C#

En el código QR tendrás acceso a los tutoriales realizados por Diego Moiseet de Espanes, para mi opinión uno de los mejores curso de programación que he podido realizar y con este manual con el resumen del mismo intento aportar un granito de arena a este fantástico mundo de la programación:

CURSO DE C#

DESDE NIVEL 0

PERE MANEL VERDUGO ZAMORA

pereverdugo@gmail.com

Contenido

Capítulo 1.- Instalación de visual Studio.Net. 6
Capítulo 2.- Creación de un proyecto 13
Capítulo 3.- Codificación del diagrama de flujo 20
Capítulo 4.- Errores sintácticos y lógicos 28
Capítulo 5.- Estructura de programación secuencial – 1 31
Capítulo 6.- Estructura de programación secuencial – 2 33
Capítulo 7.- Estructura de programación secuencial – 3 34
Capítulo 8.- Estructura de programación secuencial – 4 35
Capítulo 9.- Estructura de programación secuencial – 5 36
Capítulo 10.- Estructuras condicionales simples y compuestas – 1 37
Capítulo 11.- Estructuras condicionales simples y compuestas – 2 40
Capítulo 12.- Estructuras condicionales simples y compuestas – 3 43
Capítulo 13.- Estructuras condicionales simples y compuestas – 4 45
Capítulo 14.- Estructuras condicionales simples y compuestas – 5 47
Capítulo 15.- Estructuras condicionales anidadas – 1 48
Capítulo 16.- Estructuras condicionales anidadas – 2 52
Capítulo 17.- Estructuras condicionales anidadas – 3 54
Capítulo 18.- Estructuras condicionales anidadas – 4 56
Capítulo 19.- Estructuras condicionales anidadas – 5 58
Capítulo 20.- Condiciones compuestas con operadores lógicos – 1 60
Capítulo 21.- Condiciones compuestas con operadores lógicos – 2 63
Capítulo 22.- Condiciones compuestas con operadores lógicos – 3 66
Capítulo 23.- Condiciones compuestas con operadores lógicos – 4 67
Capítulo 24.- Condiciones compuestas con operadores lógicos – 5 68
Capítulo 25.- Condiciones compuestas con operadores lógicos – 6 69
Capítulo 26.- Condiciones compuestas con operadores lógicos – 7 70
Capítulo 27.- Condiciones compuestas con operadores lógicos – 8 73
Capítulo 28.- Condiciones compuestas con operadores lógicos – 9 75
Capítulo 29.- Estructura repetitiva while – 1 77
Capítulo 30.- Estructura repetitiva while – 2 82
Capítulo 31.- Estructura repetitiva while – 3 84
Capítulo 32.- Estructura repetitiva while – 4 87
Capítulo 33.- Estructura repetitiva while – 5 90
Capítulo 34.- Estructura repetitiva while – 6 92
Capítulo 35.- Estructura repetitiva while – 7 93

Capítulo 36.- Estructura repetitiva while – 8 95
Capítulo 37.- Estructura repetitiva while – 9 96
Capítulo 38.- Estructura repetitiva while – 10 97
Capítulo 39.- Estructura repetitiva while – 11 99
Capítulo 40.- Estructura repetitiva for – 1 101
Capítulo 41.- Estructura repetitiva for – 2 104
Capítulo 42.- Estructura repetitiva for – 3 106
Capítulo 43.- Estructura repetitiva for – 4 108
Capítulo 44.- Estructura repetitiva for – 5 111
Capítulo 45.- Estructura repetitiva for – 6 113
Capítulo 46.- Estructura repetitiva for – 7 114
Capítulo 47.- Estructura repetitiva for – 8 116
Capítulo 48.- Estructura repetitiva for – 9 117
Capítulo 49.- Estructura repetitiva for – 10 119
Capítulo 50.- Estructura repetitiva for – 11 121
Capítulo 51.- Estructura repetitiva for – 12 124
Capítulo 52.- Estructura repetitiva for – 13 126
Capítulo 53.- Estructura repetitiva do while – 1 129
Capítulo 54.- Estructura repetitiva do while – 2 132
Capítulo 55.- Estructura repetitiva do while – 3 134
Capítulo 56.- Estructura repetitiva do while – 4 137
Capítulo 57.- Estructura repetitiva do while – 5 139
Capítulo 58.- Cadena de caracteres 141
Capítulo 59.- Declaración de una clase y definición de objetos – 1 144
Capítulo 60.- Declaración de una clase y definición de objetos – 2 150
Capítulo 61.- Declaración de una clase y definición de objetos – 3 153
Capítulo 62.- Declaración de una clase y definición de objetos – 4 156
Capítulo 63.- Declaración de una clase y definición de objetos – 5 158
Capítulo 64.- Declaración de una clase y definición de objetos – 6 160
Capítulo 65.- Declaración de métodos – 1 162
Capítulo 66.- Declaración de métodos – 2 165
Capítulo 67.- Estructura de datos tipo vector – 1 168
Capítulo 68.- Estructura de datos tipo vector – 2 171
Capítulo 69.- Estructura de datos tipo vector – 3 173
Capítulo 70.- Estructura de datos tipo vector – 4 175
Capítulo 71.- Estructura de datos tipo vector – 5 177

Capítulo 72.- Estructura de datos tipo vector – 6 179
Capítulo 73.- Estructura de datos tipo vector – 7 182
Capítulo 74.- Vector (Tamaño de un vector) – 1 184
Capítulo 75.- Vector (Tamaño de un vector) – 2 186
Capítulo 76.- Vectores paralelos 188
Capítulo 77.- Vectores (mayor y menor elemento) – 1 190
Capítulo 78.- Vectores (mayor y menor elemento) – 2 192
Capítulo 79.- Vectores (ordenamiento) – 1 194
Capítulo 80.- Vectores (ordenamiento) – 2 200
Capítulo 81.- Vectores (ordenamiento) – 3 202
Capítulo 82.- Vectores (ordenamiento con vectores paralelos) – 1 205
Capítulo 83.- Vectores (ordenamiento con vectores paralelos) – 2 208
Capítulo 84.- Estructura de datos tipo matriz – 1 211
Capítulo 85.- Estructura de datos tipo matriz – 2 213
Capítulo 86.- Estructura de datos tipo matriz – 3 215
Capítulo 87.- Estructura de datos tipo matriz – 4 217
Capítulo 88.- Matrices (cantidad de filas y columnas) – 1 219
Capítulo 89.- Matrices (cantidad de filas y columnas) – 2 221
Capítulo 90.- Matrices (cantidad de filas y columnas) – 3 223
Capítulo 91.- Matrices (cantidad de filas y columnas) – 4 225
Capítulo 92.- Matrices y vectores paralelos – 1 227
Capítulo 93.- Matrices y vectores paralelos – 2 230
Capítulo 94.- Matrices irregulares o dentadas – 1 233
Capítulo 95.- Matrices irregulares o dentadas – 2 236
Capítulo 96.- Matrices irregulares o dentadas – 3 238
Capítulo 97.- Constructor de la clase – 1 240
Capítulo 98.- Constructor de la clase – 2 242
Capítulo 99.- Constructor de la clase – 3 244
Capítulo 100.- Constructor de la clase – 4 246
Capítulo 101.- Colaboración de clases – 1 247
Capítulo 102.- Colaboración de clases – 2 252
Capítulo 103.- Colaboración de clases – 3 255
Capítulo 104.- Concepto de propiedad – 1 257
Capítulo 105.- Concepto de propiedad – 2 261
Capítulo 106.- Concepto de propiedad – 3 264
Capítulo 107.- Herencia – 1 267

Capítulo 108.- Herencia – 2 271
Capítulo 109.- Orden de ejecución de los constructores con herencias 274
Capítulo 110.- Clase parcial (partial class) 277
Capítulo 111.- interfaces visuales (Windows Forms) 282
Capítulo 112.- Captura de eventos de controles visuales (Windows Forms) 291
Capítulo 113.- Controles visuales, Button, Label, TextBox (Windows Forms) 300
Capítulo 114.- Control visual CheckBox (Windows Forms) 303
Capítulo 115.- Controles visuales RadioButton, GroupBox y Panel (Windows Forms) 307
Capítulo 116.- Control visual ComboBox 311
Capítulo 117.- Control visual ListBox (Windows Forms) 315
Capítulo 118.- Controles visuales MenuStrip y ToolStripMenuItem 320
Capítulo 119.- Controles visuales ContexMenuStrip y ToolStripMenuItem 328
Capítulo 120.- Controles visuales ToolStrip, ToolStripButton, ToolStriptComoBox, ToolStripTextBox, etc. 331
Capítulo 121.- Control visual StatusStrip (Windows Forms) 339
Capítulo 122.- Control Visual Container ToolStrip (Windows Forms) 343
Capítulo 123.- Aplicación de varios Form (Windows Forms) 348
Capítulo 124.- Aplicación con varios Form – Comunicación de datos por propiedades (Windows Forms) 353
Capítulo 125.- Controles visuales OpenFileDialog, SaveFileDialog, FontDialog y ColorDialog . 361
Capítulo 126.- Control visual DateTimePicker (Windows Forms) 372
Capítulo 127.- Control visual MonthCalendar (Windows Forms) 377
Capítulo 128.- Control visual CheckedListBox (Windows Forms) 382
Capítulo 129.- Control visual PictureBox (Windows Forms) 395
Capítulo 130.- Control Timer (Windows Forms) 407
Capítulo 131.- Control DataGridView (Windows Forms) 412
Capítulo 132.- Control DataGridView – Calendario de actividades (Windows Forms) 425
Capítulo 133.- clase Graphics – métodos más comunes. 437
Capítulo 134.- Graphics – Ejercicio de gráfico de tarta 444
Capítulo 135.- Clase Graphics – Ejercicio de gráfico de barras 448
Capítulo 136.- Creación de controles en forma dinámica (Windows Form) 451
Capítulo 137.- Creación de controles de usuario – 1 454
Capítulo 138.- Creación de controles de usuario – 2 461
Capítulo 139.- Insertar filas en una tabla empleando solo la clase SqlConnetion y SqlCommand 468
Capítulo 140.- Consultar filas empleando solo las clases SqlConnection y SqlDataReader 482

Capítulo 141.- Borrado de filas de una tabla empleando solo las clases SqlConnection y SqlCommand. 485

Capítulo 142.- Modificación de filas de una tabla empleando solo las clases SqlConnection y SqlCommand 488

Capítulo 143.- Mostrar filas en un control DataGridView recuperando los datos de SqlServer 491

Capítulo 144.- Inyección de SQL y Consultas parametrizadas 499

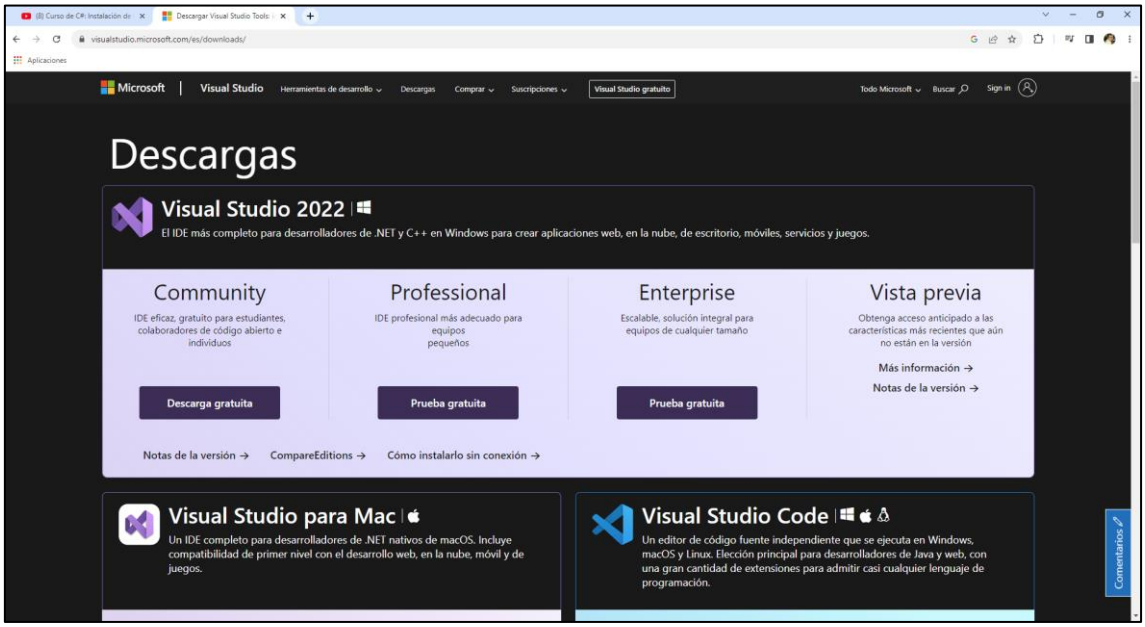
Capítulo 1.- Instalación de visual Studio.Net.

El objetivo de este tutorial es iniciarse en el arte de la programación desde cero empleando el lenguaje C# desarrollado por Microsoft.

Nos conviene utilizar este lenguaje para iniciarnos si nuestro objetivo final será desarrollar aplicaciones de escritorio, web o móviles en un futuro.

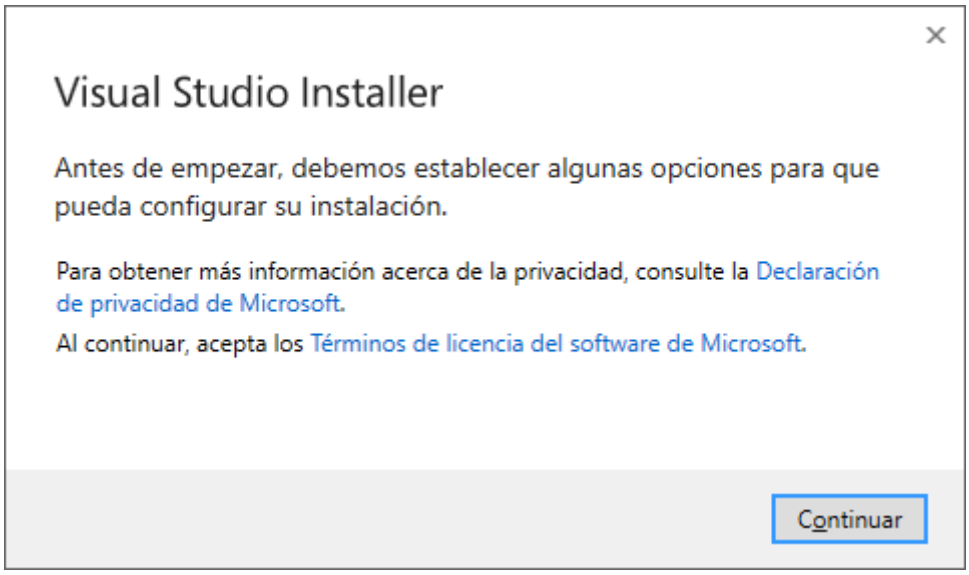
Lo primero será descargar el programa que vamos a necesitar para programar en C#,

Iremos la siguiente enlace: https://visualstudio.microsoft.com/es/downloads/

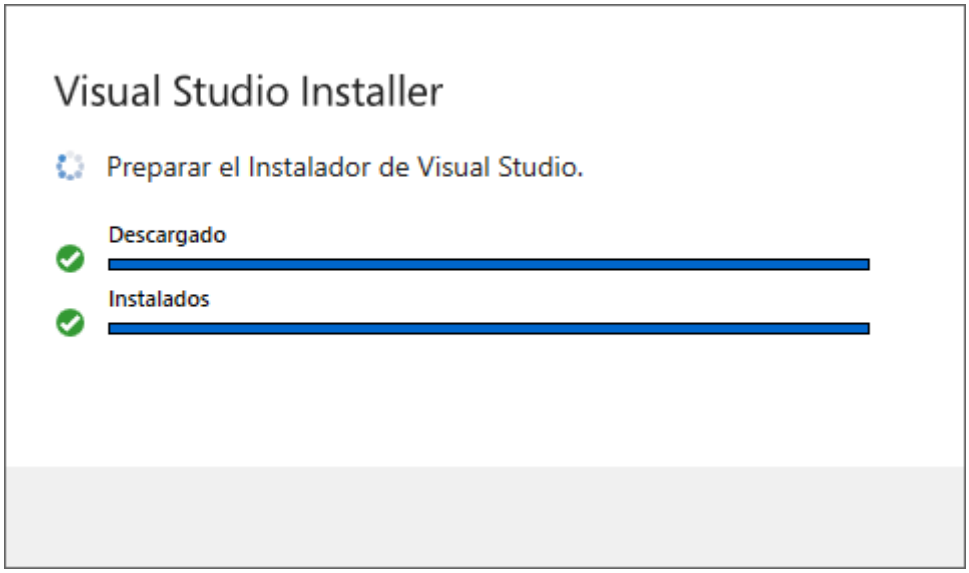


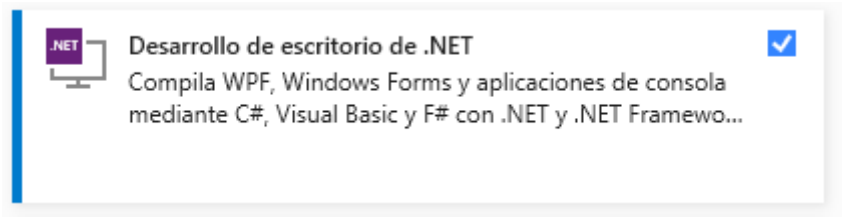
Seleccionaremos Community, IDE eficaz, gratuito para estudiantes y colaboradores del código abierto e individuos.

Procederemos a su instalación:

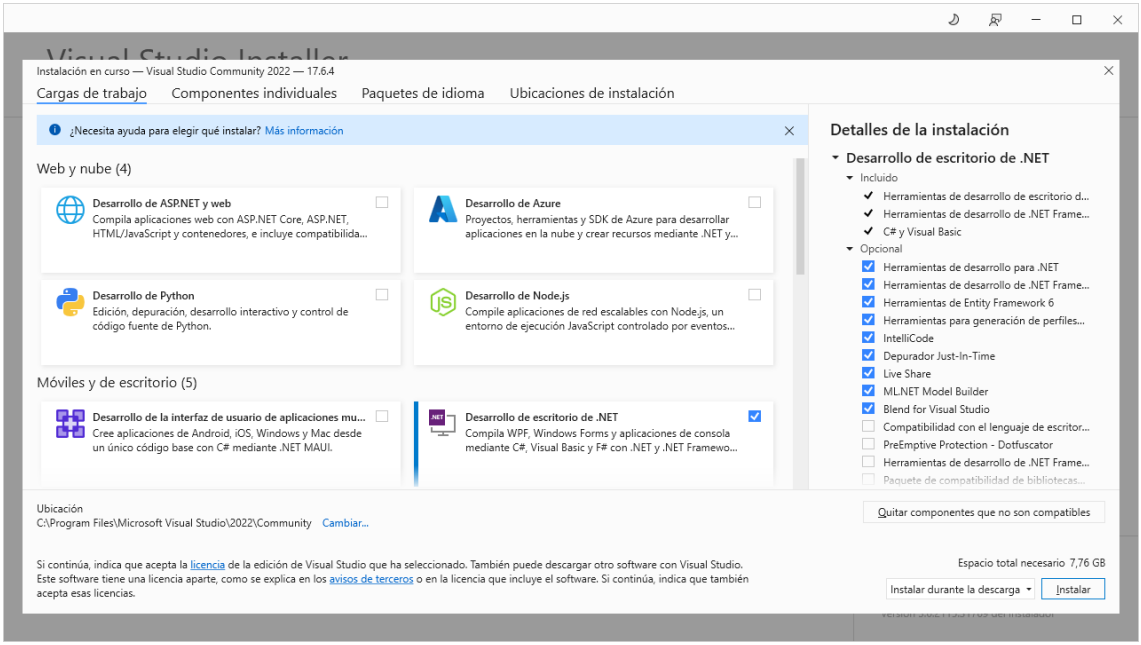


Le damos a continuar.

