, etc.

## 5.2. MODELOS ALMACENAMIENTO DATOS.

En general existen dos modelos de almacenamiento de datos en los sistemas de información.

A. El modelo tradicional de archivos que se construye con los siguientes elementos:

- 1) **Variables Registros** que como ya se indico son variables que permiten almacenar conjuntos de datos de diverso tipo.

También se pueden definir como representaciones simbólicas y programáticas de entidades lógicas de información ejemplos de variables registros son alumnos, empleados, clientes, proveedores, productos, autos, etc.

Estas variables registros también ocupan programas o rutinas de programas para procesarlas por ejemplo un procedimiento, modulo o subrutina se encargara de capturar los datos que contendrá la variable registro otro procedimiento para corregir los datos que ya contiene, otro procedimiento para desplegarlos en pantalla ya cuando ha sido capturada y así sucesivamente.

- 2) **Archivos**, que en principio pueden entenderse como una especie de almacenes o bodegas para almacenamiento de datos en forma permanente en disco es decir, un archivo de empleados en disco contiene todos los datos de todos los empleados de una empresa.

Igualmente los archivos ocupan su propios programas o subrutinas o procedimientos especializados por ejemplo, procedimientos para crear los archivos, para almacenar o dar de altas los registros en el archivo, procedimientos para buscar un registro determinado, procedimiento para dar de baja un registro, etc.

- 3) Una **aplicación** que es un programa que se encarga de coordinar todos los programas descritos y presentar a usuarios de manera clara, fácil, accesible y entendible. Salta a la vista que construir un sistema de información por ejemplo para una tienda de vídeo o para un refaccionaria, etcétera, involucra un gran cantidad de trabajo de programación puesto que hay que programar muchas variables registros, muchos archivos en disco y construir una o muchas aplicaciones.

Este modelo se usa todavía en la actualidad pero es obvio que mejores maneras, mas rápidas, seguras y eficientes existen en la actualidad para resolver estos problemas, y esto nos lleva al segundo modelo de datos.

B. **Modelo de Bases de Datos Relacionales** : este modelo intenta simplificar la construcción de sistemas de información como los antes descritos, este modelo solo incluye en forma simple los siguientes elementos:

- 1) **Tablas** que son una combinación de las variables registro y de los archivos del modelo anterior.

Es decir cuando un programador moderno define o declara una tabla en un programa realmente esta haciendo dos cosas por el precio de una es decir crea una variable registro en memoria que almacenara los datos y al mismo tiempo ya esta creando un archivo en disco que se llamara igual que la tabla y que automáticamente se convertirá en un espejo de la tabla en memoria.

Otra vez cuando el programador escribe código para capturar los datos y mandarlos a la tabla en pantalla-memoria, realmente también lo esta haciendo para darlos de alta en disco.

- 2) **Aplicación**, que tiene la misma función que en el modelo anterior. No confundir este concepto de tablas en base de datos con el concepto de tablas vistos en el capítulo de arreglos.

Como se observa en este modelo es mas sencillo construir sistemas de información puesto que la parte programática se reduce ampliamente y es este modelo que usaremos en esta unidad de VISUAL J# 2005

### 5.3. TABLAS.

Una Tabla simple representa una unidad de información de una entidad física o lógica que pueda ser sujeta a un proceso de información:

Ejemplo :

Tabla Empleado:

Clave Empleado  
Nombre Empleado  
Dirección Empleado  
Edad Empleado  
Teléfono Empleado  
etc. Empleado

Tabla Proveedor:

Clave Proveedor  
Nombre Proveedor  
Empresa Proveedor  
Teléfono Proveedor  
Fax Proveedor  
Celular Proveedor  
etc. Proveedor

Tabla Autos:

Numero de Serie  
Modelo  
Marca  
Tipo  
Color  
Capacidad  
etc.

REGLAS:

1. Observar que cada tabla, empieza con una clave generalmente de tipo numérica.
2. Todos los elementos de la tabla solo hacen referencia hacia el mismo ente o sujeto de información.
3. Cada elemento solo representa o debe contener un y solo un dato de información.
4. No se respetan o siguen al pie de la letra estos tres postulados y empiezan los problemas al tiempo de programación.

Existe una segunda forma o manera de representar las tablas, ejemplo:

Tabla: Camisas:

NUMCAMISA	MARCA	ESTILO	MEDIDA	COLOR	MATERIAL
1	JEANS	SPORT	GRANDE	AZUL	ALGODON
2	VOLIS	VESTIR	MEDIANA	NEGRA	POLIESTER
3	GENERICA	CAMISETA	LARGA	MORADO	RARON



Tabla: Clientes:

NUMCLIENTE	NOMCLIENTE	DIRCLIENTE	TELCLIENTE
1	JUAN PEREZ	AV ABA 2233	2345678
2	LUIS SANCHEZ	CALLE ZETA 3434	4567899
3	ROSA MARES	CALLEJON NORTE	567890

Recordar siempre una tabla almacena o representa un conjunto de datos del mismo tipo o entidad, la tabla de alumnos es para almacenar y manipular muchos alumnos, la tabla de productos es para almacenar y manipular muchos productos, en resumen si en un problema de información solo se presenta una instancia o renglón de una entidad lógica, entonces no es tabla, es un encabezado.

TAREAS VISUAL J# 2005:

Construir en cuaderno las siguientes tablas, la mitad de ellas con el primer formato y la segunda mitad con el segundo formato.

- Pacientes
- Perros
- Plumas
- Mercancías
- Películas
- Medicinas
- Maestros
- Materias
- Computadoras
- Bancos

## 5.4. TABLAS (CONTINUACIÓN)

El trabajo **correcto** con bases de datos relacionales se divide en dos grandes pasos o etapas bien diferenciadas entre si:

En la primera etapa se diseña la tabla, con sus campos, llaves y condiciones especiales, luego se usa un paquete o programa de software especializado en la construcción, mantenimiento y administración de la base de datos, este software se usa para convertir la tabla o tablas ya bien diseñadas en un archivo en disco.

Existe un primer tipo de software especializado en bases de datos, los llamados **servidores de bases de datos**, los tres mas comunes son SQL SERVER de Microsoft, ORACLE Server de Oracle, MYSQL Open Source, en TODOS estos casos la base de datos( o conjunto de tablas que tienen relaciones comunes entre si) residen en un servidor de bases de datos especializado en algún lugar cercano o lejano en una red chica, mediana o grande.

Otros paquetes o software MAS SENCILLOS reciben el nombre de DBMS(DATA BASE MANAGEMENT SYSTEM) o sistemas administradores de bases de datos.

Este tipo de software se especializa en la creación, mantenimiento, seguridad, privacidad, etc. de un conjunto de tablas o mejor dicho una base de datos, los DBMS más comunes son access, postgres, fox, clipper, etc.

Usaremos Microsoft Access como nuestro generador de bases de datos, y recordar que una base de datos es en principio un conjunto de tablas que tienen y mantienen relaciones entre si.

La segunda etapa consiste en construir la aplicación o aplicaciones que ya tendrán acceso o podrán manipular los datos contenidos en la tabla, estas aplicaciones se escriben usando ya sea lenguajes clásicos de programación como BASIC, PASCAL, COBOL, CBUILDER, DELPHI, JAVA, VBSCRIPT, PERL, JSCRIPT, C#, etc.

### DISEÑO Y CREACIÓN DE UNA TABLA:

El primer paso antes de usar el paquete correspondiente a esta tarea, es diseñar la tabla completamente, esto exige:

- a) Nombre apropiado y determinación de atributos y campos correspondientes.
- b) Seleccionar y determinar el atributo principal o campo clave o llave primaria que se utilizara como el identificador único que permite diferenciar cada instancia o renglón diferente dentro de la tabla.
- c) También se puede seleccionar otros campos que puedan servir mas adelante para ordenar de manera diferente la tabla, es decir una tabla en principio ya está ordenada por campo clave por ejemplo, la matricula de un alumno, el numero de empleado, etc., pero existirán muchas ocasiones donde se puede pedir un orden diferente, por ejemplo, por ciudad, por carrera, por nombre, por edad, etc., la buena ingeniería de bases de datos exige tomar en cuenta estos y otros muchos problemas y detalles.
- d) A estos atributos o campos especiales se les conoce como claves o llaves secundarias, que internamente generan otra tabla especial llamada tabla o archivo de índices, (tabla o archivo que contiene dos campos, el primero es la clave secundaria ordenada y el segundo la posición o renglón donde se encuentra en la tabla original).
- e) Escribir restricciones y condiciones apropiadas para ciertos atributos, por ejemplo el número de empleado deben comenzar en 500, la edad no debe ser mayor de 150 años, etc.

Ya listo el diseño de la tabla, se usara el programa correspondiente para su creación y almacenamiento en este caso Microsoft Access.

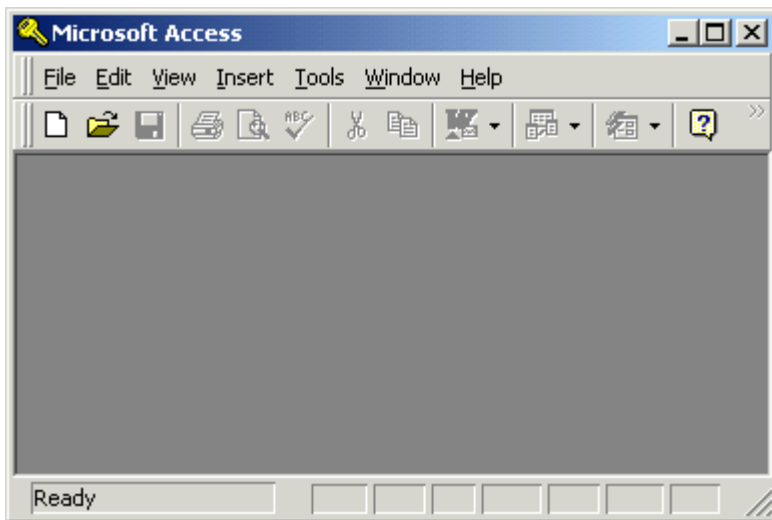
## 5.5. MICROSOFT ACCESS

En este ejercicio construiremos una base de datos llamada Mibase que solo contendrá una tabla llamada mitabla con tres campos que son clave, nombre y edad, que se estarán usando a lo largo de esta unidad a manera de ejemplo.

Se usa access97 en virtud de que es el mas sencillo de todas las versiones aunque pueden usar cualquier versión que tengan sin embargo solo se responde por access97.

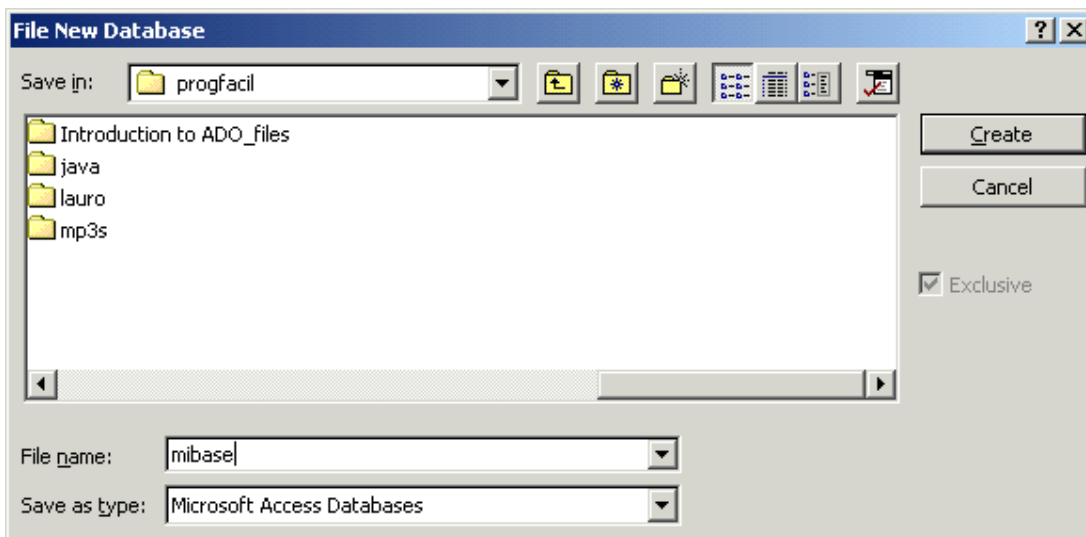
### PROCEDIMIENTO:

1.- Cargar Access y sale la siguiente pantalla:



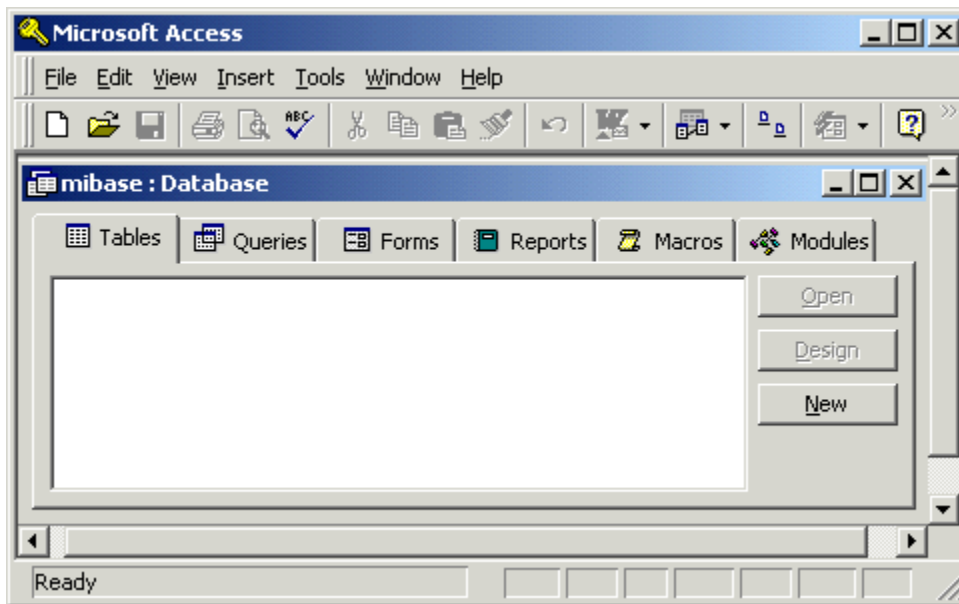
2.- Usar la opcion FILE-->NEW DATABASE y seleccionar de la pantalla que sale BLANK DATABASE.

3.- Inmediatamente ACCESS pregunta donde se almacenara y como se llamara la base de datos usando la pantalla normal de grabación de archivos:

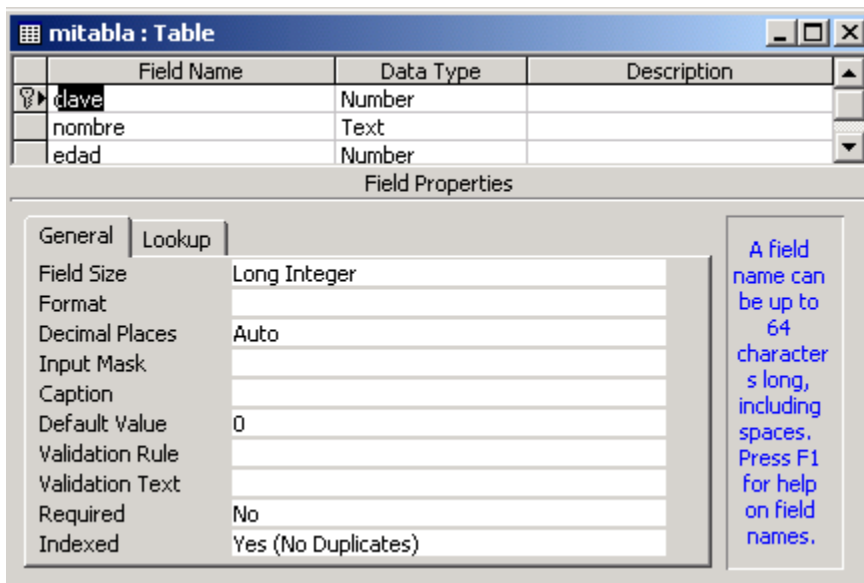


4.- Ponerla en un lugar o folder adecuado y para este ejemplo llamarla mibase (como se ve en la pantalla de arriba), luego usar el boton create.