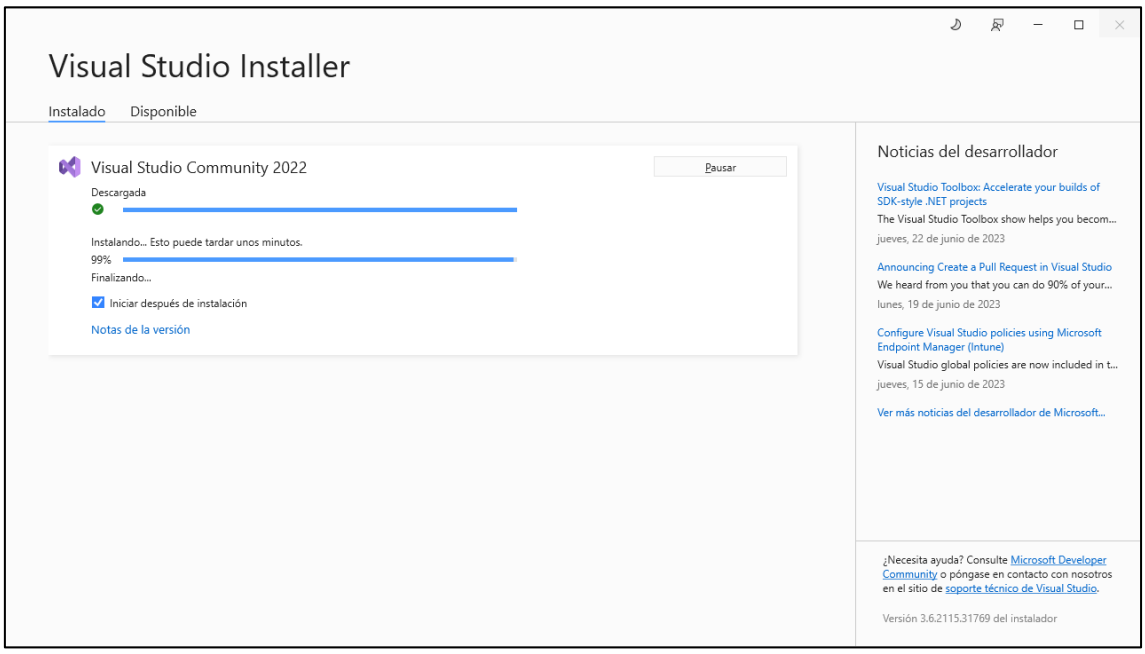




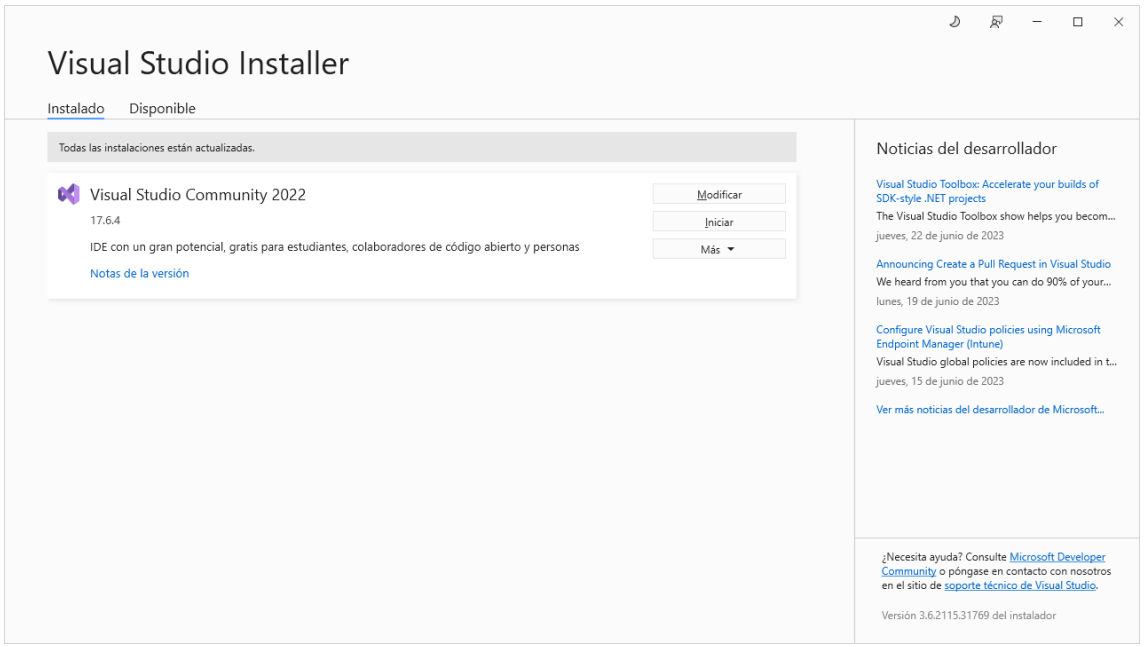
Solo vamos a descargar esta opción.

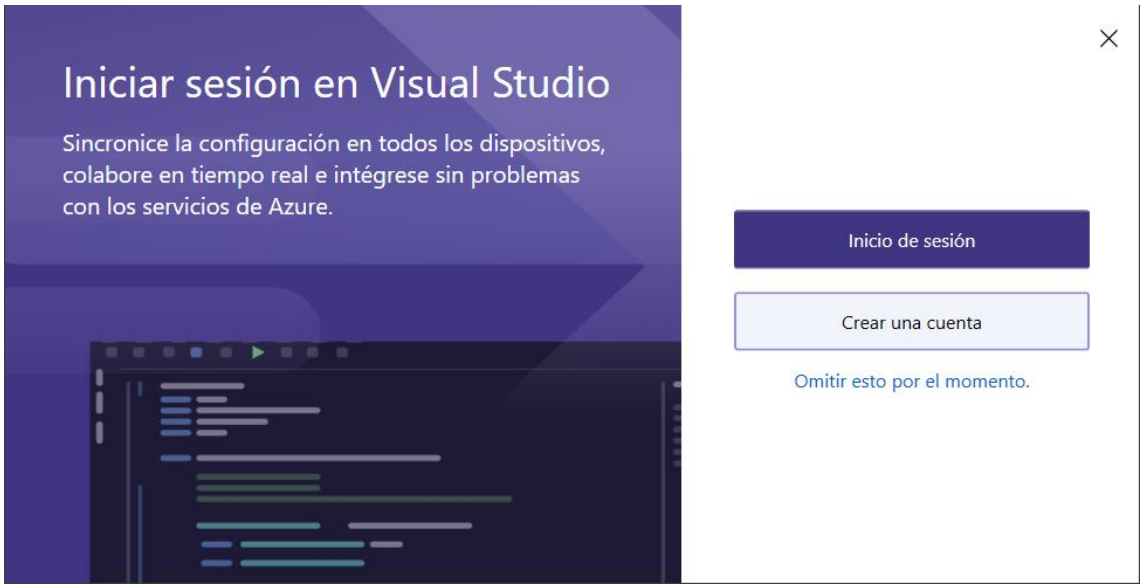


Le damos al botón instalar.



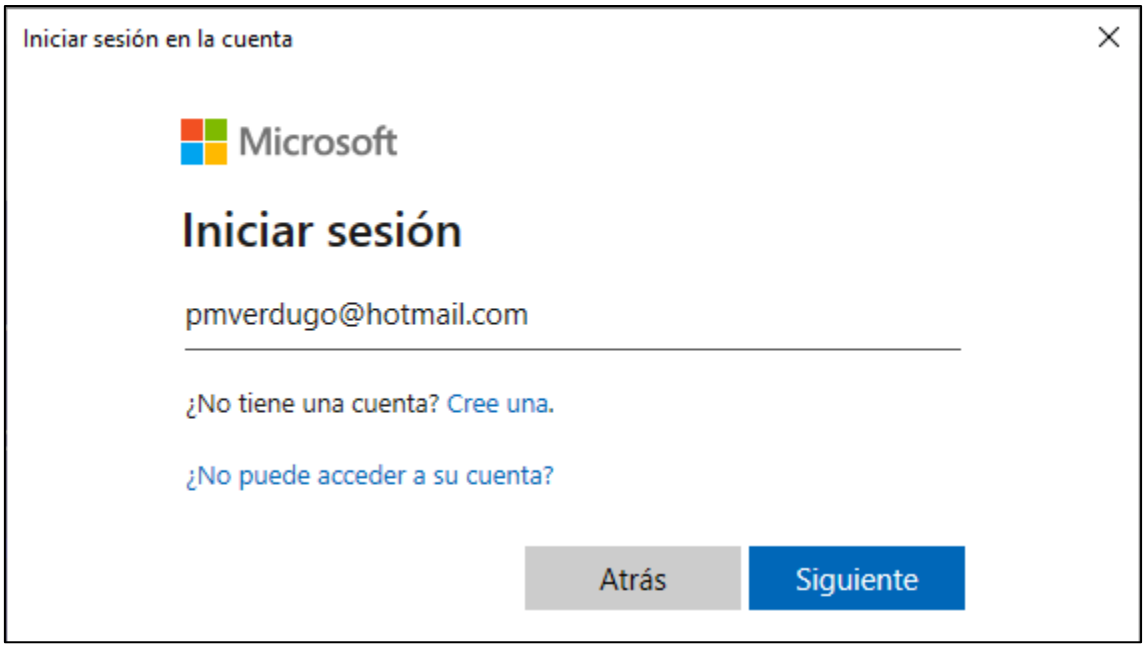
Esto puede durar un ratito.

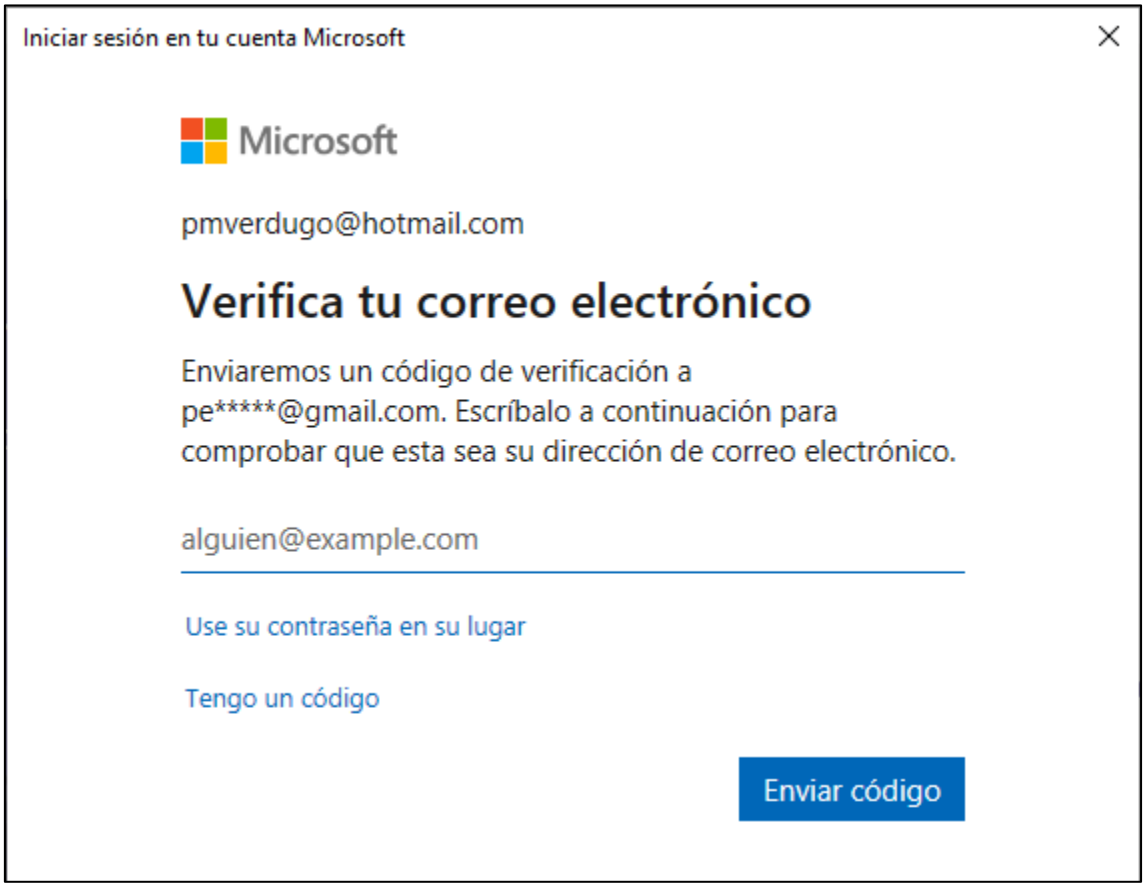




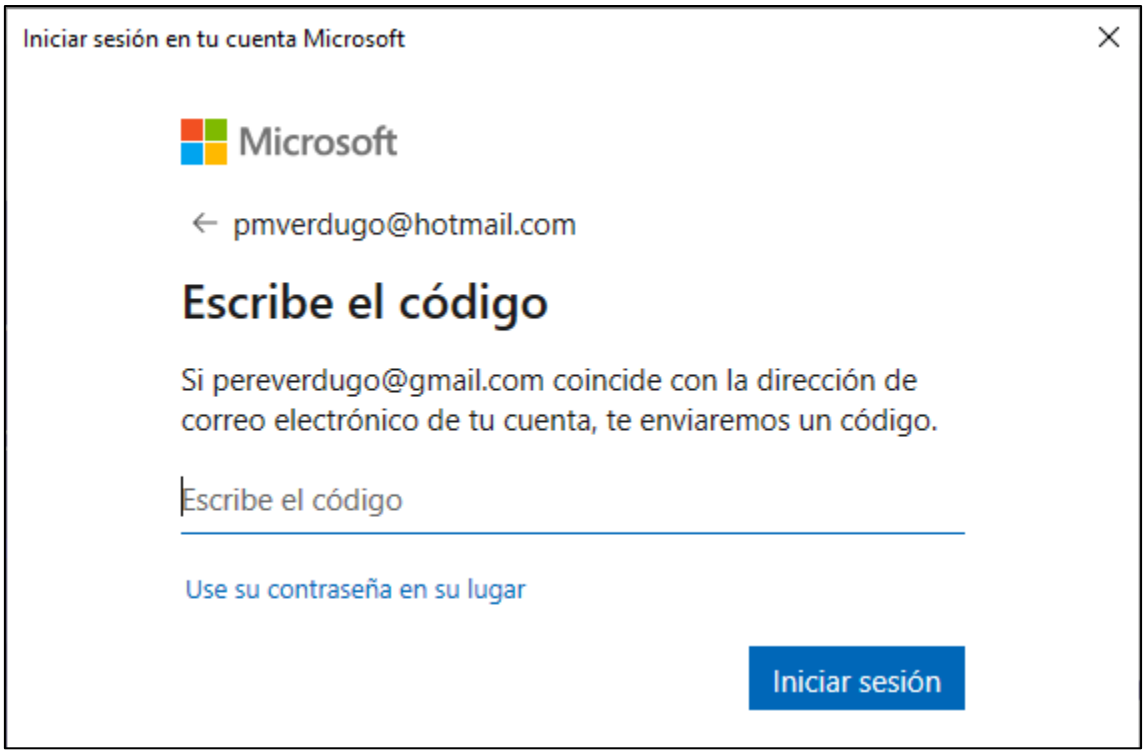
Tenemos que crear una cuenta con un correo Hotmail o Outlook.

Como ya la tengo vamos a inicio sesion:

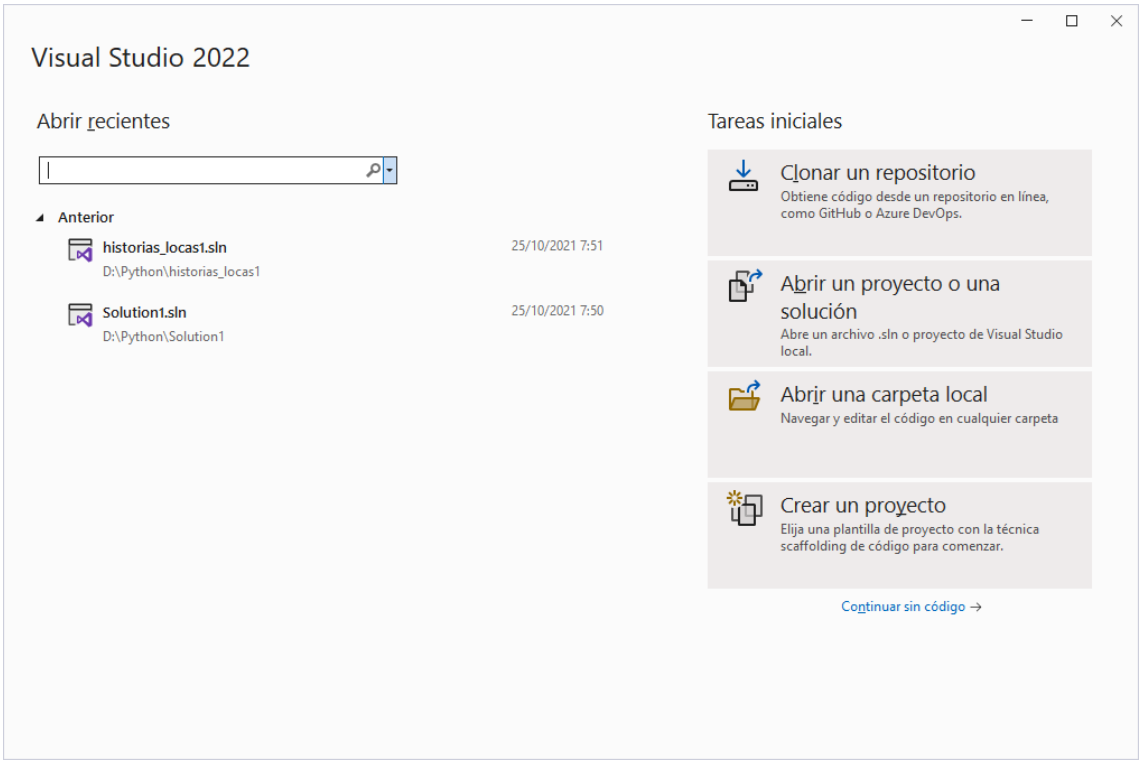




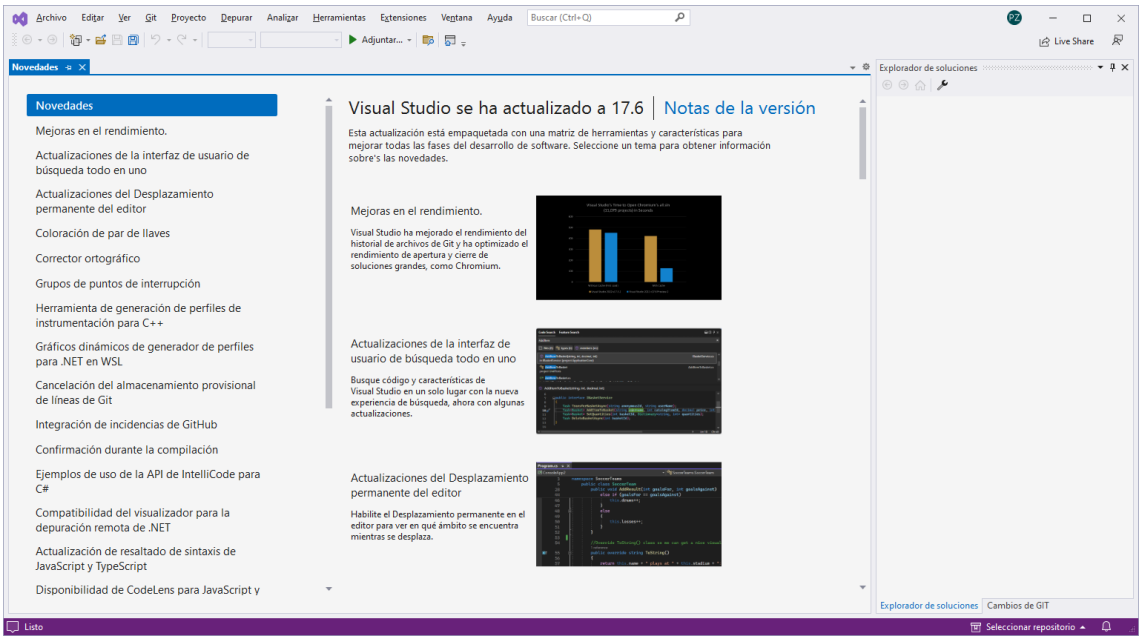
Escribe el correo donde quieres recibir el código:



Introduce el código y le damos a iniciar sesión.