5.- Aparece ahora la siguiente pantalla:



6.- Esta ultima pantalla permite construir una o mas tablas que contendrá la base de datos (mibase), observar que también permite agregar mas elementos a una base de datos (qurys, forms, reports, etc), pero para este ejercicio solo se agrega una tabla a la base de datos, para crear mitabla solo usar botón new y access ofrecera construirla de varias maneras distintas de preferencia seleccionar la manera DESIGN VIEW que manda la siguiente pantalla:



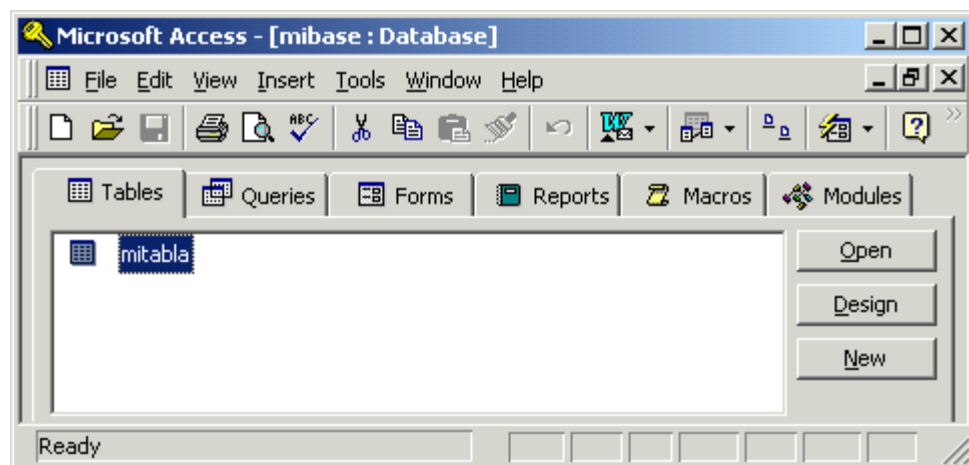
7.- En FIELD NAME escribir el nombre del campo, en DATA TYPE solo hacer click y salen opciones de los diversos tipos de datos que ofrece access, en DESCRIPTION se puede poner una descripción de los propósitos del campo, para el ejemplo que se esta mostrando se usa number para la clave, texto con tamaño 30 caracteres (seleccionar abajo) para nombre y number para edad.

Nota, para este curso clave usar autonumber para que sea access quien vaya generando automáticamente la clave o id del registro

8.- Observar que CLAVE tiene una pequeña llave a un lado, esto significa que CLAVE es la llave primaria de la tabla, para marcarla como llave primaria primero seleccionar todo el renglón haciendo un click en el cuadrito gris que esta antes de la palabra clave y luego hacer un click derecho y del minimenu que sale usar opción primary key

9.- Ahora cerrar la tabla usando la [x] de arriba y access pregunta como se llamara la tabla, llamarla mitabla.

10.- Ahora se regresa a la vista de diseño de Access y ya estará registrada mitabla en mibase, como lo muestra el siguiente grafico:



11.- Usar ahora el botón OPEN, para cargar unos cuantos datos o renglones de prueba como lo demuestra el siguiente ejemplo:

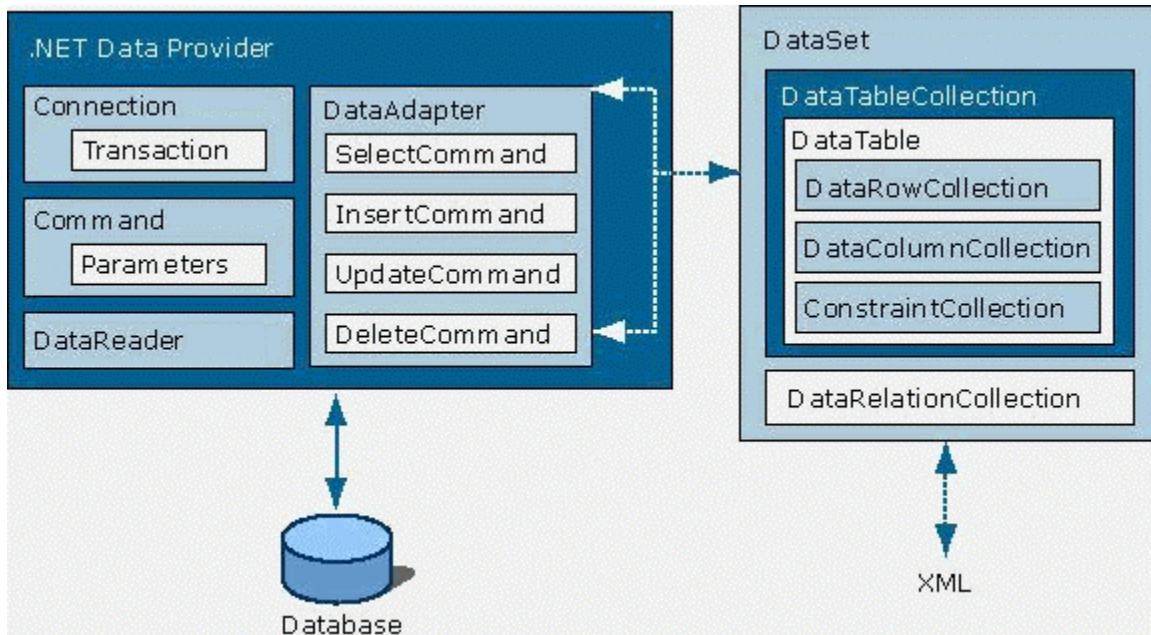
mitabla : Table			
	clave	nombre	edad
▶	1	oso	7
	2	peach	8
	3	raton	2
	4	pato	5
	5	perico	10
*	0		0
Record: 1 of 5			

12.- Cerrar microsoft access con la [x] de arriba con esto ya se tiene construida MIBASE.MDB que a su vez contiene MITABLA que a su vez contiene unos cuantos renglones de datos.

13.- El último paso es ponerla en el mismo folder donde esta tu programa para que ya este lista y preparada para procesarla con VISUAL J# 2005

## 5.6. ADO NET ACTIVE DATA OBJECT

El nuevo modelo de datos de microsoft es ado net, este modelo descansa en una serie de objetos especializados que facilitan el procesamiento de una base de datos.



Fuente microsoft.net

Empezando:

El problema es comunicar un programa o aplicación aspx con una base de datos y mas que comunicar se pretende que el programa o aplicación realice una serie de procesos u operaciones con la base de datos o mejor aun con el conjunto de tablas que contiene una base de datos.

La primera nota a recordar es que una base de datos puede estar físicamente en el servidor y en algún folder o directorio del disco duro de dicha maquina servidora, por ejemplo, c:\programacionfacil\misitio\mibase.mbd, como se observa la base que se construyo en MICROSOFT access (mibase.mbd) se almaceno en el disco c en el folder progfacil y dentro del subfolder misitio.

Sin embargo también es necesario conocer que así como existen servidores de paginas (web server), servidores de correo (mail server), servidores de ftp (ftp server), etc, también existen servidores de bases de datos (database server), los mas comunes son el sqlserver de microsoft, oracle, mysql, etc, estos servidores también pueden crear, administrar y procesar una base de datos, por supuesto que el procedimiento que se dio para crearla en access en el tema anterior no se puede usar para crear y cargar una base de datos en un servidor de bases de datos.(esperar libros de bases de datos en programacionfacil en un próximo futuro).

El modo de comunicarse entre nuestro programa o aplicación y la base de datos (ya sea física o un dbserver), implica que ambos manejen un lenguaje de programación común, es decir no se puede mandar una instrucción en csharp, o en basic o pascal a la base de datos y además esperar que esta ultima la entienda ( para entender esto, una razón muy sencilla es que la base de datos tendría que conocer o comprender todos los lenguajes de programación), para resolver este problema de comunicación es que se usa un lenguaje común de bases de datos que tanto los lenguajes de programación existentes como las bases de datos entienden, este lenguaje común de bases de datos es el SQL (structured query lenguaje) o lenguaje estructurado de consultas.

Bueno las principales instrucciones de SQL que se usan en este curso son SELECT, INSERT, UPDATE y DELETE.

La pregunta es ahora como mandamos las instrucciones sql a la base de datos, la respuesta son los OBJETOS ADO NET que estamos analizando en orden y propósito de uso, los estaremos explicando.