Le demos a Continuar sin código:



Ya hemos terminado con la instalación.

Capítulo 2.- Creación de un proyecto

El curso está ideado para ser desarrollado por una persona que no conoce nada de programación y se utilice C# como primer lenguaje.

El objetivo fundamental de este tutorial es permitir que el estudiante pueda resolver problemas de distinta índole (matemáticos, administrativos, gráficos, contables, etc.) empleando como herramienta su ordenador.

Hay que tener en cuenta que para llegar a ser programador se debe recorrer un largo camino donde cada tema es fundamental para conceptos futuros. Es importante no dejar temas sin entender y relacionar.

La programación a diferencia de otras materias como podría ser la historia requiere un estudio metódico y ordenado (en historia se puede estudiar la edad media sin tener grandes conocimientos de la edad antigua).

La programación es una actividad nueva para el estudiante, no hay en los estudios primarios y secundarios una materia parecida.

Es bueno tenerse paciencia cuando los problemas no se resuelven por completo, pero es de fundamental importancia dedicar tiempo al análisis de los problemas.

¿Qué es un programa?

Programa: Conjunto de instrucciones que entiende un ordenador para realizar una actividad. Todo programa tiene un objetivo bien definido: un procesador de textos en un programa que permite cargar, modificar e imprimir textos, un programa de ajedrez permite jugar al ajedrez contra el ordenador u otro contrincante humano.

La actividad fundamental del programador es resolver problemas empleando el ordenador como herramienta fundamental.

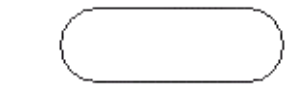
Para la resolución de un problema hay que plantear un algoritmo.

Algoritmo: Son los pasos a seguir para resolver un problema.

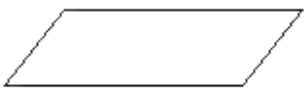
Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un ALGORITMO.

Los símbolos gráficos a utilizar para el planteo de diagramas de flujo son:



Inicio y fin del diagrama.



Entrada de Datos

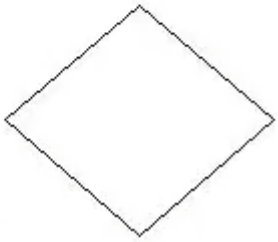
(Vamos a considerar que las entradas de datos se realizan siempre por el teclado de la computadora).

(Vamos a considerar que las entradas de datos se realizan siempre por el teclado de la computadora).

Salida de Datos

(Vamos a considerar que las salidas de datos se realizan siempre por la pantalla de la computadora).

Operación



Condición

Estos son los elementos esenciales que intervienen en el desarrollo de un diagrama de flujo.

Creación de un proyecto en C#

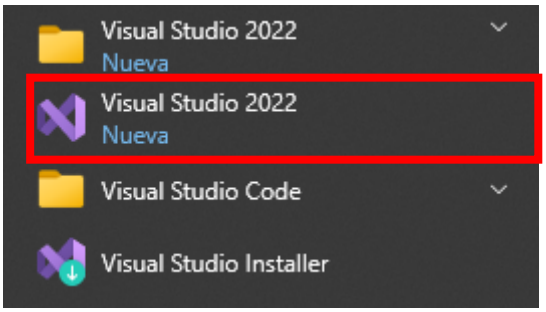
Consideraciones a tener en cuenta en cada proyecto:

Hay que tener en cuenta que el entorno de programación de "Microsoft Visual Studio" no ha sido desarrollado pensando en un principiante de la programación. Lo mismo ocurre con el propio lenguaje C#, es decir su origen no tiene como objetivo el aprendizaje de la programación. Debido a estos dos puntos veremos que a medida que avancemos con el tutorial muchos conceptos que iremos dejando pendientes se irán aclarando.

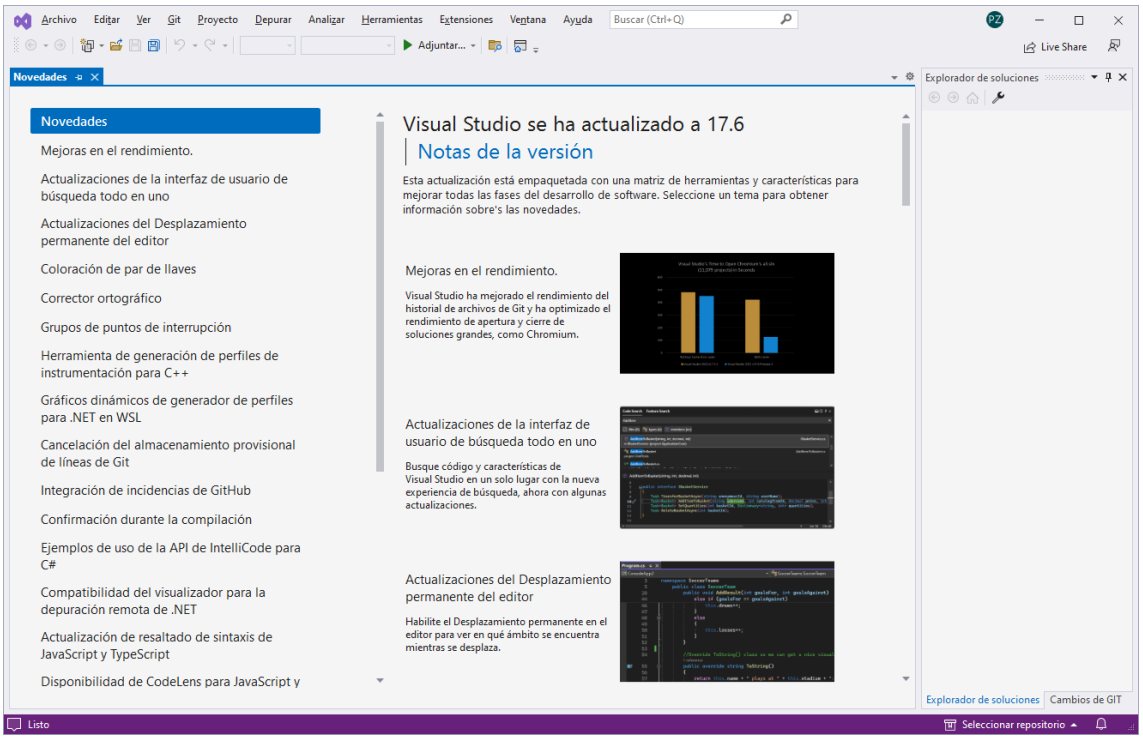
Veremos los pasos para la creación de un proyecto en C#.

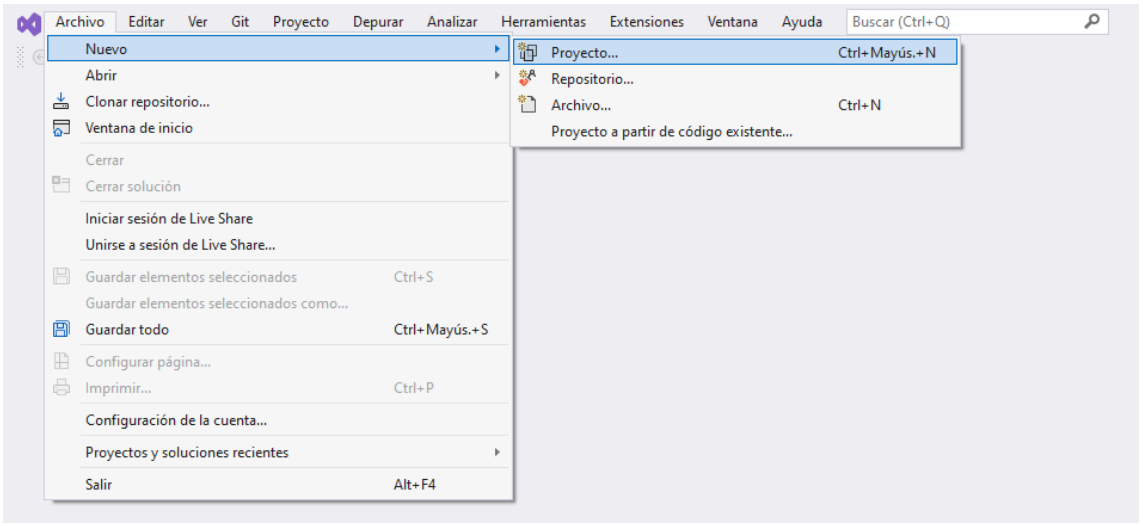
Pasos.

1.- Ingresamos al "Microsoft Visual Studio".

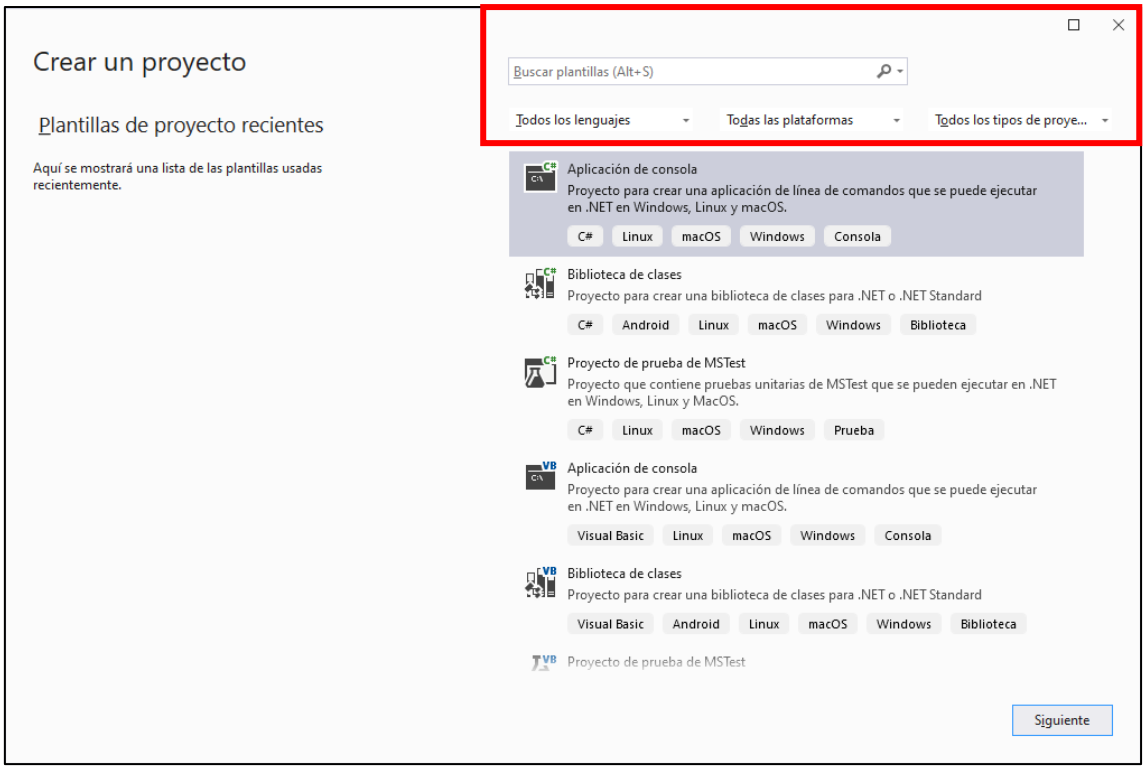


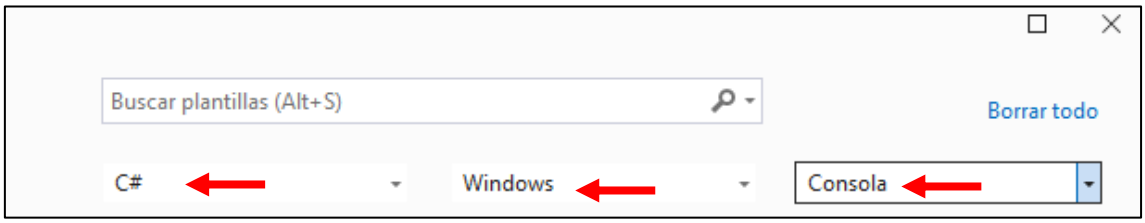
Aparece un diálogo donde tenemos distintas opciones para abrir proyectos anteriores, clonar proyectos almacenados en GitHub, etc, nosotros elegiremos la opción de 'Continuar sin código' para conocer el entorno de Visual Studio 2022:



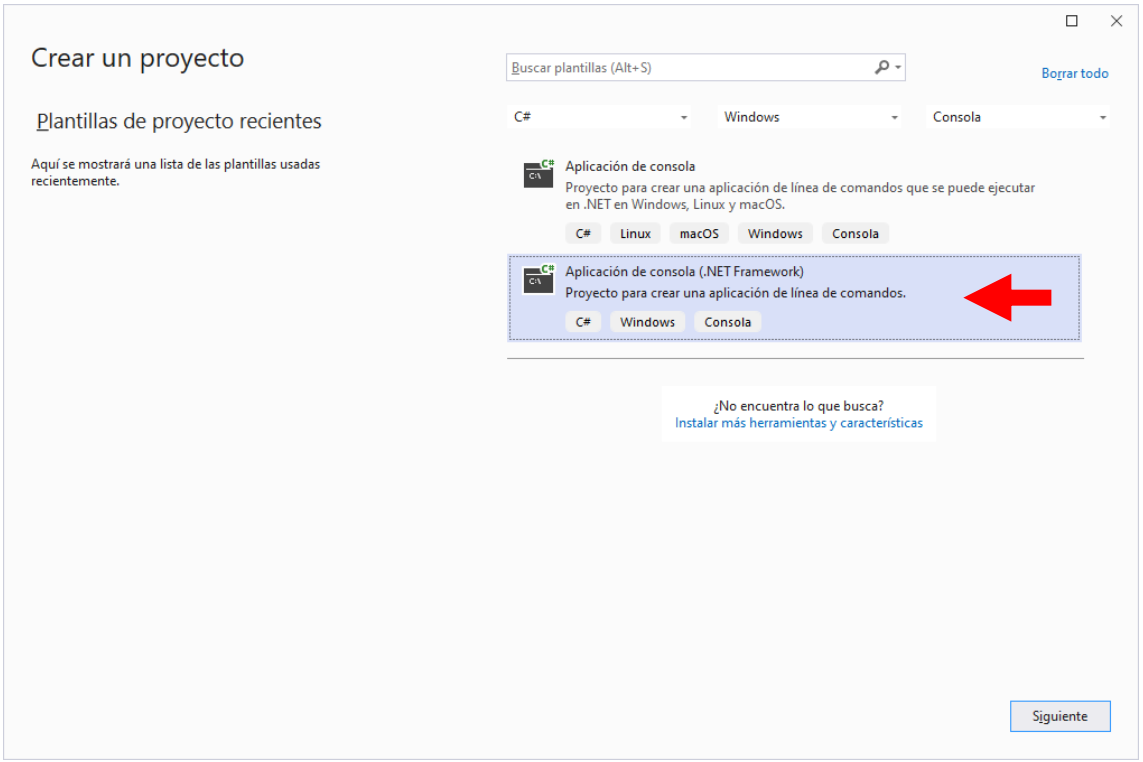


Del menú Archivo seleccionaremos Nuevo y de este proyecto.

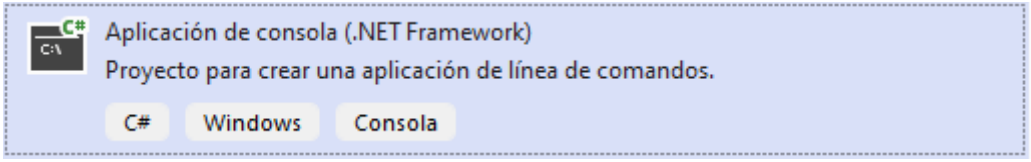




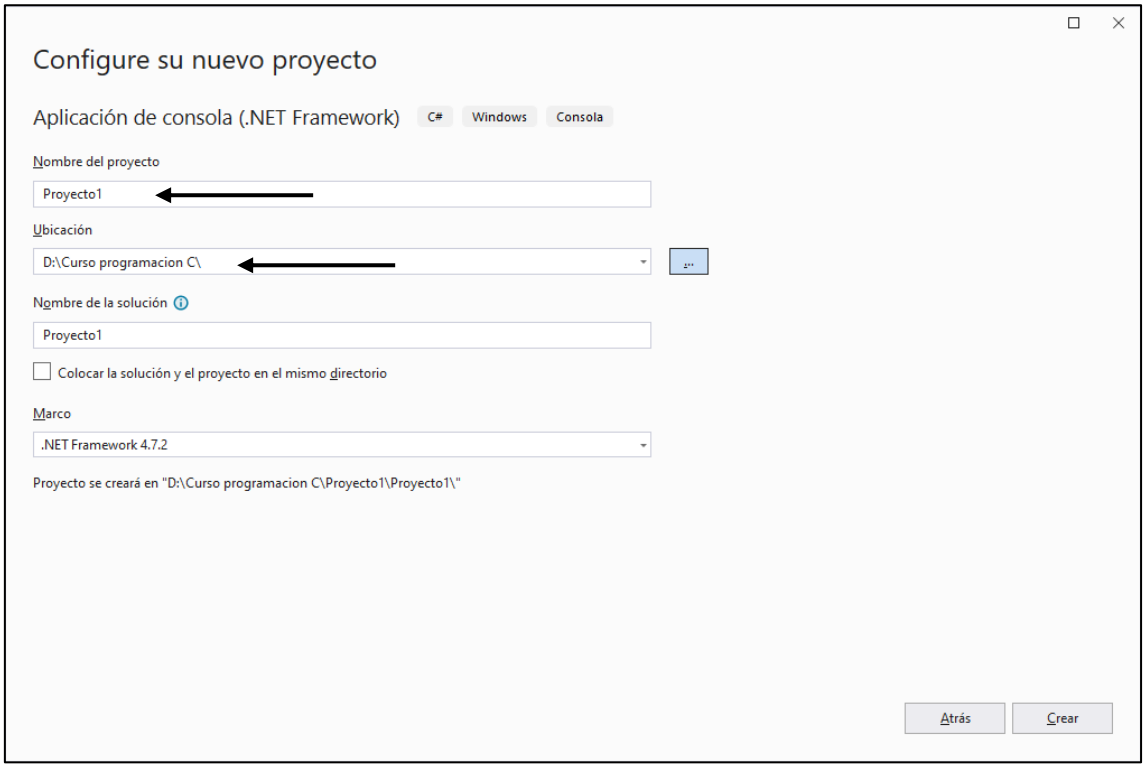
Seleccionaremos para programa en C# para el sistema operativo Windows y para Consola.