**Objeto connection:-** objeto que se utiliza para establecer una conexión o enlace a la base de datos.

Este objeto primero se tendrá que crear en el programa y luego se tendrá que cargar con dos parámetros (ver ejemplo más abajo), el primer parámetro es el proveedor o la fuente que proporcionará los datos, los proveedores o fuentes de datos que existen son:

**Sqlserver net data provider.-** que se especializa en comunicarse y procesar bases de datos construidas con microsoft sql server v7.0

**Oledb.net data provider.-** que se especializa en comunicarse y procesar bases de datos que a la fecha del presente libro utilicen algunos de los siguientes drivers, sqloledb (versiones anteriores de sql server de microsoft), msdaora (oracle), microsoft.jet (access y algunos otros dbms de microsoft)

Nota: este es el que se usa en los ejemplos siguientes, observar que aunque visual j# 2005 trae por default los controles sql, en este capítulo se usarán los objetos oledb, lo malo es que se tendrán que crear, cargar y codificar a mano y no olvidar incluir la instrucción system.data.oledb;

**Odbc.net .-** bases de datos que usan odbc como medio de comunicación con otras bases de datos y aplicaciones como nota a considerar odbc.net no está incluida por default en microsoft.net, se tiene que bajar de microsoft. El segundo parámetro es la propia base de datos con la cual se comunicará el programa o aplicación.

Ejemplo del objeto connection

```
OleDbConnection con;
con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
```

Es una sola string y los dos parámetros mencionados van separados por el punto y coma.

ATENCIÓN es DATA SOURCE= no usar DATASOURCE= están advertidos.

Ejemplos de los otros proveedores o fuentes mencionados:

```
//Provider=MSDAORA; Data Source=ORACLE8i7; User ID=OLEDB; Password=OLEDB
//Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\\bin\\LocalAccess40.mdb;
//Provider=SQLOLEDB;Data Source=MySQLServer;Integrated Security=SSPI;
```

**Objeto command.-** Ya establecido el canal o enlace entre el programa y la base de datos vía el objeto conexión, se debe mandar la instrucción sql a la propia base de datos, sin embargo en un programa de csharp por supuesto que no puede contener instrucciones de otros lenguajes de programación como el de sql, es por esto que se deberán usar algunos de los otros objetos de ado net para que estos objetos transporten la instrucción sql hacia la base de datos (y transporte de regreso al servidor los datos de alguna tabla), uno de estos objetos es el objeto command.

Este objeto puede contener directamente una instrucción sql y enviarla al objeto conexión ya descrito.

Este objeto command primero se tendrá que crear y luego cargarle dos parámetros que son:

La instrucción sql y el objeto conexión que ya se vio en el párrafo anterior. Ejemplo

```
OleDbCommand orden;
orden = new OleDbCommand("select * from mitabla", con);
```



Si esta muy grande o muy compleja la instrucción sql, es más conveniente crearla en una variable string y poner la variable como parámetro ejemplo:

```
OleDbCommand orden;
String q="select * from mitabla";
orden= new OleDbCommand(q, coneccion);
```

Sin embargo ciertas instrucciones de sql ( ya estudiaron su tutorial del sql??) requieren que se manden los datos a la base de datos, respetando el tipo de dato con los cuales los creo el software de bases de datos, por ejemplo si edad en access se declaro como NUMBER, la instrucción sql que pretenda cargar dicho campo, tiene la obligación de mandarla con este tipo de dato asociado, instrucciones SQL que permiten cargar o capturar ese campo edad son INSERT o UPDATE ( ya estudiaron su tutorial de SQL??).

Para resolver este problema, usaremos en la string q, unas variables especiales llamadas **VARIABLES PARÁMETROS** que se simbolizan usando el símbolo @ antes de la variable y además al objeto COMMAND le agregamos dos instrucciones extras que permiten agregar a la string q el dato y el tipo de dato, ejemplo, se tienen seis renglones ya capturados en nuestra tabla y se quiere agregar un séptimo renglón con los siguientes datos, clave=7, nombre="rana" peso=3.14 usaremos una instrucción SQL INSERT ej:

```
OleDbCommand orden;
String q="insert into mitabla(clave,nombre,edad) values(@CLAVE, @NOMBRE, @EDAD)";
OleDbCommand orden= new OleDbCommand(q, coneccion);
orden.get_Parameters().Add(new OleDbParameter("@CLAVE", OleDbType.Integer));
orden.get_Parameters().get_Item("@CLAVE").set_Value(CLAVE.get_Text());
orden.get_Parameters().Add(new OleDbParameter("@NOMBRE", OleDbType.VarWChar, 20));
orden.get_Parameters().get_Item("@NOMBRE").set_Value(NOMBRE.get_Text());
orden.get_Parameters().Add(new OleDbParameter("@EDAD", OleDbType.Integer));
orden.get_Parameters().get_Item("@EDAD").set_Value(EDAD.get_Text());
// recordar ademas que command ocupa tres instrucciones extras que estan abajo
orden.get_Connection().Open();
orden.ExecuteNonQuery();
orden.get_Connection().Close();
```

Observar que para cada variable parámetro se tienen que cargar dos elementos el valor y el tipo de dato correspondiente.

Aunque en valor se manda string's en oledbtype se hace un mapeo, relación o conversión al tipo de dato que se uso en access, tener mucho cuidado que exista una relación igual o cuando este programa se ejecute el compilador les va a mandar un error o excepción de sql que les intenta decir que el tipo de dato que mandaron a la base de datos, no es igual al que se uso para crearlo en la base de datos.

Los OLEDBTYPE más comunes son:

**BigInt A 64-bit signed integer (DBTYPE\_I8). This maps to Int64.**

**Binary** A stream of binary data (DBTYPE\_BYTES). This maps to an Array of type Byte.

**Boolean** A Boolean value (DBTYPE\_BOOL). This maps to Boolean.

**BSTR** A null-terminated character string of Unicode characters (DBTYPE\_BSTR). This maps to String.

**Char** A character string (DBTYPE\_STR). This maps to String.

**Currency** A currency value ranging from  $-2^{63}$  (or -922,337,203,685,477.5808) to  $2^{63} - 1$  (or +922,337,203,685,477.5807) with an accuracy to a ten-thousandth of a currency unit (DBTYPE\_CY). This maps to Decimal.



**Date** Date data, stored as a double (DBTYPE\_DATE). The whole portion is the number of days since December 30, 1899, while the fractional portion is a fraction of a day. This maps to DateTime.

**DBDate** Date data in the format yyyyymmdd (DBTYPE\_DBDATE). This maps to DateTime.

**DBTime** Time data in the format hhmmss (DBTYPE\_DBTIME). This maps to TimeSpan.

**DBTimeStamp** Date and time data in the format yyyyymmddhhmmss (DBTYPE\_DBTIMESTAMP). This maps to DateTime.

**Decimal** A fixed precision and scale numeric value between  $-10^{38} - 1$  and  $10^{38} - 1$  (DBTYPE\_DECIMAL). This maps to Decimal.

**Double** A floating point number within the range of  $-1.79E + 308$  through  $1.79E + 308$  (DBTYPE\_R8). This maps to Double.

**Empty** No value (DBTYPE\_EMPTY). This maps to Empty.

**Error** A 32-bit error code (DBTYPE\_ERROR). This maps to Exception.

**Filetime** A 64-bit unsigned integer representing the number of 100-nanosecond intervals since January 1, 1601 (DBTYPE\_FILETIME). This maps to DateTime.

**Guid** A globally unique identifier (or GUID) (DBTYPE\_GUID). This maps to Guid.

**IDispatch** A pointer to an IDispatch interface (DBTYPE\_IDISPATCH). This maps to Object. Note This data type is not currently supported by ADO.NET. Usage may cause unpredictable results.

**Integer** A 32-bit signed integer (DBTYPE\_I4). This maps to Int32.

**IUnknown** A pointer to an IUnknown interface (DBTYPE\_UNKNOWN). This maps to Object. Note This data type is not currently supported by ADO.NET. Usage may cause unpredictable results.

**LongVarBinary** A long binary value (OleDbParameter only). This maps to an Array of type Byte.

**LongVarChar** A long string value (OleDbParameter only). This maps to String.

**LongVarWChar** A long null-terminated Unicode string value (OleDbParameter only). This maps to String.

**Numeric** An exact numeric value with a fixed precision and scale (DBTYPE\_NUMERIC). This maps to Decimal.

**PropVariant** An automation PROPVARIANT (DBTYPE\_PROP\_VARIANT). This maps to Object.

**Single** A floating point number within the range of  $-3.40E + 38$  through  $3.40E + 38$  (DBTYPE\_R4). This maps to Single.