Seleccionaremos:

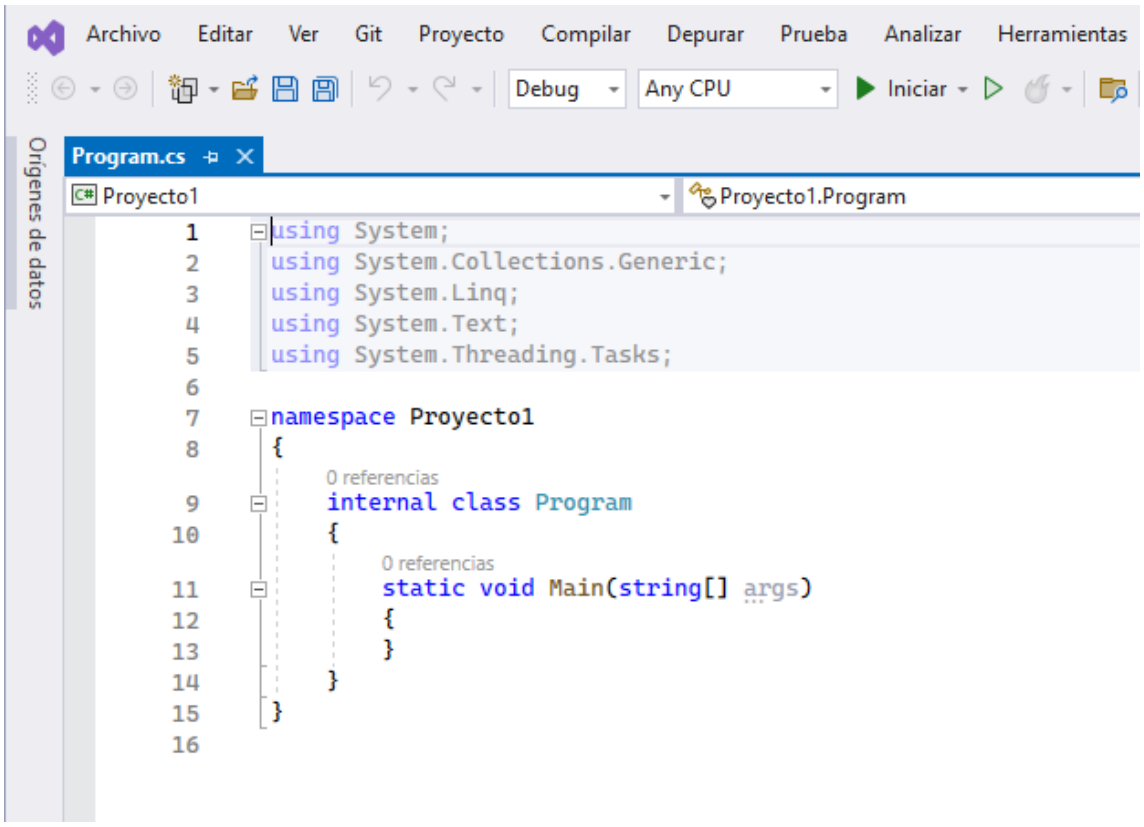


Le damos al botón siguiente.



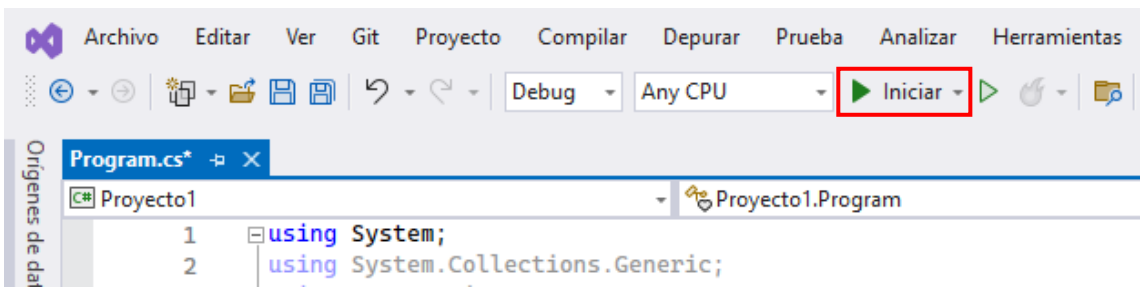
Tenemos que dar nombre al proyecto y la ubicación.

A continuación le damos al botón Crear.

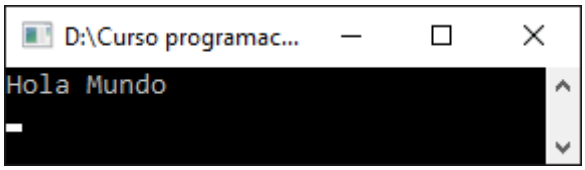


Agregamos el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto1
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            Console.WriteLine("Hola Mundo");
            Console.ReadKey();
        }
    }
}
```



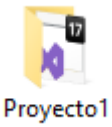
Le damos al botón Iniciar.



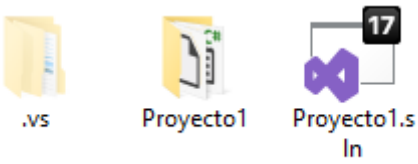
Cuando presionemos cualquier tecla esta ventana se cerrar gracias a la instrucción Console.ReadKey().

si has llegado hasta aquí esto significa que todo lo tenemos bien instalado.

Si vamos a la carpeta donde se ha guardado nuestro proyecto veremos:



Si la abrimos:

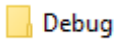


Sí abrimos Proyecto1:

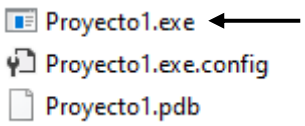
bin	27/06/2023 15:11	Carpeta de archivos	
obj	27/06/2023 15:11	Carpeta de archivos	
Properties	27/06/2023 15:11	Carpeta de archivos	
App.config	27/06/2023 15:11	XML Configuratio...	1 KB
Program.cs	27/06/2023 15:17	C# Source File	1 KB
Proyecto1.csproj	27/06/2023 15:11	C# Project file	3 KB

En Program.cs está el código fuente.

Si abrimos la carpeta bin:

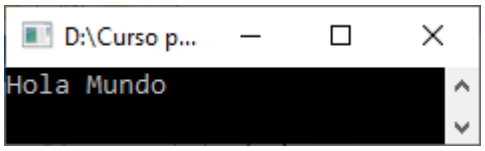


Encontramos la carpeta Debug y si la abrimos:



Encontramos el programa compilado.

Si hacemos doble clic en el se ejecuta el programa.

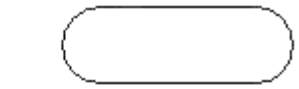


Capítulo 3.- Codificación del diagrama de flujo

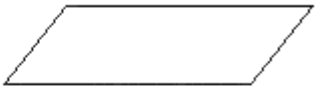
Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un ALGORITMO.

Las símbolos gráficos a utilizar para el plateo de diagramas de flujo son:



Inicio y fin del diagrama.

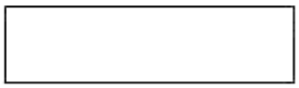


Entrada de Datos

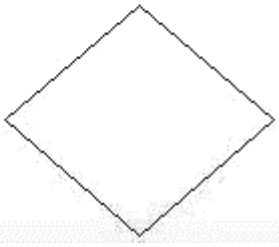
(Vamos a considerar que las entradas de datos se realizan siempre por el teclado de la computadora).

Salida de Datos

(Vamos a considerar que las salidas de datos se realizan siempre por la pantalla de la computadora).



Operación



Condición

Estos son los elementos esenciales que intervienen en el desarrollo de un diagrama de flujo.

Planteo de un problema utilizando diagrama de flujo

Para plantear un diagrama de flujo debemos tener muy claro el problema a resolver.

Ejemplo: Calcular el sueldo mensual de un operario conociendo la cantidad de horas trabajadas y el pago por hora.

Podemos indicar:

Datos conocidos:

Horas trabajadas en el mes.

Pago por hora.

Proceso:

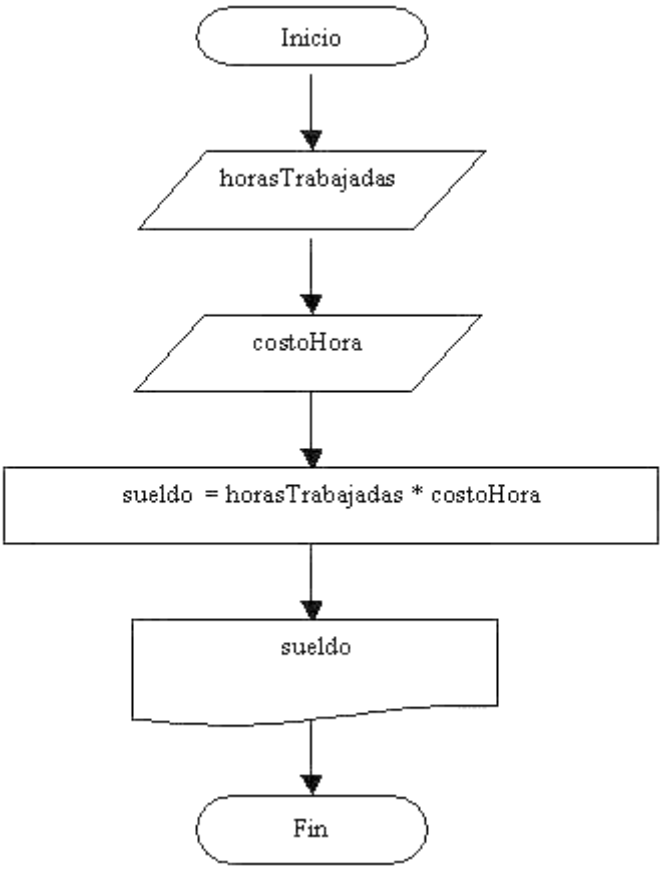
Cálculo del sueldo multiplicado la cantidad de horas por el pago por hora.

Información resultante:

Si hacemos un análisis todo problema está constituido por:

- Datos conocidos: Datos con los que se cuenta al plantear el problema.
- Proceso: Operaciones a realizar con los datos conocidos.
- Información resultante: Es la información que resuelve el problema.

Esta forma de expresar un problema identifica sus datos conocidos, procesos e información resultante puede llegar a ser engorrosa para problemas complejos donde hay muchos datos conocidos y procesos. Es por eso que resulta mucho más efectivo representar los pasos para la resolución del problema mediante un diagrama de flujo.



Resulta mucho más fácil entender un gráfico que un texto.

El diagrama de flujo nos identifica claramente los datos de entrada, operaciones y datos de salida.

En el ejemplo tenemos dos datos de entrada: horasTrabajadas y costoHora, en las entradas las representamos con un paralelogramo y hacemos un paralelogramo por cada dato de entrada.

La operación se representa con un rectángulo, debemos hacer un rectángulo por cada operación. A la salida la representamos con la hoja rota.

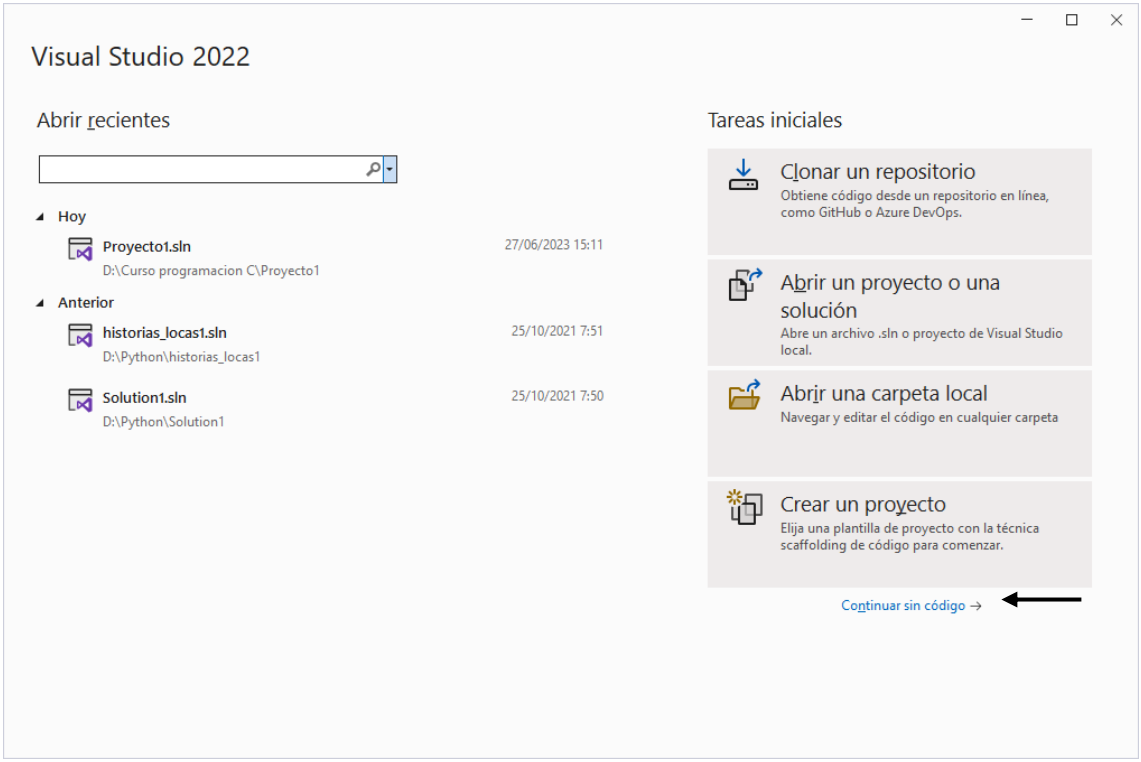
El diagrama de flujo nos da una idea ordenada de la ejecución de las actividades en tiempo, Primero cargamos los datos de entrada, luego hacemos las operaciones necesarias y por último mostramos los resultados.

Ahora debemos codificar el diagrama de flujo utilizando las instrucciones del lenguaje C#. Como hemos visto el entorno de programación de Visual C# nos creó un esqueleto básico sobre el cual continuaremos el programa.

A medida que avancemos en el curso veremos que significa una clase y namespace, cual es el objetivo del using, etc. por el momento nos centraremos donde codificamos nuestros diagramas de flujo.

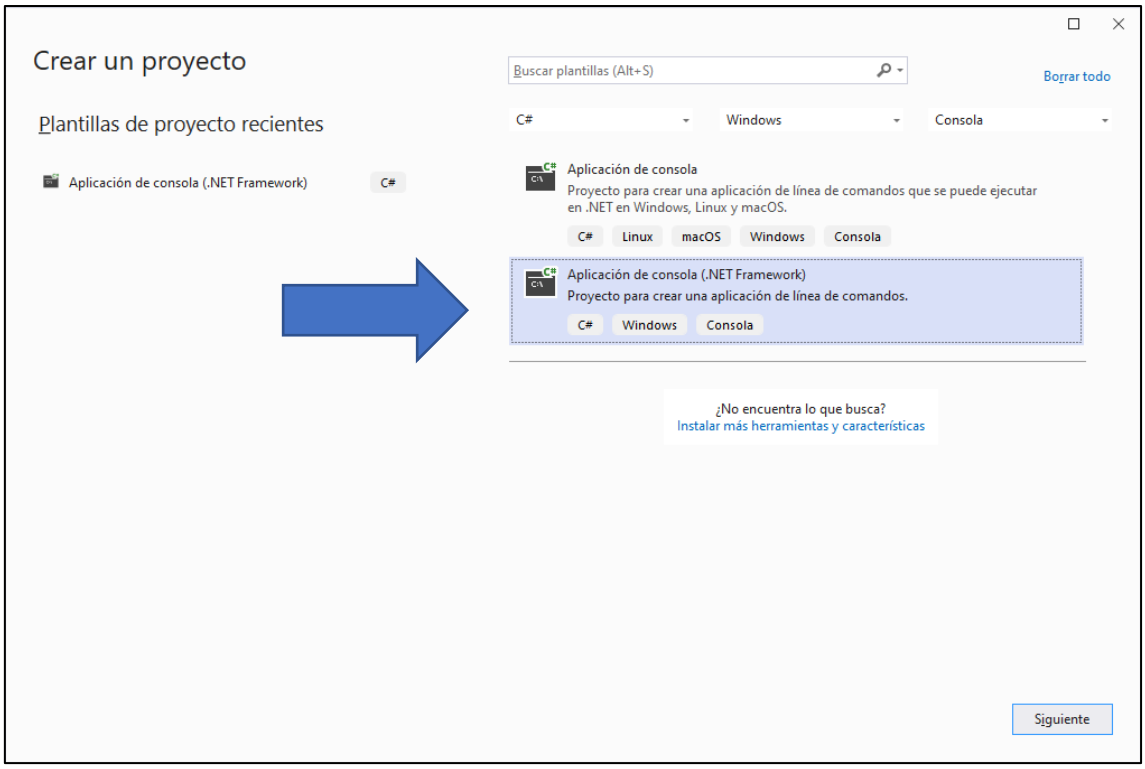
La codificación del diagrama de flujo la haremos dentro de la función Main (la función Main es la primera que se ejecuta al iniciarse un programa):

Vamos a programar:

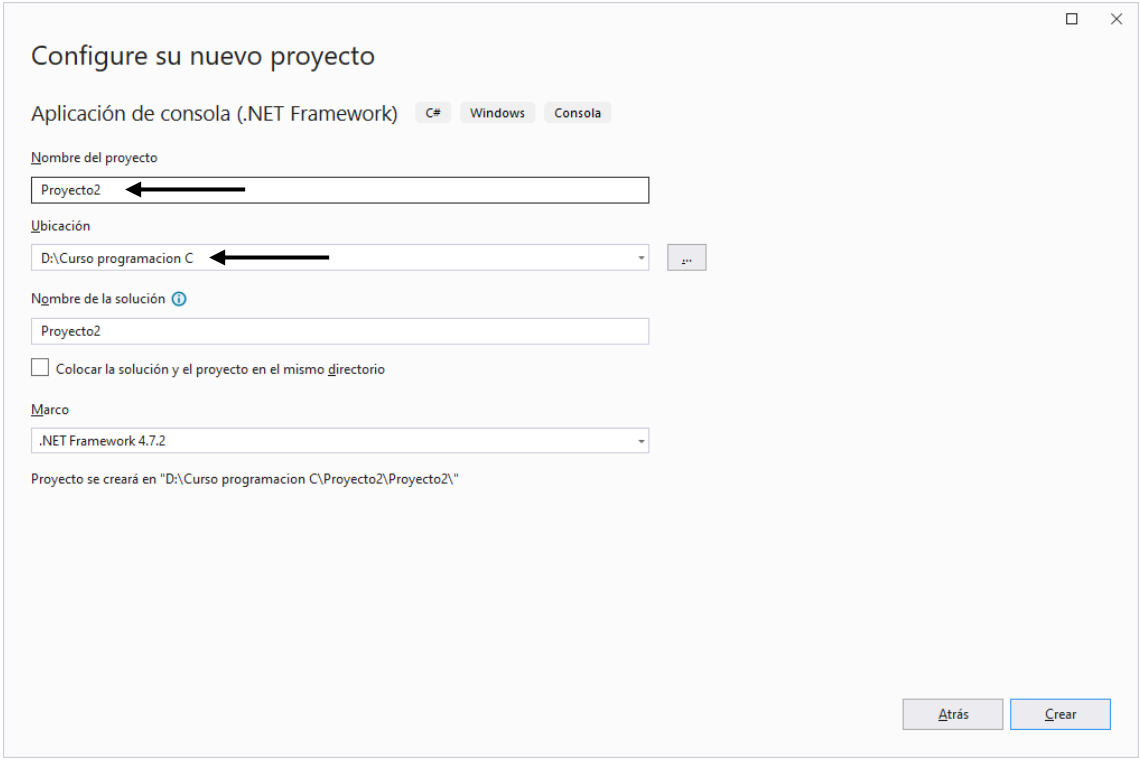


Le damos a continuar sin código.

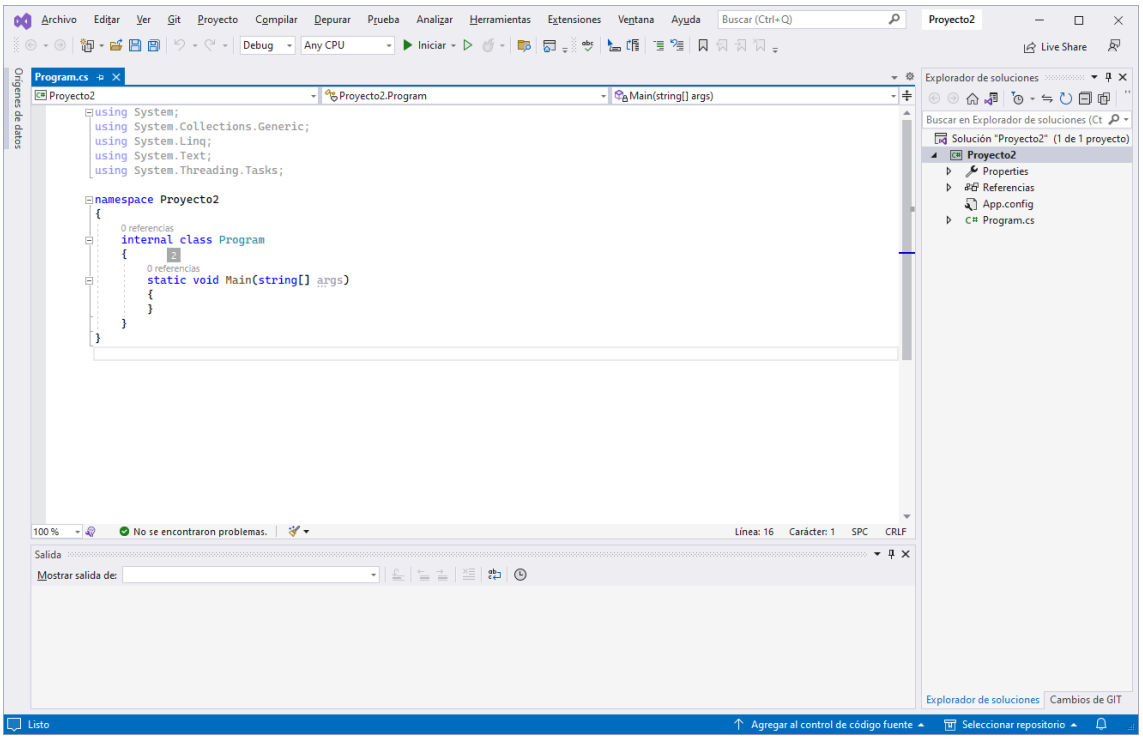
Del menú Archivo seleccionaremos Nuevo y de este Proyecto…



Seleccionaremos Aplicaciones de consola (.NET Framework), seguido de siguiente:



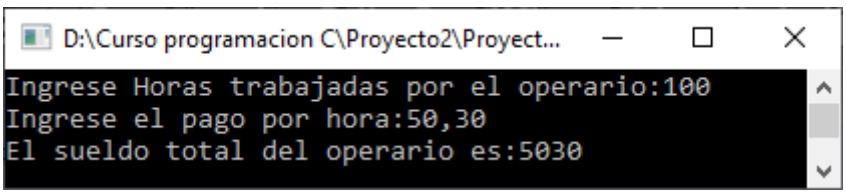
Como nombre del proyecto 'Proyecto2' y en ubicación seleccionaremos la carpeta donde guardamos los proyecto, seguido del botón crear.



Ahora vamos a escribir el programa:

```
namespace Proyecto2
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int horasTrabajadas;
            float costoHora;
            float sueldo;
            string linea;
            Console.Write("Ingrese Horas trabajadas por el operario:");
            linea = Console.ReadLine();
            horasTrabajadas = int.Parse(linea);
            Console.Write("Ingrese el pago por hora:");
            linea = Console.ReadLine();
            costoHora = float.Parse(linea);
            sueldo = horasTrabajadas * costoHora;
            Console.Write("El sueldo total del operario es:");
            Console.Write(sueldo);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Para probar el funcionamiento del programa debemos presionar el icono con un triangulo verde (o la tecla especial F5 o desde el menú elegir la opción "Depurar -> "Iniciar depuración").

Conceptos que deben de quedar claros:

1.- Por el momento haremos todo el algoritmo dentro de la función Main. Es decir el resto siempre lo crea el entorno del Visual C#.

2.- Si observamos el diagrama de flujo vemos que debemos definir tres variables:

(horasTrabajadas, costoHora y sueldo), aquí es donde debemos definir que tipos de datos se almacenarán en las mismas. La cantidad de horas normalmente será un valor entero (ej. 100 – 150 – 230 etc.), pero el costo de hora es muy común que sea un valor real (ej. 5.35 – 7.50 etc.) y como el sueldo resulta de multiplicar las horas trabajadas por el costo hora el mismo deberá ser real.

La definición de las variables la hacemos en el Main:

```
int horasTrabajadas;
float costoHora;
float sueldo;
```

Utilizamos la palabra clave int para definir variables enteras (en C# las palabras claves debe ir obligatoriamente en minúsculas, sino se produce un error sintáctico).Luego de la palabra clave debemos indicar el nombre de la variable, por ejemplo: horasTrabajadas (se propone que el nombre comience en minúsculas y en caso de estás constituidas por dos o más palabras deber ir en mayúscula el primer carácter (un nombre de variable no puede tener espacios en blanco, empezar con un número, ni tampoco utilizar caracteres especiales).

Debemos buscar siempre nombres de variables que nos indiquen que almacenan (no es conveniente llamar a nombres de variables con letras individuales).

3.- Para mostrar mensajes en la pantalla utilizamos el objeto "Console":

```
Console.Write("Ingrese Horas trabajadas por el operario:");
```

Con esta sintaxis todo lo que se encuentra contenido entre comillas aparecerá exactamente en la ventana de la "Console".

Si disponemos una variable:

```
Console.Write(sueldo);
```

Aparecerá el contenido de la variable. Es decir el valor almacenado en la variable sueldo y no el mensaje "sueldo".

4.- Para hacer la entrada de datos por teclado en C# se complica. Debemos definir una variable de tipo string que le llamaremos línea:

```
string linea;
```

Luego cada vez que necesitemos ingresar por teclado un conjunto de caracteres utilizaremos la función ReadLine del objeto Console con la siguiente sintaxis:

```
linea = Console.ReadLine();
```

Un segundo paso es copiar el contenido de la variable línea en una variable de tipo int:

```
horasTrabajadas = int.Parse(linea);
```

O una variable de tipo float:

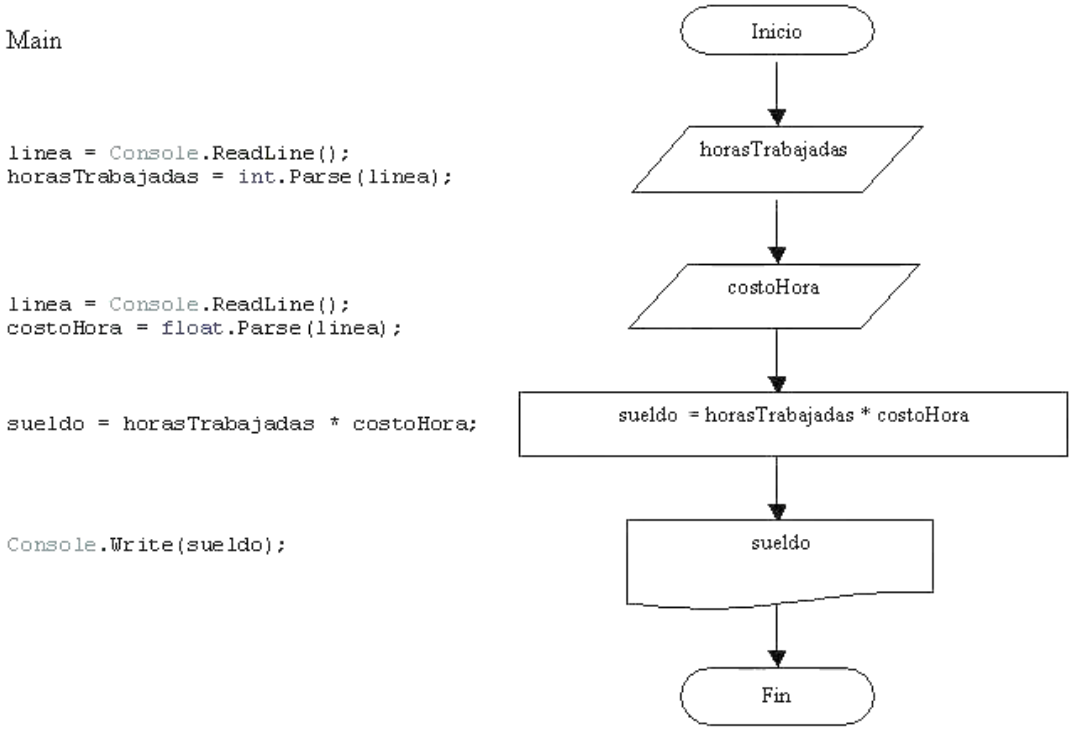
```
costoHora = float.Parse(linea);
```

La variable línea almacena temporalmente los datos que ingresa el operador del programa, para luego copiarse a la variable respectiva (como vemos si queremos convertir un string a tipo de dato entero utilizamos la función Parse del objeto int (int.Parse)).

Las operaciones que indicamos en el diagrama de flujo mediante la figura rectángulo la codificamos tal cual:

```
sueldo = horasTrabajadas * costoHora;
```

Podemos ver una relación entre las instrucciones que debemos utilizar para cada símbolo del diagrama de flujo.



En el diagrama de flujo no indicamos la definición de variables.

```
int horasTrabajadas;
float costoHora;
float sueldo;
string linea;
```

No representamos con símbolos los mensajes a mostrar previo a la carga de datos por teclado:

```
Console.Write("Ingrese Horas trabajadas por el operario:");
```

Como hemos visto hasta ahora hay muchas partes de nuestro código que no entendemos pero son indispensables para la implementación de nuestros programas, a medida que avancemos con el curso muchos de estos conceptos se irán aclarando.

Capítulo 4.- Errores sintácticos y lógicos

Confeccionaremos un problema y agregaremos adrede una serie de errores tipográficos. Este tipo de errores siempre son detectados por el COMPILADOR, antes de ejecutar el programa.

A los errores tipográficos, como por ejemplo la falta de puntos y coma, nombres de variable incorrectas, falta de paréntesis, palabras clave mal escritas, etc. los llamamos errores SINTACTICOS.

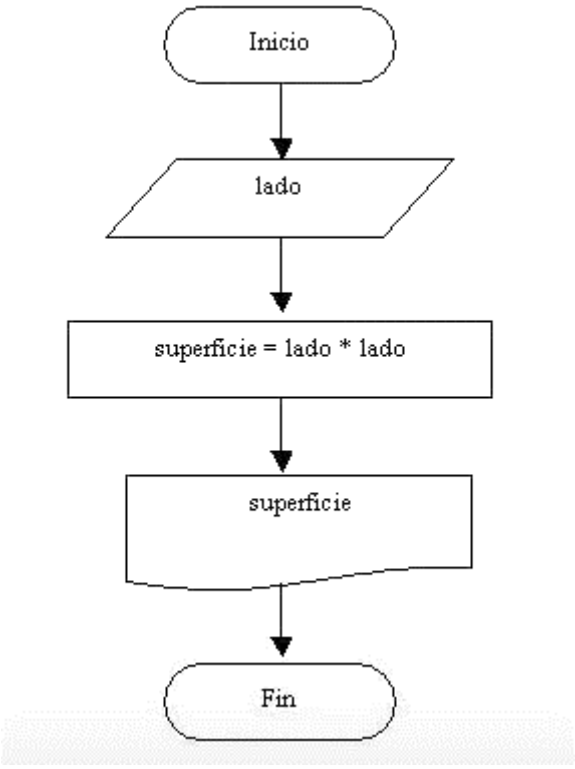
Un programa no se puede ejecutar sin corregir absolutamente todos los errores sintácticos.

Este otro tipo de errores llamador ERRORES LOGICOS. Este tipo de errores es programas grandes (miles de líneas) son más difíciles de localizar. Por ejemplo un programa que permite hacer la facturación pero la salida de datos por impresora es incorrecta.

Problema

Hallar la superficie de un cuadrado conociendo el valor de un lado.

Diagrama de flujo:



Copia el siguiente código y corrige los posibles errores:

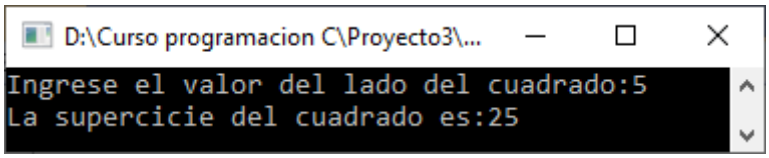
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto3
{
    internal class Program
    {
        static void Main(string[] args)
```

```
        {
            int lado;
            int Superficie;
            string linea;
            Console.Write("Ingrese el valor del lado del cuadrado:");
            linea = Console.ReadLine();
            lado = int.Parse(linea)
            superficie = lado * Lado;
            Console.Write("La supercicie del cuadrado es:");
            Console.write(superficie);
            Console.Readkey();
        }
    }
}
```

Solución:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto3
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int lado;
            int superficie;
            string linea;
            Console.Write("Ingrese el valor del lado del cuadrado:");
            linea = Console.ReadLine();
            lado = int.Parse(linea);
            superficie = lado * lado;
            Console.Write("La supercicie del cuadrado es:");
            Console.Write(superficie);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

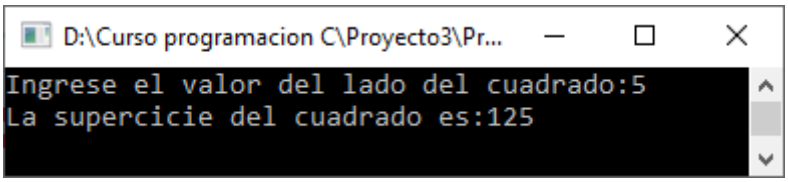


Ejemplo de un error lógico:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto3
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int lado;
            int superficie;
            string linea;
            Console.Write("Ingrese el valor del lado del cuadrado:");
            linea = Console.ReadLine();
            lado = int.Parse(linea);
            superficie = lado * lado * lado;
            Console.Write("La supercicie del cuadrado es:");
            Console.Write(superficie);
            Console.ReadKey();
        }
    }
}
```

Este error el compilador no lo detecta, lo veremos cuando ejecutemos el programa y el resultado del mismo nos da un dato incorrecto.

Vamos a ejecutar:



Capítulo 5.- Estructura de programación secuencial – 1

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial.

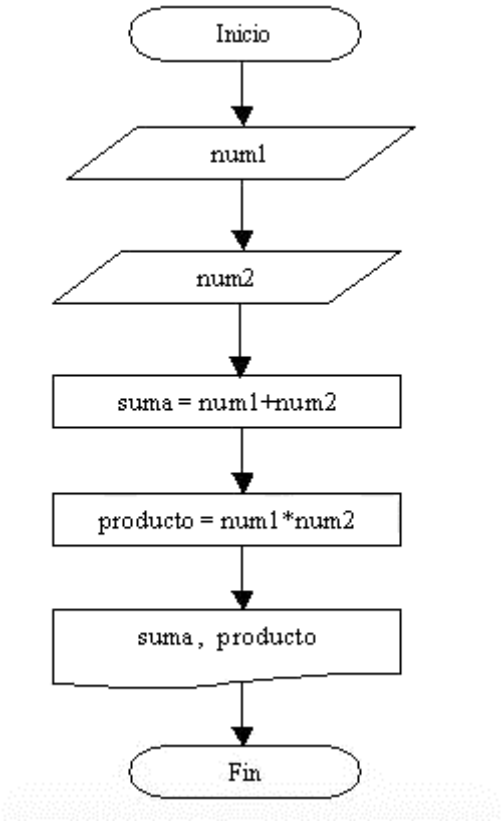
Los problemas diagramados y codificados previamente emplean solo estructuras secuenciales.

La programación requiere una práctica ininterrumpida de diagramación y codificación de problemas.

Problema

Realiza la carga de dos números enteros por teclado e imprimir su suma y su producto.