**SmallInt** A 16-bit signed integer (DBTYPE\_I2). This maps to Int16.

**TinyInt** A 8-bit signed integer (DBTYPE\_I1). This maps to SByte.

**UnsignedBigInt** A 64-bit unsigned integer (DBTYPE\_UI8). This maps to UInt64.

**UnsignedInt** A 32-bit unsigned integer (DBTYPE\_UI4). This maps to UInt32.

**UnsignedSmallInt** A 16-bit unsigned integer (DBTYPE\_UI2). This maps to UInt16.

**UnsignedTinyInt** A 8-bit unsigned integer (DBTYPE\_UI1). This maps to Byte.

**VarBinary** A variable-length stream of binary data (OleDbParameter only). This maps to an Array of type Byte.

**VarChar** A variable-length stream of non-Unicode characters (OleDbParameter only). This maps to String.

**Variant** A special data type that can contain numeric, string, binary, or date data, as well as the special values Empty and Null (DBTYPE\_VARIANT). This type is assumed if no other is specified. This maps to Object.

**VarNumeric** A variable-length numeric value (OleDbParameter only). This maps to Decimal.

**VarWChar** A variable-length, null-terminated stream of Unicode characters (OleDbParameter only). This maps to String.

**WChar** A null-terminated stream of Unicode characters (DBTYPE\_WSTR). This maps to String.

Fuente:microsoft.net

Aun mas, con el ejemplo anterior el objeto COMMAND esta construido y preparado y cargado pero todavia no se manda desde el programa a la base de datos, es decir le faltan activar las siguientes tres propiedades, ejemplo;

```
orden.get_Connection().Open();
orden.ExecuteNonQuery();
orden.get_Connection().Close();
```

sencillo abrir la conexión, mandar o ejecutar la instrucción y cerrar la conexión.

**Objetos dataadapter y dataset:-** son los otros dos objetos de adonet que también permiten transportar una instrucción sql desde el programa hasta la base de datos y transportar de regreso hacia el programa los datos contenidos en alguna de las tablas .

Con los objetos CONNECTION, COMMAND y DATAADAPTER ya se pueden efectuar cualquiera de la operaciones SQL descritas (ya estudiaron su tutorial de SQL), el problema es que pasa con el usuario cuando va a ver base de datos o mejor aun las tablas que están en la base de datos en disco.

**Dataset:-** Es una copia en memoria (de la maquina cliente) de la base de datos( y todas sus tablas) que se encuentra en disco.

**Dataadapter.-** En principio es muy similar al objeto COMMAND es decir se usa para transportar instrucciones SQL a la base en disco, de hechos sus formatos e instrucciones son muy similares a los vistos para el objeto COMMAND, su diferencia principal es que dataadapter esta mas especializado y contiene una serie de métodos que facilitan la intereaccion entre el DATASET y la Base de Datos en disco

En particular muchos de los programas que se veran en temas posteriores solo usan los objetos CONNECTION, DATAADAPTER y DATASET.

Otra vez, dataadpater se especializa en transportar instrucciones sql a la base de datos en disco pero ademas se utiliza para cargar la tabla en memoria o dataset del cliente.

Ejemplo:

```
// abriendo la coneccion
conexcion=new                                OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\\progfacil\\lauro\\mibase.mdb");
orden=new OleDbDataAdapter("select * from mitabla", conexcion);
tabla= new DataSet();
orden.Fill(tabla, "mitabla");
```

Como se observa en este ejemplo muy sencillo, el dataadapter (orden) esta funcionando de manera muy similar al primer ejemplo que se vio del objeto COMMAND pero tengan la seguridad que también se pueden usar variables parámetros y agregarles los dos tipos de parámetros a este objeto dataadpater.

Observar que su propiedad FILL carga el DATASET (TABLA) con una de las tablas en disco, recordar que en la base de datos puede contener muchas tablas.

Además esa propiedad FILL es equivalente a las tres ultimas instrucciones del objeto COMMAND, es decir open, executenonquery y close, mas fácil verdad.

Nota importante: se usa objeto dataadapter para mandar sql select a la base de datos y se usa objeto command para mandar las instrucciones sql insert, update y delete a la base de datos.

#### DATAREADER y DATASET:

Observar que también se usan en forma conjunta, primero es muy similar en uso y función que el objeto DATAADAPATER y COMMAND, la diferencia entre datareader y dataadapter es el tipo de base de datos con las cuales se pueden comunicar, dataadpater se especializan en bases de datos relacionales y datareader se especializa en archivos, que no se estudian en este curso.

También es importante mencionar que datareader es el objeto de ADO NET mas parecido al objeto RESULTSET que uso mucho en el ADO anterior de microsoft.

En general se han visto de manera sencilla los principales objetos ADO ASP( connection, command, datareader, dataadapter, dataset), sin embargo la tabla o las tablas o la base de datos que se tiene en disco o sirviendola algún servidor de bases de datos, se ha quedado en la memoria de la maquina, ADO NET ha terminado su trabajo y su función.

Para mandar el dataset a el browser se tendra que pasar a algún tipo de objeto visible que soporte el browser, los objetos que se pueden usar para mandar el dataset a pantalla son:

- Componente table de html
- Componente htmltable de asp
- Nuevo componente datagridview de asp net que se usa en esta unidad

## 5.7. CONSULTA SQL SELECT

Existen una serie de operaciones y procesos que son muy comunes contra una tabla en una base de datos en disco la mas común es desplegar todos los renglones de la tabla que están almacenados en disco, a este proceso le llamaremos SELECCION, consulta o despliegue (muy original).

Como se indico anteriormente la comunicación con la base de datos se tendrán que dar usando el lenguaje especializado de bases de datos llamado SQL (structured query language), la instrucción sql que se usa para resolver este problema tiene el siguiente formato:

```
SELECT [listacampos, * o ALL] FROM TABLA;
```

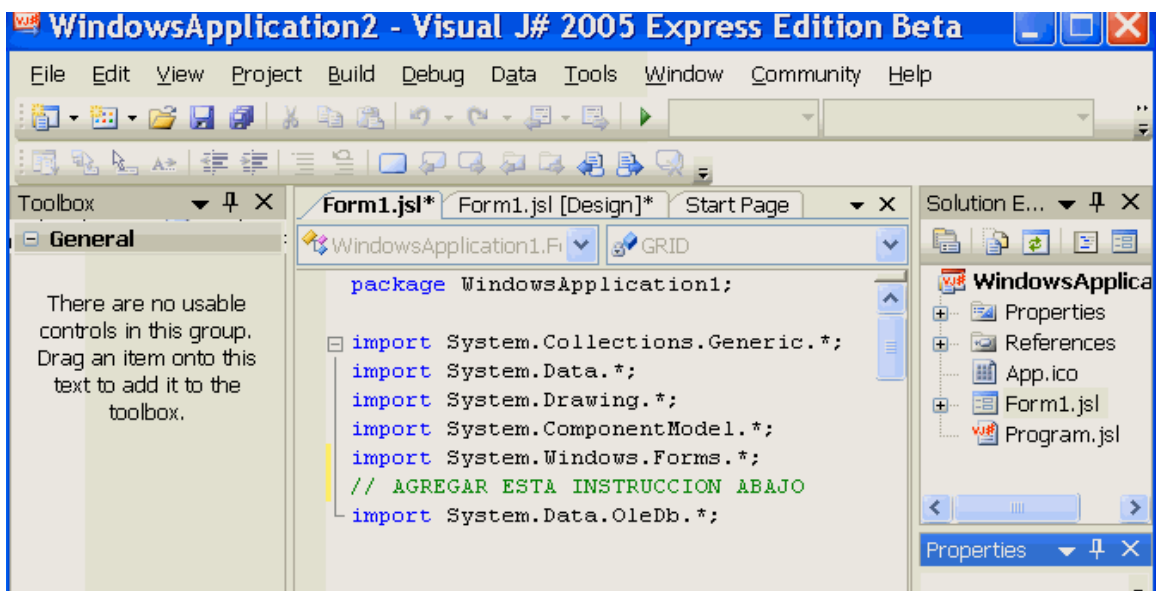
Esta instrucción se enviara a la base de datos usando un objeto DATAADAPTER

También es importante recordar que de las cuatro operaciones básicas de SQL ( ya leyeron el tutorial ??) SELECT, INSERT, UPDATE y DELETE sus formato INST SQL ??? FROM TABLA; afectara a todos los renglones de la tabla.