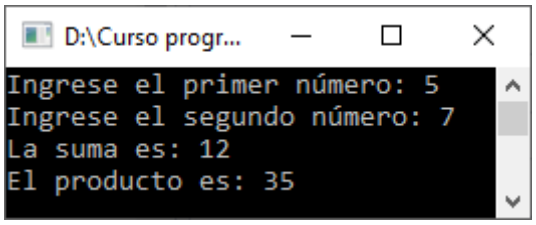
Diagrama de flujo:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto4
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
```

```
            int num1, num2;
            int suma, producto;
            string linea;
            Console.Write("Ingrese el primer número: ");
            linea = Console.ReadLine();
            num1 = int.Parse(linea);
            Console.Write("Ingrese el segundo número: ");
            linea = Console.ReadLine();
            num2 = int.Parse(linea);
            suma = num1 + num2;
            producto = num1 * num2;
            Console.Write("La suma es: ");
            Console.WriteLine(suma);
            Console.Write("El producto es: ");
            Console.Write(producto);
            Console.ReadKey();
        }
    }
}
```

si ejecutamos este será el resultado:



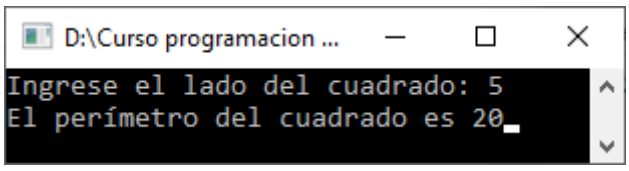
Capítulo 6.- Estructura de programación secuencial – 2

Problema propuesto

Realizar la carga del lado de un cuadrado, mostrar por pantalla el perímetro del mismo (El perímetro de un cuadrado se calcula multiplicando el valor del lado por cuatro).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto5
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int lado, perimetro;
            string linea;
            Console.Write("Ingrese el lado del cuadrado: ");
            linea = Console.ReadLine();
            lado = int.Parse(linea);
            perimetro = lado * 4;
            Console.Write("El perímetro del cuadrado es ");
            Console.Write(perimetro);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



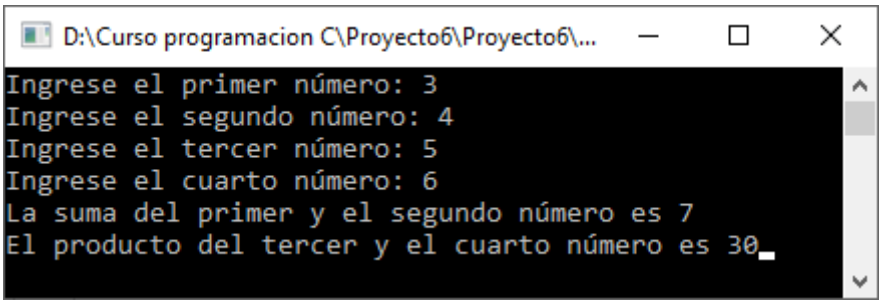
Capítulo 7.- Estructura de programación secuencial – 3

Problema propuesto

Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primero y el producto del tercer y el cuarto.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto6
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3, num4;
            int suma, producto;
            string linea;
            Console.Write("Ingrese el primer número: ");
            linea = Console.ReadLine();
            num1 = int.Parse(linea);
            Console.Write("Ingrese el segundo número: ");
            linea = Console.ReadLine();
            num2 = int.Parse(linea);
            Console.Write("Ingrese el tercer número: ");
            linea = Console.ReadLine();
            num3 = int.Parse(linea);
            Console.Write("Ingrese el cuarto número: ");
            linea = Console.ReadLine();
            num4 = int.Parse(linea);
            suma = num1 + num2;
            producto = num3 * num4;
            Console.Write("La suma del primer y el segundo número es ");
            Console.WriteLine(suma);
            Console.Write("El producto del tercer y el cuarto número es ");
            Console.Write(producto);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 8.- Estructura de programación secuencial – 4

Problema propuesto

Realizar un programa que lea cuatro valores numéricos e informar su suma y su promedio.

```
namespace Proyecto7
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3, num4;
            int suma, promedio;
            string linea;
            Console.Write("Ingrese el primer número: ");
            linea = Console.ReadLine();
            num1 = int.Parse(linea);
            Console.Write("Ingrese el segundo número: ");
            linea = Console.ReadLine();
            num2 = int.Parse(linea);
            Console.Write("Ingrese el tercer número: ");
            linea = Console.ReadLine();
            num3 = int.Parse(linea);
            Console.Write("Ingrese el cuarto número: ");
            linea = Console.ReadLine();
            num4 = int.Parse(linea);
            suma = num1 + num2 + num3 + num4;
            promedio = suma / 4;
            Console.Write("La suma de los cuatro valores es ");
            Console.WriteLine(suma);
            Console.Write("Su promedio es ");
            Console.Write(promedio);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

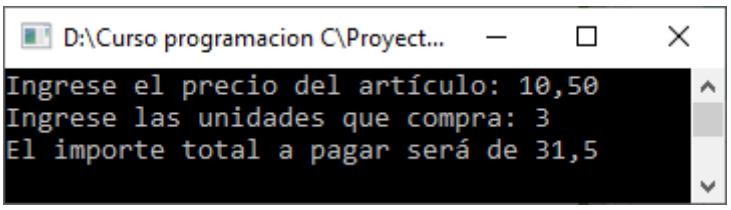
```
D:\Curso programacion C\Proyecto7\...
Ingrese el primer número: 1
Ingrese el segundo número: 2
Ingrese el tercer número: 3
Ingrese el cuarto número: 4
La suma de los cuatro valores es 10
Su promedio es 2
```

Capítulo 9.- Estructura de programación secuencial – 5

Se debe desarrollar un programa que pida el ingreso del precio de un artículo y la cantidad que lleva el cliente. Mostrar lo que debe abonar el comprador.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto8
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int cantidad;
            float precio, total;
            string linea;
            Console.Write("Ingrese el precio del artículo: ");
            linea = Console.ReadLine();
            precio = float.Parse(linea);
            Console.Write("Ingrese las unidades que compra: ");
            linea = Console.ReadLine();
            cantidad = int.Parse(linea);
            total = precio * cantidad;
            Console.Write("El importe total a pagar será de ");
            Console.Write(total);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 10.- Estructuras condicionales simples y compuestas – 1

No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.

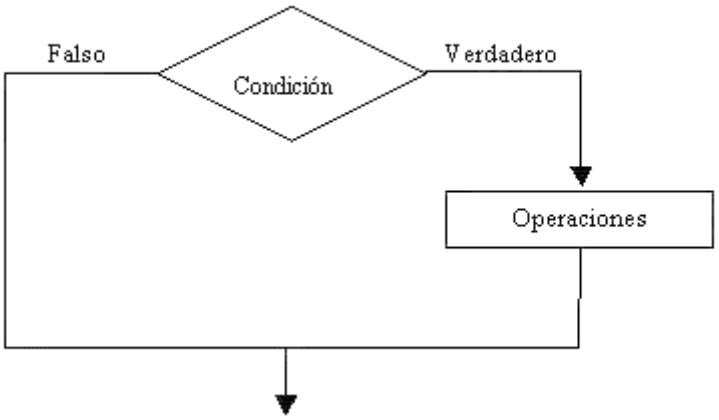
En nuestra vida diaria se nos presentan situaciones donde debemos decidir.
¿Elijo la carrera A o la carrera B?
¿Me pongo un pantalón?
Para ir al trabajo, ¿elijo el camino A o el camino B?
Al cursar una carrera. ¿elijo el turno mañana, tarde o noche?

Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.

Estructura condicional simple

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna.

Representación gráfica:



Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición es verdadera se sigue el camino del verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda.

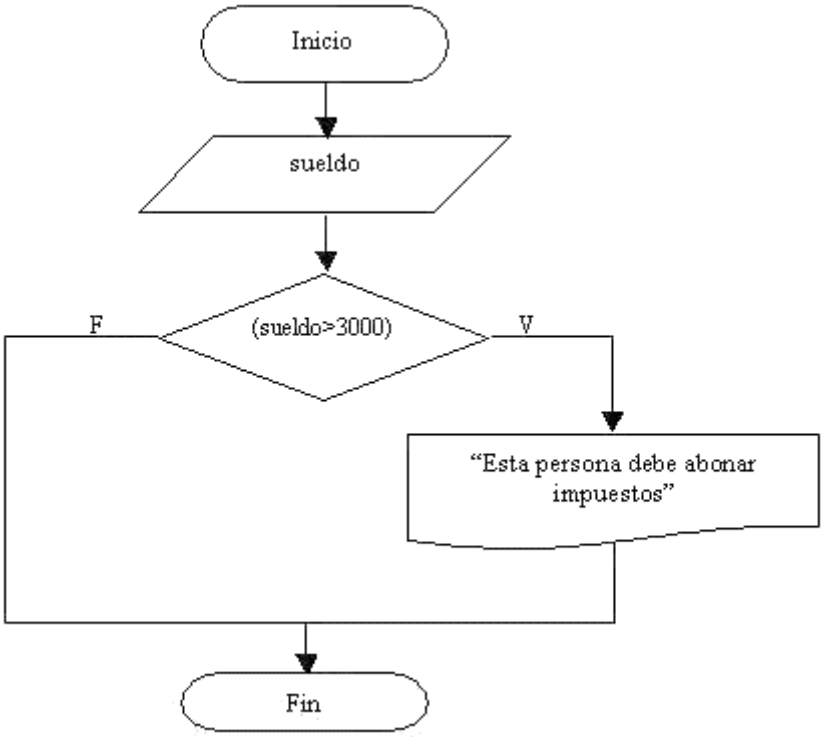
Se trata de una estructura CONDICIONAL SIMPLE porque por el camino verdadero hay actividad y por el camino falso no hay actividad.

Por el camino del verdadero pueden existir varias operaciones, entradas, salidas, inclusive ya veremos que pueden haber otras secuencias condicionales.

Problema

Ingresar el sueldo de una persona, si supera los 3000 pesos mostrar un mensaje en pantalla indicando que debe abonar impuestos.

Diagrama de flujo



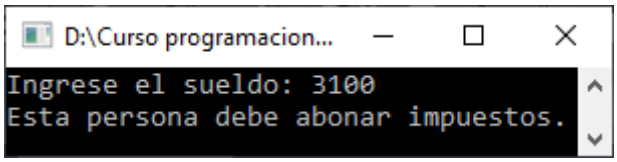
Podemos observar lo siguiente: Siempre que hacemos la carga del sueldo, pero si el sueldo que ingresamos supera 3000 pesos se mostrará por pantalla el mensaje "Esta persona debe abonar impuestos", en caso que la persona cobre 300 o menos no aparecerá nada por pantalla.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto9
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            float sueldo;
            string linea;
            Console.Write("Ingrese el sueldo: ");
            linea = Console.ReadLine();
            sueldo = float.Parse(linea);
            if (sueldo > 3000)
            {
                Console.Write("Esta persona debe abonar impuestos.");
            }
            Console.ReadKey();
        }
    }
}
```

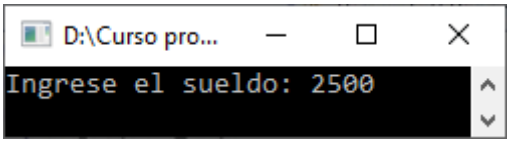
La palabra clave "if" indica que estamos en presencia de una estructura condicional; seguidamente disponemos la condición entre paréntesis. Por último encerrada entre llaves las instrucciones de la rama del verdadero.

En necesario que las instrucciones a ejecutar en caso que la condición sea verdadera estén encerradas entre llaves { }, con ellas marcamos el comienzo y el fin del bloque verdadero.

Ejecutamos el programa e ingresamos un sueldo superior a 3000 pesos. Podemos observar como aparece en pantalla el mensaje "Esta persona debe abonar impuestos", ya que la condición del if es verdadera.



Volvemos a ejecutar el programa y carguemos un sueldo menor o igual a 3000 pesos. No debe mostrar el mensaje.

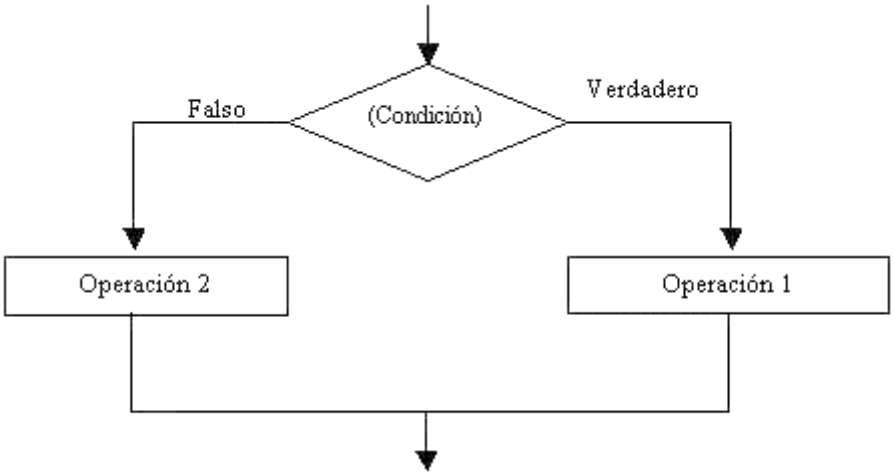


Capítulo 11.- Estructuras condicionales simples y compuestas – 2

Estructura condicional compuesta

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso. NUNCA ser realizan las actividades de las dos ramas.

Representación gráfica:

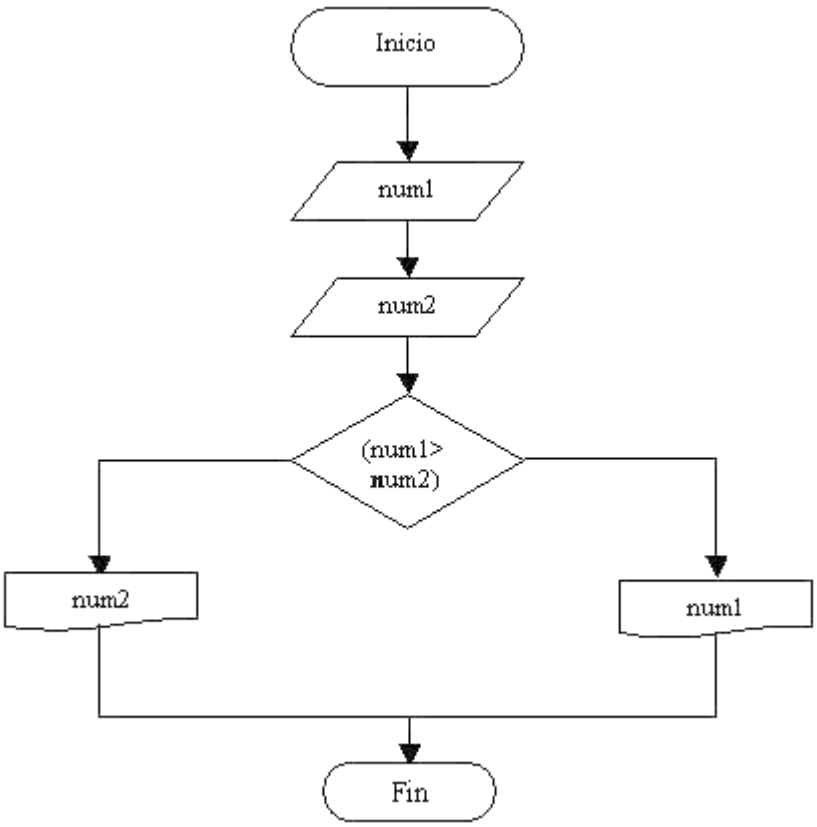


En una estructura condicional compuesta tenemos entradas, salidas, operaciones, tanto por la rama del verdadero como por la rama del falso.

Problema

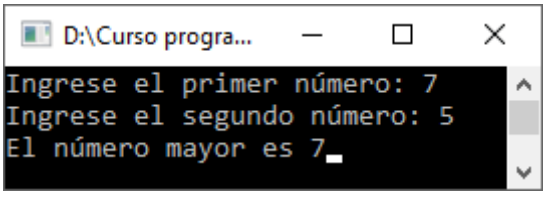
Realizar un programa que solicite ingresar dos números distintos y muestre por pantalla el mayor de ellos.

Diagrama de flujo

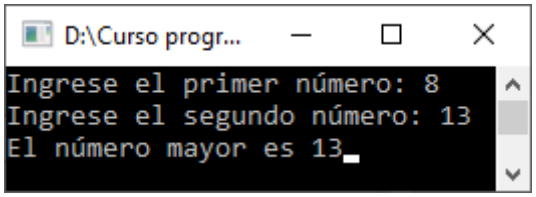


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto10
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2;
            string linea;
            Console.Write("Ingrese el primer número: ");
            linea = Console.ReadLine();
            num1 = int.Parse(linea);
            Console.Write("Ingrese el segundo número: ");
            linea = Console.ReadLine();
            num2 = int.Parse(linea);
            if(num1 > num2)
            {
                Console.Write("El número mayor es ");
                Console.Write(num1);
            }
            else
            {
                Console.Write("El número mayor es ");
                Console.Write(num2);
            }
            Console.ReadKey();
        }
    }
}
```

Ejecutamos introduciendo los siguientes valores:



Ejecutamos de nuevo introduciendo un valor mayor en el segundo número:



Operadores

En una condición deben disponer únicamente variables, valores constantes y operadores relacionales.

Operadores relacionales

```
>  (mayor)
<  (menor)
>= (mayor o igual)
<= (menor o igual)
== (igual)
!= (distinto)
```

Operadores matemáticos

```
+ (más)
- (menos)
* (producto)
/ (división)
% (resto de una división)  Ej.:  x=13%5;  {se guarda 3}
```

Hay que tener en cuenta que al disponer una condición debemos seleccionar que operador relacional se adapta a la pregunta.

Ejemplos:

- Se ingresa un número multiplicado por 10 si es distinto a 0 (!=).
- Se ingresan dos números mostrar una advertencia si son iguales (==).

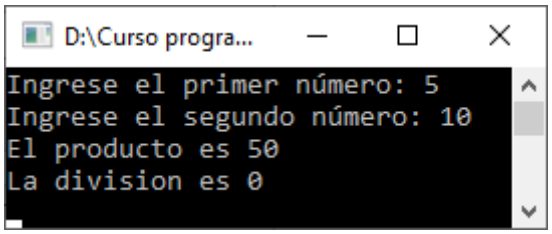
Capítulo 12.- Estructuras condicionales simples y compuestas – 3

Problema propuesto

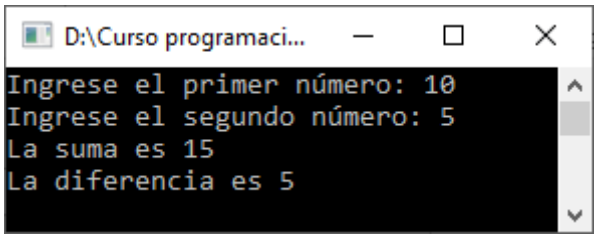
Realizar un programa que lea por teclado dos números, si el primero es mayor al segundo informar su suma y diferencia, en caso contrario informar el producto y la división del primero con respecto al segundo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto11
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2;
            int suma, diferencia, producto, division;
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            if(num1 > num2)
            {
                suma = num1 + num2;
                diferencia = num1 - num2;
                Console.Write("La suma es ");
                Console.WriteLine(suma);
                Console.Write("La diferencia es ");
                Console.WriteLine(diferencia);
            }
            else
            {
                producto = num1 * num2;
                division = num1 / num2;
                Console.Write("El producto es ");
                Console.WriteLine(producto);
                Console.Write("La division es ");
                Console.WriteLine(division);
            }
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar introduciendo los valores 5 y 10:



Ejecutamos de nuevo introduciendo los valores 10 y 5.



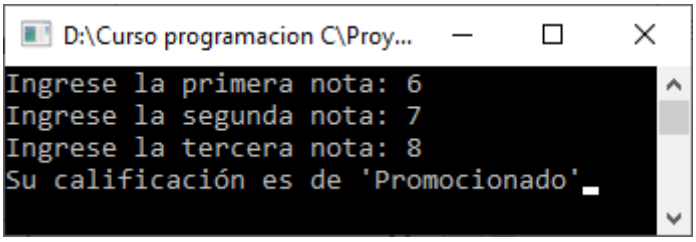
Capítulo 13.- Estructuras condicionales simples y compuestas – 4

Problema propuesto

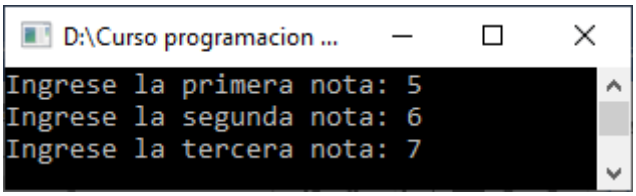
Se ingresan tres notas de un alumno, si el promedio es mayor o igual a siete mostrar un mensaje "Promocionado".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto12
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int nota1, nota2, nota3;
            int suma, promedio;
            string line;
            Console.Write("Ingrese la primera nota: ");
            line = Console.ReadLine();
            nota1 = int.Parse(line);
            Console.Write("Ingrese la segunda nota: ");
            line = Console.ReadLine();
            nota2 = int.Parse(line);
            Console.Write("Ingrese la tercera nota: ");
            line = Console.ReadLine();
            nota3 = int.Parse(line);
            suma = nota1 + nota2 + nota3;
            promedio = suma / 3;
            if (promedio >= 7)
            {
                Console.Write("Su calificación es de 'Promocionado'");
            }
            Console.ReadKey();

        }
    }
}
```

Ejecutamos e introducimos notas para que el promedio sea igual a mayor a 7.



Ejecutamos de nuevo e introducimos notas para que el promedio será inferior a 7.



El promedio también lo podemos obtener utilizando solo la variable promedio y sin la variable suma:

```
promedio = (nota1 + nota2 + nota3) / 3;
```

Capítulo 14.- Estructuras condicionales simples y compuestas – 5

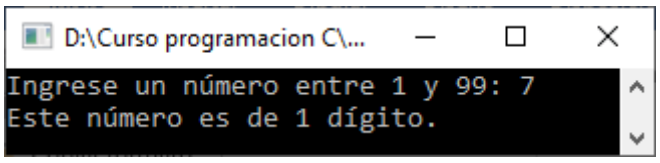
Problema propuesto

Se ingresa por teclado un número positivo de uno o dos dígitos (1..99) mostrar un mensaje indicando si el número tiene uno o dos dígitos.

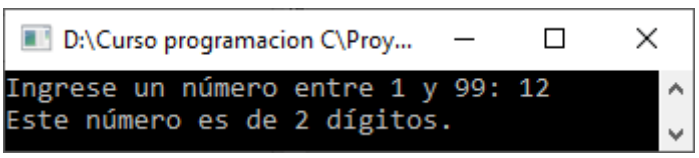
(Tener en cuenta que condición debe cumplirse para tener dos dígitos, un número entero).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto13
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num;
            string line;
            Console.Write("Ingrese un número entre 1 y 99: ");
            line = Console.ReadLine();
            num = int.Parse(line);
            if (num < 10)
            {
                Console.Write("Este número es de 1 dígito.");
            }
            else
            {
                Console.Write("Este número es de 2 dígitos.");
            }
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar introduciendo el valor 7:

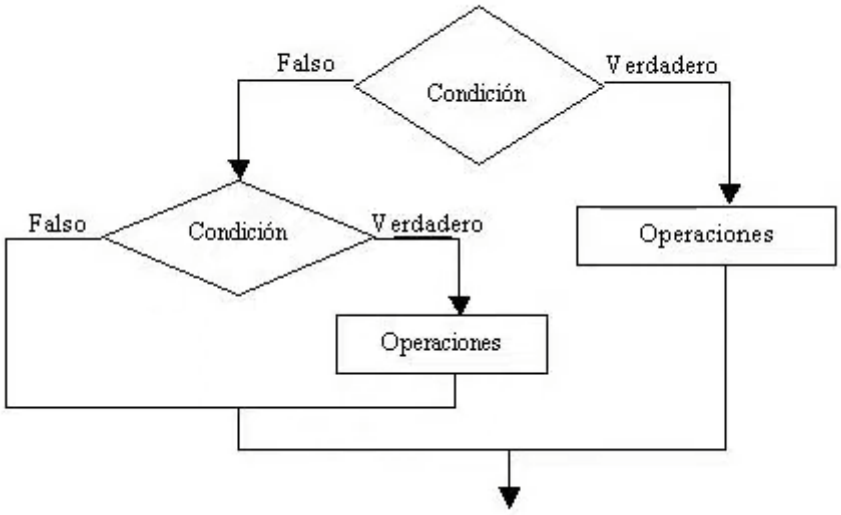


Vamos a ejecutar de nuevo introduciendo el valor 12:



Capítulo 15.- Estructuras condicionales anidadas – 1

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o falso de una estructura condicional hay otra estructura condicional.



El diagrama de flujo que se presenta contiene dos estructuras condicionales. La principal se trata de una estructura condicional compuesta y la segunda es una estructura condicional simple y está contenida por la rama del falso de la primera estructura.

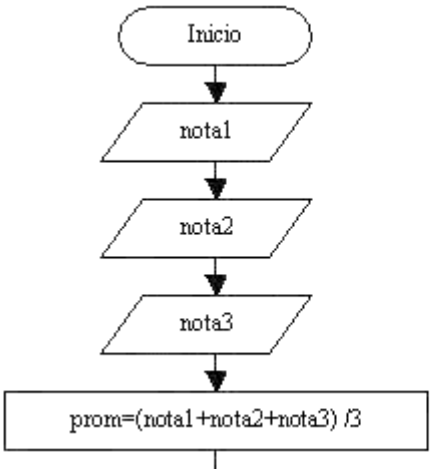
Es común que se presenten condiciones anidadas aún más complejas.

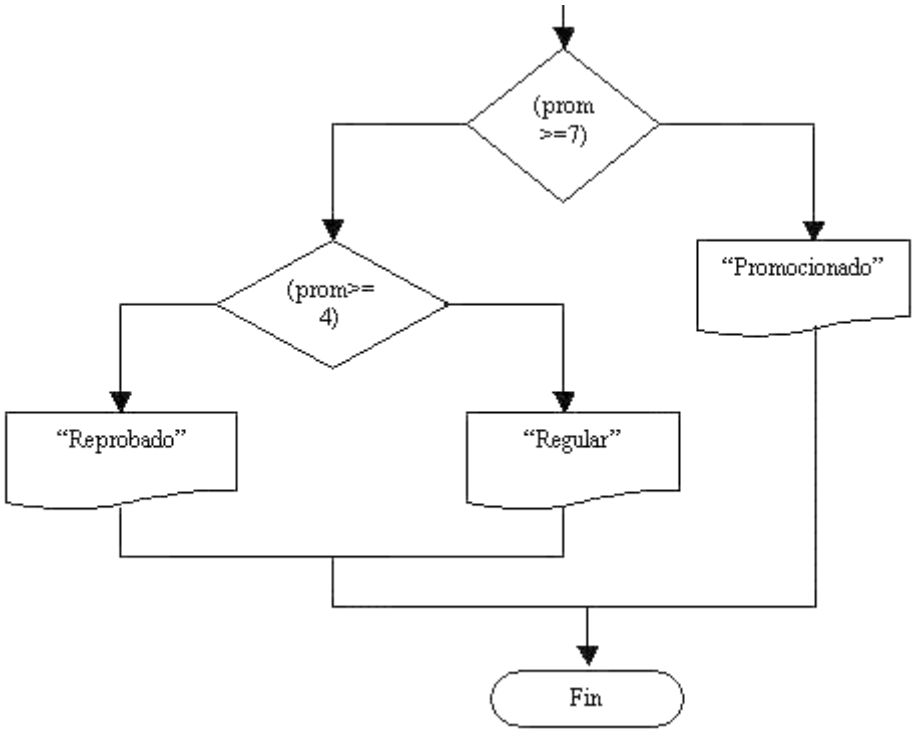
Problema

Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:

- Si el promedio es >=7 mostrar "Promocionado".
- Si el promedio es >=4 y <7 mostrar "Regular".
- Si el promedio es <4 mostrar "Reprobado".

Diagrama de flujo

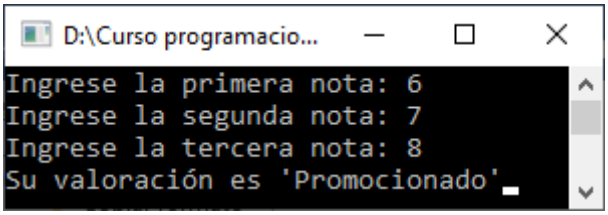




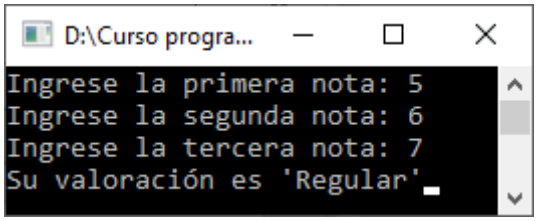

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto14
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int nota1, nota2, nota3;
            int promedio;
            string line;
            Console.Write("Ingrese la primera nota: ");
            line = Console.ReadLine();
            nota1 = int.Parse(line);
            Console.Write("Ingrese la segunda nota: ");
            line = Console.ReadLine();
            nota2 = int.Parse(line);
            Console.Write("Ingrese la tercera nota: ");
            line = Console.ReadLine();
            nota3 = int.Parse(line);
            promedio = (nota1 + nota2 + nota3) / 3;
            if (promedio >= 7)
            {
                Console.Write("Su valoración es 'Promocionado'");
            }
```

```
            else
            {
                if (promedio >= 4)
                {
                    Console.Write("Su valoración es 'Regular'");
                }
                else
                {
                    Console.Write("Su valoración es 'Reprobado'");
                }
            }
            Console.ReadKey();
        }
    }
}
```

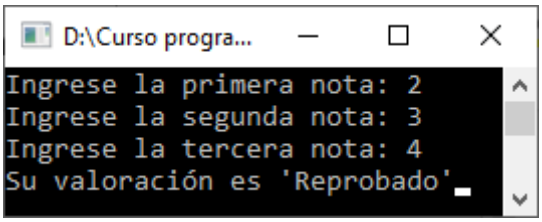
Vamos a ejecutar e introducir las notas 6, 7 y 8:



Ejecutamos de nuevo e introducimos las notas 5, 6 y 7.



Ejecutamos otra vez e introducimos las notas 2, 3 y 4.



Analicemos el siguiente diagrama. Se ingresa tres valores por teclado que representan las notas de un alumno, se obtiene el promedio sumando los tres valores y dividiendo por 3 dicho resultado (Tener en cuenta que si el resultado es un valor real solo se almacena la parte entera).

Primeramente preguntamos si el promedio es superior o igual a 7, en caso afirmativo va por la rama del verdadero de la estructura condicional mostrando un mensaje que indica "Promocionado" (con comillas indicamos un texto que debe imprimirse en pantalla).

En caso que la condición nos de falso, por la rama del falso aparece otra estructura condicional, porque todavía debemos averiguar si el promedio del alumno es superior o igual a cuatro o inferior a cuatro.

Estamos en presencia de dos estructuras compuestas.

Codifiquemos y ejecutemos este programa. Al correr el programa deberá solicitar por teclado la carga de tres notas y mostrarnos un mensaje según el promedio de las mismas.

Podemos definir un conjunto de variables del mismo tipo en una misma línea.

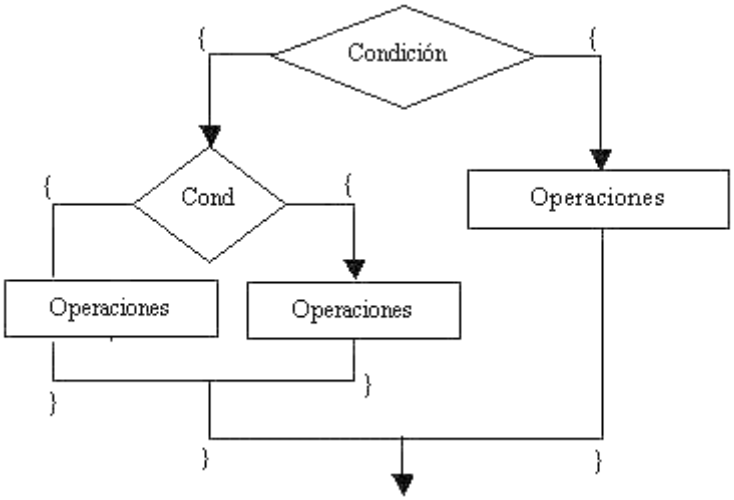
```
int nota1,nota2,nota3;
```

Esto no es obligación pero a veces, por estar relacionadas, conviene.

A la codificación del if anidado podemos observar por el else del primer if.

Para no tener problemas (olvidarnos) con las llaves de apertura y cerrado podemos ver la siguiente regla:

Cada vértice representa una llave de apertura y una de cierre.



Capítulo 16.- Estructuras condicionales anidadas – 2

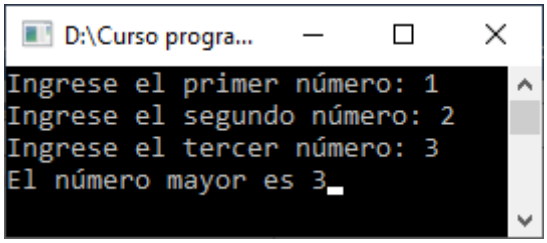
Problema propuesto

Se cargan por teclado tres números distintos. Mostrar por pantalla el mayor de ellos.

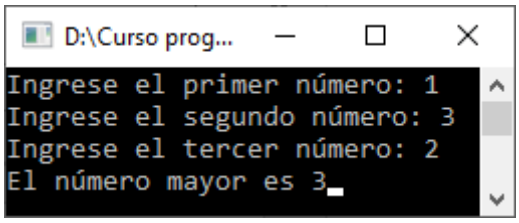
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto15
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3;
            String line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
            if (num1>num2)
            {
                if(num1>num3)
                {
                    Console.Write("El número mayor es ");
                    Console.Write(num1);
                }
                else
                {
                    Console.Write("El número mayor es ");
                    Console.Write(num3);
                }
            }
            else
            {
                if (num2>num3)
                {
                    Console.Write("El número mayor es ");
                    Console.Write(num2);
                }
                else
```

```
                {
                    Console.Write("El número mayor es ");
                    Console.Write(num3);
                }
            }
            Console.ReadKey();
        }
    }
}
```

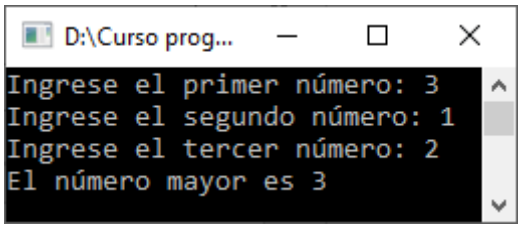
Vamos a ejecutar introduciendo los valores 1, 2 y 3.



Ejecutamos de nuevo introduciendo los valores 1, 3 y 2:



Ejecutamos de nuevo introduciendo los valores 3, 1 y 2:



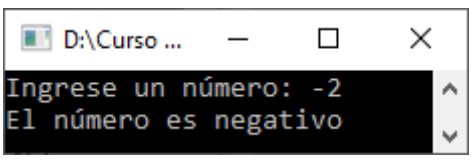
Capítulo 17.- Estructuras condicionales anidadas – 3

Problema propuesto

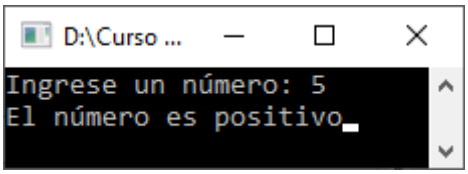
Se ingresa por teclado un valor entero, mostrar una leyenda que indique si el número es positivo, nulo o negativo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto16
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num;
            string line;
            Console.Write("Ingrese un número: ");
            line = Console.ReadLine();
            num = int.Parse(line);
            if(num == 0)
            {
                Console.Write("El número es nulo");
            }
            else
            {
                if(num>0)
                {
                    Console.Write("El número es positivo");
                }
                else
                {
                    Console.Write("El número es negativo");
                }
            }
            Console.ReadKey();
        }
    }
}
```

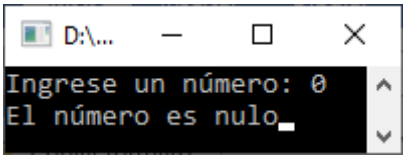
Ejecutamos e introducimos un número negativo:



Ejecutamos e introducimos un número positivo:



Ejecutamos de nuevo e introducimos el valor 0:



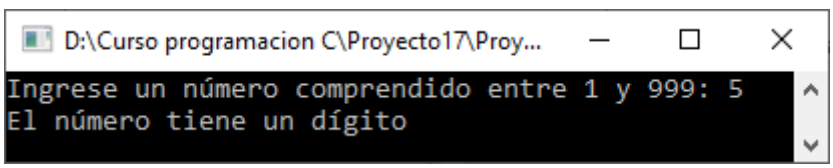
Capítulo 18.- Estructuras condicionales anidadas – 4