El procedimiento que se intenta seguir cuando se construya un programa asp net que tenga que manipular una tabla en disco deberá seguir los siguientes pasos:

- Crear una conexión o enlace a la base de datos.
- Abrir la conexión a la base de datos.
- Crear ADAPTER y cargarlo con la instrucción sql.
- Crear el dataset y cargarlo a través del adapter o del command.
- Cargar el DataGridView con el dataset y enlazarlo(binding)
- Procesar el DataGridView (editar un renglon, agregar un renglón, modificar un renglón, etc)
- Cerrar la conexión

Atención primero instalar o mejor dicho importar los drivers de oledb al programa solo cargar la pantalla de código y agregar la siguiente instrucción apropiada como lo muestra esta pantalla de ejemplo:



Código de Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // creando coneccion, adpater, dataset
    OleDbConnection CANAL;
    DataSet TABLA;
    OleDbDataAdapter ORDEN;
    // creando y enlazando coneccion a la base de datos
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    // recordar espacio en blanco en DATA SOURCE=
    // creando y cargando el adapter con la instruccion sql
    // recordar usar dataadapter para select's
    ORDEN = new OleDbDataAdapter("select * from mitabla", CANAL);
    // creando y cargando el dataset
    TABLA = new DataSet();
    ORDEN.Fill(TABLA, "mitabla");
    // cargando y enlazando el datagridview
    GRID.set_DataSource(TABLA);
    GRID.set_DataMember("mitabla");
}
```

Corrida:

clave	nombre	edad
1	oso	12
2	peach	8
3	raton	5

DataGridView en FORM1 las propiedades que se usaron en el programa fueron:

Name = GRID <-- nombre del grid dentro del programa

ColumnsHeadersVisible = true <-- Para que se muestren las celdas de encabezados

ReadOnly = True <-- Para evitar que el usuario entre a editar una celda

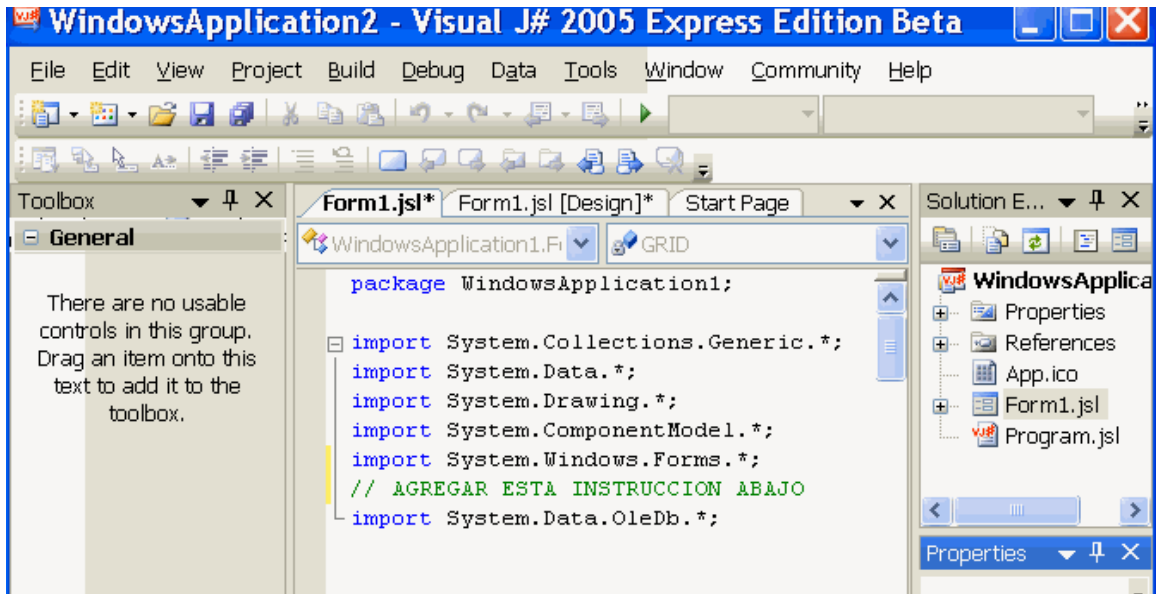
RowHeaderVisible = false <-- Para que no se muestre una columna de celdas con numero de renglon.

AllowUserToOrder, BackGroundColor, BorderStyle, CellBorderStyle, Enabled, DefaultCellStyle los pueden cargar con valores apropiados para darle buena presentación y funcionamiento a la tabla.

notas:

1. Se sigue el procedimiento genérico para procesar tablas usando ADO NET
2. Observar y siempre incluir los namespaces indicados.

Agregar el import System.Data.oledb.\* en la parte de imports arriba dentro del Form1.jsl, el encabezado de este programa debe quedar así:



3. Se usa un objeto button = SELECT con el código cargado en su evento onclick.
4. Recordar que DATAGRIDVIEW es un objeto por tanto hay que crearlo e inicializarlo al principio del programa, también recordar que datagridview tiene muchas propiedades que le mejoran la interfase con que se despliega y es en esta parte donde se cargan dichas propiedades.
5. Se empieza creando los objetos ADO a ocupar y abriendo la conexión a la base de datos, si se les hace muy grande la string del proveedor, pueden cargarla primero en una variable string y carguen la string en el constructor de la conexión, pero esto es opcional.
  - a. Recordar que hay otros proveedores de bases de datos para cuando se quieran acceder bases de datos diferentes de access.
6. Tomar nota como se hace una referencia a la base de datos, esto es en c:\\datos\\mibase.mdb ( ojo con las diagonales)
7. Se crea el adapter y se carga el constructor con la instrucción sql y la conexión.
8. Luego se creo el dataset y se cargo con toda la base de datos en disco, entender esto bien, dataset puede quedar cargado con todas las tablas que tenga la base de datos por eso se usa un FILL para pasar al dataset solo una de las tablas (mitabla), esto da origen a dos notas:
  - a. Al programar mas adelante se ocupara explícitamente indicarle al compilador con cual tabla se va a trabajar, es por esta razón que se verán instrucciones tales como tabla.tables["clientes"].etc.etc. aquí se esta diciendo al compilador que del dataset(TABLA) se va a realizar una proceso con la tabla de clientes.

- b. Para procesar dos o mas tablas, entonces se tendra que usar mucho el formato que se vio en la nota.
9. Al final se carga el datagridview , se enlaza al dataset y se cierra la base de datos.

TAREAS VISUAL J# 2005:

1. Construir y desplegar una primera base de datos que contenga la primera tabla que diseñaron en el tema de tablas.
2. Construir una segunda base de datos que contenga cuando menos tres de las tablas ya diseñadas y desplegar cualquiera de ellas usando una ventana de menú y ventanas para cada tabla, en menú el usuario selecciona cual quiere desplegar.

## 5.8. INSERCIÓN REGISTROS SQL INSERT

Insertar o agregar registros o renglones nuevos a una tabla en disco, es un proceso sencillo que usa la siguiente instrucción sql:

```
INSERT INTO TABLA(CAMPO1,CAMPO2..) VALUES(VALOR1,VALOR2..);
```

SQL INSERT se manda a la base de datos usando el objeto COMMAND

Recordar que solo se está usando lo mínimo de cada instrucción sql, es conveniente estudiar un tutorial de sql.

También recordar que INSERT, UPDATE y DELETE van dentro de un objeto COMMAND.

Recordar agregar `IMPORT SYSTEM.DATA.OLEDB.*;` en la parte de arriba o imports del programa o VISUAL J# 2005 no reconocerá ningún objeto ado.net.

Programa:

```
// un contador global
int cont = 0;
private void button1_Click_1(Object sender, System.EventArgs e)
{
    // creando conexión
    OleDbConnection CANAL;
    // creando y enlazando conexión a la base de datos
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    OleDbCommand ORDEN;
    // creando y cargando un objeto OLEDBCOMMAND
    // instrucción sql insert into mitabla(listacampos) values(listadatos)
    // @variable es una variable de tipo parámetro ( o textbox de la forma para que
    // se entienda mas)
    // y cada variable de estas ocupa un tipo de dato y un valor
    String q = "insert into mitabla(nombre,edad) values(@NOMBRE, @EDAD)";
    OleDbCommand ORDEN = new OleDbCommand(q, CANAL);
    ORDEN.Parameters.Add(new OleDbParameter("@NOMBRE", OleDbType.VarWChar, 20));
    ORDEN.Parameters.Get_Item("@NOMBRE").Set_Value(NOMBRE.GetText());
    ORDEN.Parameters.Add(new OleDbParameter("@EDAD", OleDbType.Integer));
    ORDEN.Parameters.Get_Item("@EDAD").Set_Value(EDAD.GetText());
    // recordar que command ocupa tres instrucciones extras que están abajo
    ORDEN.Connection.Open();
    ORDEN.ExecuteNonQuery();
    ORDEN.Connection.Close();
    // limpiando TEXTBOXS para otra inserción
    NOMBRE.SetText("");
    EDAD.SetText("");
    // avisando inserción
    cont = cont + 1;
    label4.SetText("REGISTRO no: " + System.Convert.ToString(cont) + " Insertado");
}
```



Corrida de programa:



Notas:

Se agregaron dos textboxes arriba para capturar los nuevos datos a insertar en la tabla.

Recordar que campo clave es de tipo autonumber y access se encarga de incrementarlo por su cuenta.

En función INSERTAR(), se crea la string q con el formato apropiado sql( como se dijo al principio de este tema), observar que existen dos variables que llevan un @ antes, estas variables se llaman VARIABLES PARÁMETROS y se cargan con el objeto command.get\_parameters()

Otra vez, en este ejemplo para mandar la instrucción sql a la base de datos se crea y se usa un objeto command (llamado orden) que lleva como datos la string q y la coneccion, pero se deben agregar dos métodos command.get\_parameters (orden.get\_parameters()) por cada textbox que se vaya a enviar a la tabla de la base de datos , en estos métodos se cargan las variables parámetro primero con el valor de dato del textbox y luego se transforman al tipo de dato apropiado usando los oledbtype(que hay que estudiar porque se tienen que asociar directamente a los tipos de datos que se usaron en access)

Ya con el objeto COMMAND (orden) listo y cargado para comunicar la instrucción sql a la base de datos se abre la conexión a la base de datos se manda el executeNonQuery (no se quiere regresar nada en esta parte, recordar la nota respectiva que se dio en un tema anterior) y se cierra la conexión y ya se mando el nuevo renglón a la base de datos en disco.

Para asegurarse que ya se efectuó la inserción en la base de datos, se tendra que usar el programa de consulta o despliegue (SELECT) de el tema anterior

TAREAS VISUAL J# 2005:

1. Construir muchos programas de inserción en las tablas de las bases de datos que tengan construidas
2. Ir Preparando un menú que contenga las opciones de consulta (select) e inserción (INSERT) para una tabla.

## 5.9. BUSQUEDA SQL SELECT

En este tema se analiza la búsqueda de un registro o renglón determinado en este proceso el usuario del programa quiere que se despliegue un y solo un registro de información proporcionando un dato de búsqueda generalmente la clave del registro.

La solución es sencilla, solo usar otra vez la instrucción select, con el siguiente formato:

```
SELECT [ *, all, campos] FROM TABLA WHERE clave=claveabuscar;  
Y recordar que select usa objeto dataadapter
```

Recordar también que agregando cláusulas condicionales where a algunas de las operaciones básicas de sql (select, delete, update) permite seleccionar un subconjunto de registros de la tabla para su procesamiento.

Se recuerda que deben buscar y estudiar un buen tutorial de sql.

Y no olvidar agregar el import system.data.oledb.\*; arriba en form1.jsl

Código

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // objetos OLEDB que se ocupan
    OleDbConnection CANAL;
    DataSet TABLA;
    OleDbDataAdapter ORDEN;
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    String q = "select * from mitabla where clave = @CLAVE";
    ORDEN = new OleDbDataAdapter(q, CANAL);
    ORDEN.get_SelectCommand().get_Parameters().Add(new OleDbParameter("@CLAVE", OleDbType.Integer));
    ORDEN.get_SelectCommand().get_Parameters().get_Item("@CLAVE").set_Value(CLAVE.get_Text());
    // creando el dataset y cargandolo
    TABLA = new DataSet();
    ORDEN.Fill(TABLA, "mitabla");
    // cargando y enlazando el datagridview
    GRID.set_DataSource(TABLA);
    GRID.set_DataMember("mitabla");
}
```

### Nota:

hay nada nuevo es una combinación de los programas anteriores con las mismas notas, solo se usa un textbox para pedir la clave, aunque se puede usar cualquier campo para buscar.

Corrida del Programa:

Form1

BUSQUEDA x CLAVE

CLAVE 4 BUSCAR

clave	nombre	edad
4	elefante	33

TAREAS VISUAL J# 2005:

1. hacer programas de búsquedas para las bases hechas e ir construyendo un programa de menú completo para una sola tabla, recordar que pueden construirlo usando procedimientos o mejor aun llamado a ventanas.

### 5.10. FILTROS SQL SELECT

Otro problema similar al anterior es el de filtros es decir en muchas ocasiones es necesario obtener información acerca de un subconjunto de renglones de la tabla.

Por ejemplo todos los estudiantes que sean mayores de 17 años, todos los clientes que sean de Tijuana, etc., a esto le llamamos filtros o condiciones.

También se resuelve de manera similar al anterior, es decir usando la instrucción select y objeto adapter etc, from tabla, where condición; y no olvidar poner el import oledb arriba en .jsl

Código:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // objetos OLEDB que se ocupan
    OleDbConnection CANAL;
    DataSet TABLA;
    OleDbDataAdapter ORDEN;
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    String q = "select * from mitabla where edad >= @EDAD";
    ORDEN = new OleDbDataAdapter(q, CANAL);
    ORDEN.get_SelectCommand().get_Parameters().Add(new OleDbParameter("@EDAD", OleDbType.Integer));
    ORDEN.get_SelectCommand().get_Parameters().get_Item("@EDAD").set_Value(EDAD.get_Text());
    // creando el dataset y cargandolo
    TABLA = new DataSet();
    ORDEN.Fill(TABLA, "mitabla");
    // cargando y enlazando el datagridview
    GRID.set_DataSource(TABLA);
    GRID.set_DataMember("mitabla");
}
```

Nota:

Es el programa anterior pero con otra condición WHERE pero seria prudente mejor usar dos combobox uno para la variable, otro para el operador relacional y un text para el dato y mandar estos tres datos al programa (se ocupan varios command.parameters()), pero eso queda de tarea.

Corrida:

Form1

FILTROS

EDAD > QUE 10 FILTRAR

clave	nombre	edad
1	oso	12
4	elefante	33

TAREAS VISUAL J# 2005:

1.- Preparar programas de filtrado para sus bases de datos y su programa de menú (acuérdense usar procedimientos y usar los objetos command, adapter, coneccion, dataset como globales o también pueden construir el programa de menú y en las opciones de menú ir activando una ventana correspondiente), recordar que sus formas deben construirlas con 2 combos y un text, suerte

### 5.11. OPERACIONES CAMPOS SQL UPDATE

Este es también un caso común con elementos de una tabla, sin embargo es también fácil de resolver.

Solo recordar la instrucción UPDATE ( usando objeto command), que se puede manejar con SET para definir los campos o columnas a cambiar y la cláusula WHERE que permite condicionar los renglones a actualizar.

El siguiente programa convierte la edad a meses de todos los renglones de la tabla.

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // objetos OLEDB que se ocupan
    OleDbConnection CANAL;
    OleDbCommand ORDEN;
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    String q = " UPDATE mitabla SET edad = edad * 12 ";
    ORDEN = new OleDbCommand(q, CANAL);
    // mandando la ORDEN
    ORDEN.get_Connection().Open();
    ORDEN.ExecuteNonQuery();
    ORDEN.get_Connection().Close();
    // avisando
    label1.set_Text("EADADES AUMENTADAS");
}
```

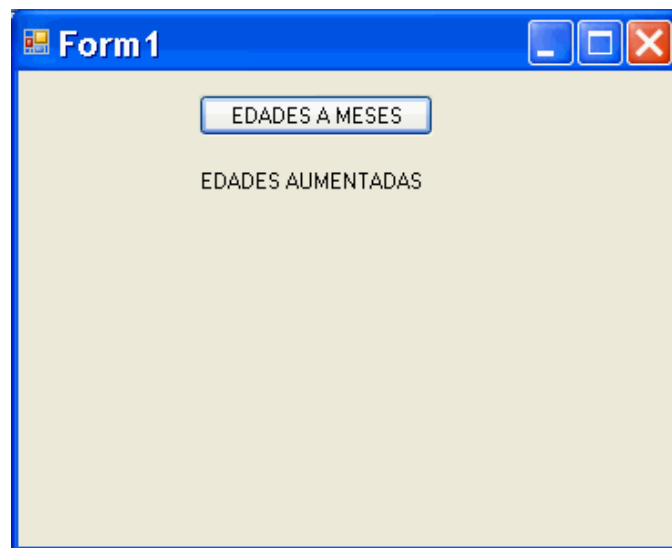
Nota:

Como olvidar el import oledb arriba y seguir estudiando y practicando su tutorial de SQL.

Y recordar que si usan TEXTBOX para el SET o el WHERE, deberán usar las variables parámetros (@TEXTBOX) y sus dos correspondientes orden.parameters

Si quieren ver los cambios usar el programa de consulta SELECT \* ya construido.

Corrida:



TAREAS VISUAL J# 2005:

1. Construir una tabla en access que traiga matricula, nombre, calif1, calif2, calif3 y promedio, cargar en access unos 5 renglones de alumnos, no cargar promedio, el promedio lo deberán calcular en un programa.
2. Seguir construyendo su menú con varias opciones de update.

## 5.12. BAJA O ELIMINACION SQL DELETE

Eliminación es otro proceso simple y común con las bases de datos el modelo con ADO NET que estamos usando hace este tipo de operaciones muy fáciles:

La instrucción sql a usar es:

DELETE FROM TABLA WHERE CONDICION  
Y DELETE USA OBJETO COMMAND

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // objetos OLEDB que se ocupan
    OleDbConnection CANAL;
    OleDbCommand ORDEN;
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    // instruccion sql DELETE FROM TABLA WHERE CLAVE=DATO
    String q = "delete from mitabla where clave=@CLAVE";
    ORDEN = new OleDbCommand(q, CANAL);
    ORDEN.get_Parameters().Add(new OleDbParameter("@CLAVE", OleDbType.Integer));
    ORDEN.get_Parameters().get_Item("@CLAVE").set_Value(CLAVE.get_Text());
    ORDEN.get_Connection().Open();
    ORDEN.ExecuteNonQuery();
    ORDEN.get_Connection().Close();
    // avisando
    label2.set_Text(" REGISTRO ELIMINADO");
}
```

notas: estudiar y analizar el código

Corrida:



TAREAS VISUAL J# 2005:

1. Construir este proceso para las tablas y bases de datos que tengan del programa de menu.

### 5.13. EDICION DE REGISTROS SQL UPDATE

Editar registros significa cambiar el contenido de algunos de los campos o columnas por nueva información o para corregir algún error de captura original o para agregar alguna columna que no existía por modificación de la tabla o la base de datos.

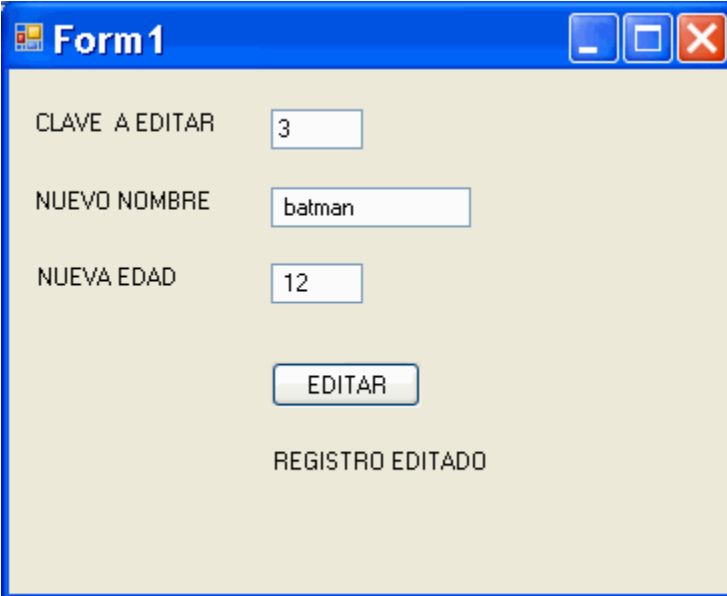
En general se tiene otro problema de sql UPDATE (usando command por supuesto), con una serie de textboxes arriba para capturar los nuevos datos.

Y no se olviden del import oledb arriba en form1.jsl

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // objetos OLEDB que se ocupan
    OleDbConnection CANAL;
    OleDbCommand ORDEN;
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    // instruccion sql UPDATE
    String q = "Update mitabla set nombre=@NOMBRE, edad=@EDAD where clave=" + CLAVE.get_Text();
    ORDEN = new OleDbCommand(q, CANAL);
    // cargando parámetros
    ORDEN.get_Parameters().Add(new OleDbParameter("@NOMBRE", OleDbType.VarWChar, 20));
    ORDEN.get_Parameters().get_Item("@NOMBRE").set_Value(NOMBRE.get_Text());
    ORDEN.get_Parameters().Add(new OleDbParameter("@EDAD", OleDbType.Integer));
    ORDEN.get_Parameters().get_Item("@EDAD").set_Value(EDAD.get_Text());
    // mandando sql a base de datos
    ORDEN.get_Connection().Open();
    ORDEN.ExecuteNonQuery();
    ORDEN.get_Connection().Close();
    // limpiando TEXTBOXS para otra edicion
    NOMBRE.set_Text(" ");
    EDAD.set_Text(" ");
    // avisando edicion
    label4.set_Text("REGISTRO EDITADO");
}
```

Corrida:



Form1

CLAVE A EDITAR 3

NUEVO NOMBRE batman

NUEVA EDAD 12

EDITAR

REGISTRO EDITADO

Realmente este programa debe combinarse con el de búsqueda por ejemplo en un panel buscar y desplegar el registro y en otro panel este programa que hace la edición.

También se puede poner un botón que active el programa con la ventana de búsqueda para que el usuario vea el registro original y en un panel poner este código de edición.

En ambos casos queda de tarea pero recordar que el menú que ya deben estar construyendo ya puede activar la ventana de búsqueda.

Un registro editado o modificado, analizar con cuidado el código del programa, que esta documentado

#### TAREAS VISUAL J# 2005:

1. Construir el modulo o procedimiento de edición al sistema de menú que están construyendo

## 5.14. GRAFICOS PICTUREBOX

Campos de gráficos o de imágenes, se han convertido en elementos importantes de cualquier base de datos. Para manejar este elemento con ado asp net existen dos maneras:

1.- Agregar un campo BLOB a la tabla en Access y usar componentes asp net especializado en imágenes tanto para subirlas como para desplegar la imagen.

Este método provoca que la base de datos crezca mucho recordar que una imagen aun de tipo jpg ocupa mucho espacio.

2.- El segundo método es mas sencillo primero poner las imágenes ( de preferencia jpg) en tu folder donde esta la base de datos, después agregar un objeto PictureBox en el programa y además agregar un campo de texto llamado foto a la tabla en access y grabar el nombre de la imagen en este campo, por ejemplo pato.jpg

Usar el programa de búsqueda normal ya visto y agregarle en tiempo real la propiedad apropiada al componente PictureBox, como lo muestra el programa ejemplo.

Programa:

```
private void button1_Click(Object sender, System.EventArgs e)
{
    // objetos OLEDB que se ocupan
    OleDbConnection CANAL;
    DataSet TABLA;
    OleDbDataAdapter ORDEN;
    CANAL = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\\datos\\mibase.mdb");
    String q = "select * from mitabla where clave = @CLAVE";
    ORDEN = new OleDbDataAdapter(q, CANAL);
    ORDEN.get_SelectCommand().get_Parameters().Add(new OleDbParameter("@CLAVE", OleDbType.Integer));
    ORDEN.get_SelectCommand().get_Parameters().get_Item("@CLAVE").set_Value(CLAVE.get_Text());
    // creando el dataset y cargandolo
    TABLA = new DataSet();
    ORDEN.Fill(TABLA, "mitabla");
    // cargando y enlazando el datagridview
    GRID.set_DataSource(TABLA);
    GRID.set_DataMember("mitabla");
    // cargando la imagen
    String temp = System.Convert.ToString(
    TABLA.get_Tables().get_Item("mitabla").get_Rows().get_Item(0).get_Item(3));
    // cargando picturebox
    pictureBox1.set_Image(Image.FromFile("c:\\datos\\" + temp));
}
```

Corrida:

clave	nombre	edad	foto
1	oso	144	os

Recordar que para que no salga el campo fotourl en el datagridVIEW el select de sql puede pedirse como select campo1, campo2, campo. from mi tabla etcétera.

Proyecto programación VISUAL J# 2005 construir un sistema completo integrado con todas las operaciones usando al menos una tabla.

Por último recordar que el componente datagridview es uno de los controles que mas ha sido modificado y actualizado por microsoft, entre sus principales cambios es la cantidad increíble de eventos que puede detectar y de hecho muchos de los procesos que se estudiaron en esta unidad se podrian facilitar mucho si se usaran los eventos de dataviewgrid, sin embargo por razones pedagógicas se selecciono el modelo seguido en la unidad, sin embargo en la próxima actualización de este libro se analizaran y estudiaran todos estos nuevos conceptos.