Problema propuesto

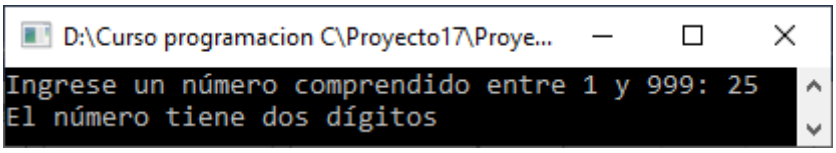
Confeccionar un programa que permita cargar un número entero positivo de hasta tres cifras y muestre un mensaje indicando si tiene 1, 2 o 3 cifras. Mostrar un mensaje de error si el número de cifras es mayor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto17
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num;
            string line;
            Console.Write("Ingrese un número comprendido entre 1 y 999: ");
            line = Console.ReadLine();
            num = int.Parse(line);
            if (num <10)
            {
                Console.Write("El número tiene un dígito");
            }
            else
            {
                if (num<100)
                {
                    Console.Write("El número tiene dos dígitos");
                }
                else
                {
                    if (num<1000)
                    {
                        Console.Write("El número tiene tres dígitos");
                    }
                    else
                    {
                        Console.Write("Error el número tiene más de tres dígitos");
                    }
                }
            }
            Console.ReadKey();
        }
    }
}
```

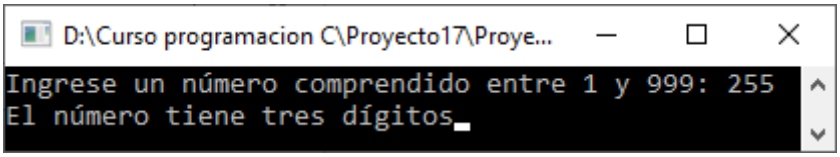
Si ejecutamos este seré el resultado introduciendo el valor 5:



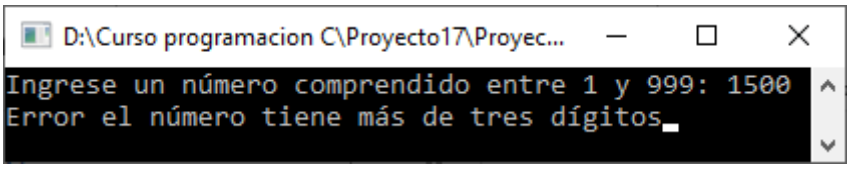
Ejecutamos de nuevo introduciendo el valor 25:



Ejecutamos de nuevo introduciendo el valor 255:



Volvemos a ejecutar introduciendo el valor 1500:



Capítulo 19.- Estructuras condicionales anidadas – 5

Problema propuesto

Un postulante de empleo, realiza un test de capacitación, se obtuvo la siguiente información:

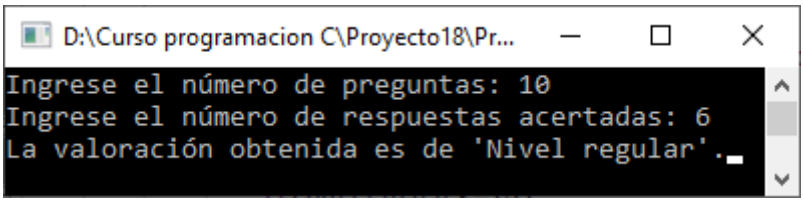
Cantidad total de preguntas que se le realizaron y la cantidad de preguntas que contestó correctamente. Se pide confeccionar un programa que ingrese los dos datos por teclado e informe el nivel del mismo según el porcentaje de respuestas correctas que ha obtenido y sabiendo que:

- Nivel máximo: Porcentaje>=90%.
- Nivel medio: Porcentaje>=75% y <90%.
- Nivel regular: Porcentaje>=50% y <75%.
- Fuera de nivel: Porcentaje<50%.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto18
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int totPreguntas, resAcertadas, porcentaje;
            string line;
            Console.Write("Ingrese el número de preguntas: ");
            line = Console.ReadLine();
            totPreguntas = int.Parse(line);
            Console.Write("Ingrese el número de respuestas acertadas: ");
            line = Console.ReadLine();
            resAcertadas = int.Parse(line);
            porcentaje =  resAcertadas * 100 / totPreguntas;
            if (porcentaje>=90)
            {
                Console.Write("La valoración obtenida es de 'Nivel máximo'.");
            }
            else
            {
                if(porcentaje>=75)
                {
                    Console.Write("La valoración obtenida es de 'Nivel medio'.");
                }
                else
                {
                    if(porcentaje>=50)
                    {
                        Console.Write("La valoración obtenida es de 'Nivel regular'.");
                    }
                    else
                    {
                        Console.Write("La valoración obtenida es de 'Fuera de nivel'.");
                    }
                }
            }

            Console.ReadKey();
```

```
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 20.- Condiciones compuestas con operadores lógicos – 1

Hasta ahora hemos visto los operadores:

```
relacionales (>, <, >=, <= , ==, !=)
matemáticos (+, -, *, /, %)
```

Pero nos están faltando otros operadores imprescindibles:

```
lógicos (&&, ||).
```

Estos dos operadores se emplean fundamentalmente en las estructuras condicionales para agrupar varias condiciones simples.

Operador &&

Traducido se lo lee como "Y". Si la Condición 1 es verdadera Y la Condición 2 es verdadera luego ejecuta la rama de verdadero.

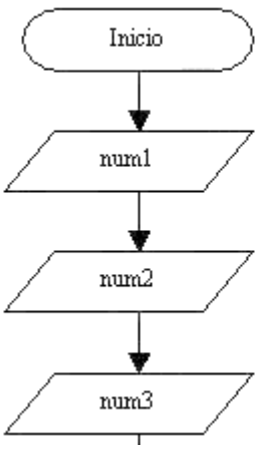
Cuando vinculamos dos o más condiciones con el operador "&&", todas las condiciones deben ser verdaderas para que el resultado de la condición compuesta sea Verdadero y continúe por la rama del verdadero de la estructura condicional.

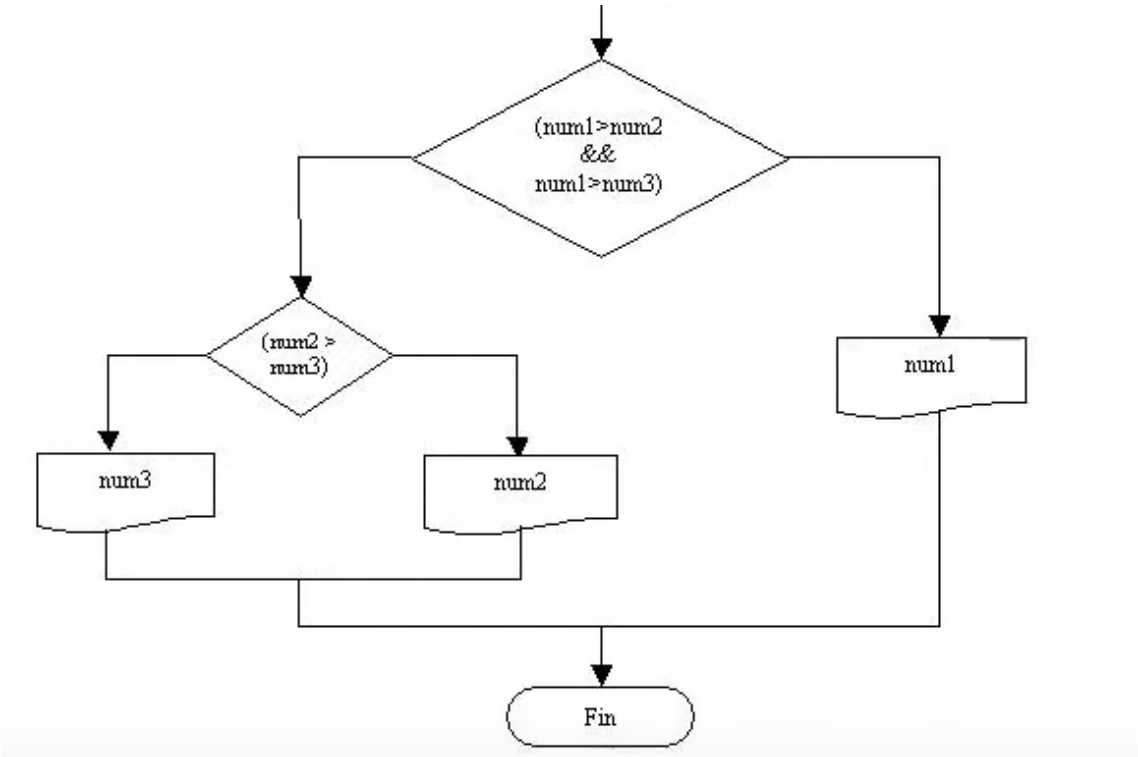
La utilización de operadores lógicos permiten en muchos casos plantear algoritmos más cortos y comprensibles.

Problema

Confecciona un programa que lea por teclado tres números distintos y nos muestre el mayor.

Diagrama de flujo:





Este ejercicio está resuelto sin emplear operadores lógicos en un capítulo anterior del tutorial. La primera estructura condicional es una ESTRUCTURA CONDICIONAL COMPUESTA con una CONDICIÓN COMPUESTA.

Podemos leerla de la siguiente forma:

Si el contenido de la variable num1 es mayor al contenido de la variable num2 Y si el contenido de la variable num1 es mayor al contenido de la variable num3 entonces la CONDICIÓN COMPUESTA resulta verdadera.

Si una de las condiciones simples da falso la CONDICIÓN COMPUESTA da Falso y continúa por la rama del falso.

Es decir que se mostrará el contenido de num1 y si solo num1>num2 y num1>num3.

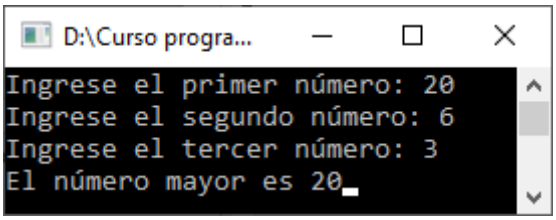
En caso de ser Falsa la condición, analizamos el contenido de num2 y num3 para ver cual tiene un valor mayor.

En esta segunda estructura condicional no se requiere operadores lógicos al haber una condición simple.

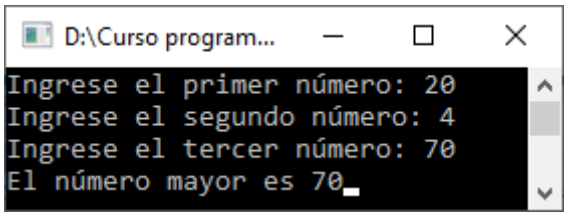
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto19
{
    0 referencias
    internal class Program
    {
        0 referencias
```

```
        static void Main(string[] args)
        {
            int num1, num2, num3;
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
            if (num1>num2 && num1>num3)
            {
                Console.Write("El número mayor es ");
                Console.Write(num1);
            }
            else
            {
                if(num2>num3)
                {
                    Console.Write("El número mayor es ");
                    Console.Write(num2);
                }
                else
                {
                    Console.Write("El número mayor es ");
                    Console.Write(num3);
                }
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

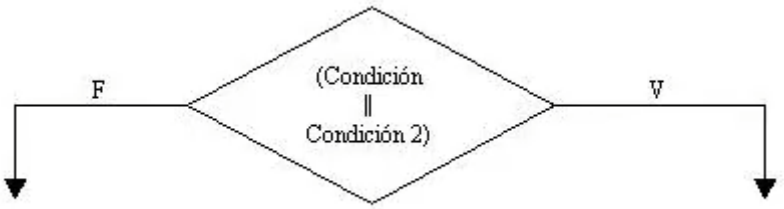


Ejecutamos de nuevo:



Capítulo 21.- Condiciones compuestas con operadores lógicos – 2

Operador ||



Traducido se lo lee como "O". Si la condición 1 es Verdadera O la condición 2 es Verdadera, luego se ejecuta la rama del Verdadero.

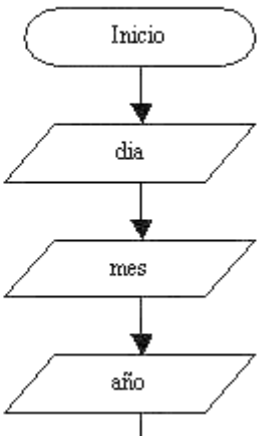
Cuando vinculamos dos o más condiciones con el operador 2Or", con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

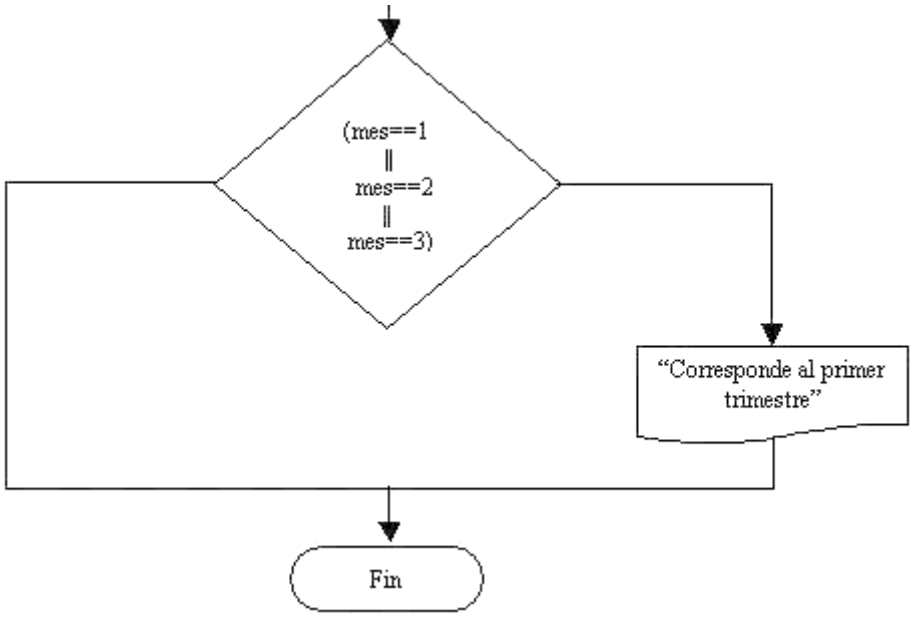
Problema

Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje si corresponde al primer trimestre del año (enero, febrero o marzo) Cargar por teclado el valor numérico del día, mes y años.

Ejemplo: día: 10: mes:1 año: 2020.

Diagrama de flujo





La carga de una fecha se hace por partes, ingresamos las variables dia, mes y año.

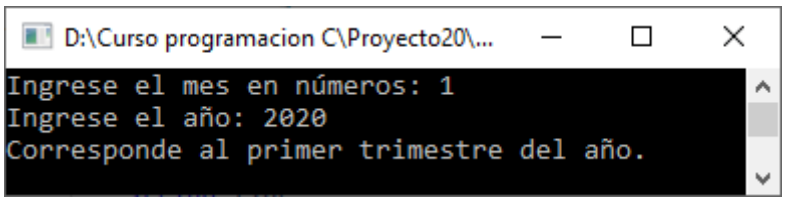
Mostramos el mensaje "Corresponde al primer trimestre" en caso que el mes ingresado por teclado sea igual a 1, 2 ó 3.

En la condición no participan las variables día y año.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto20
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int dia, mes, año;
            string line;
            Console.Write("Ingrese el día: ");
            line = Console.ReadLine();
            dia = int.Parse(line);
            Console.Write("Ingrese el mes en números: ");
            line = Console.ReadLine();
            mes = int.Parse(line);
            Console.Write("Ingrese el año: ");
            line = Console.ReadLine();
            año = int.Parse(line);
            if (mes==1 || mes==2 || mes==3)
            {
                Console.Write("Corresponde al primer trimestre del año.");
            }
            Console.ReadKey();
```

```
        }
    }
}
```

Vamos a ejecutar:



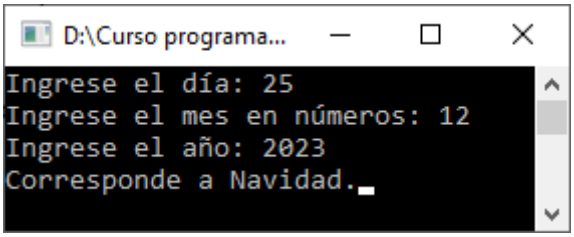
Capítulo 22.- Condiciones compuestas con operadores lógicos – 3

Problema propuesto

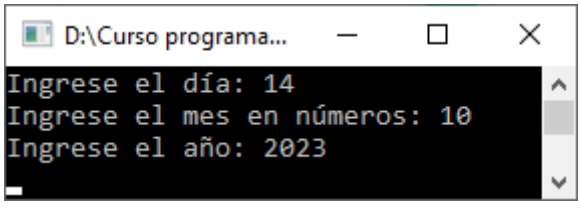
Realizar un programa que pida cargar una fecha cualquiera, luego verifica si dicha fecha corresponde a Navidad.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proycto21
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int dia, mes, año;
            string line;
            Console.Write("Ingrese el día: ");
            line = Console.ReadLine();
            dia = int.Parse(line);
            Console.Write("Ingrese el mes en números: ");
            line = Console.ReadLine();
            mes = int.Parse(line);
            Console.Write("Ingrese el año: ");
            line = Console.ReadLine();
            año = int.Parse(line);
            if (dia == 25 && mes == 12)
            {
                Console.Write("Corresponde a Navidad.");
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Ahora con una fecha que no coincide con Navidad:



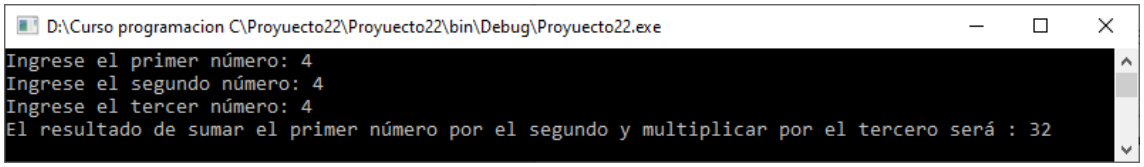
Capítulo 23.- Condiciones compuestas con operadores lógicos – 4

Problema propuesto

Se ingresan tres valores por teclado, si todos son iguales se imprime la suma del primero con el segundo y a este resultado se lo multiplica por el tercero.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyuecto22
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3;
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
            if (num1 == num2 && num1 == num3)
            {
                int suma = num1 + num2;
                int resultado = suma * num3;
                Console.Write("El resultado de sumar el primer número por el segundo ");
                Console.Write("y multiplicar por el tercero será : ");
                Console.Write(resultado);

            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



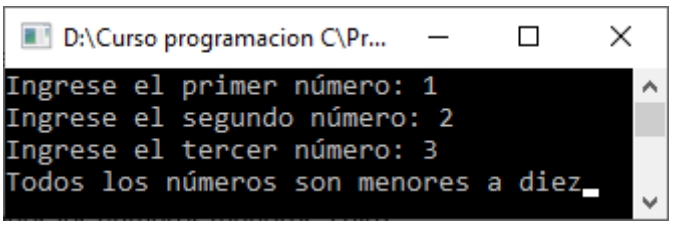
Capítulo 24.- Condiciones compuestas con operadores lógicos – 5

Problema propuesto

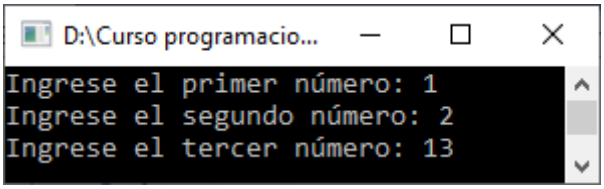
Se ingresan por teclado tres números, si todos los valores ingresaos son menores a 10, imprimir en pantalla la leyenda "Todos los números son menores de diez".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto23
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3;
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
            if (num1 < 10 && num2 < 10 & num3 < 10)
            {
                Console.Write("Todos los números son menores a diez");
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos introduciendo todos los números menores a diez:



Ahora algún número no será menor de diez:



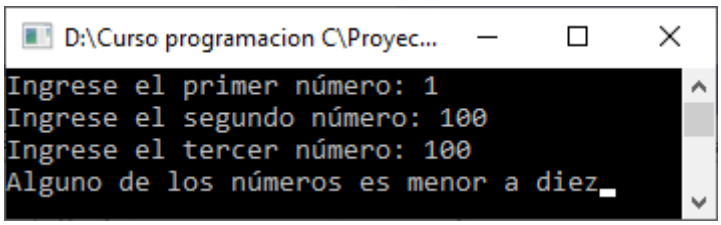
Capítulo 25.- Condiciones compuestas con operadores lógicos – 6

Problema propuesto

Se ingresan por teclado tres números, si al menos uno de los valores ingresados es menor a 10, imprimir en pantalla la leyenda "Alguno de los números es menor a diez".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto24
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3;
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
            if (num1 < 10 || num2 < 10 || num3 < 10)
            {
                Console.Write("Alguno de los números es menor a diez");
            }
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar:

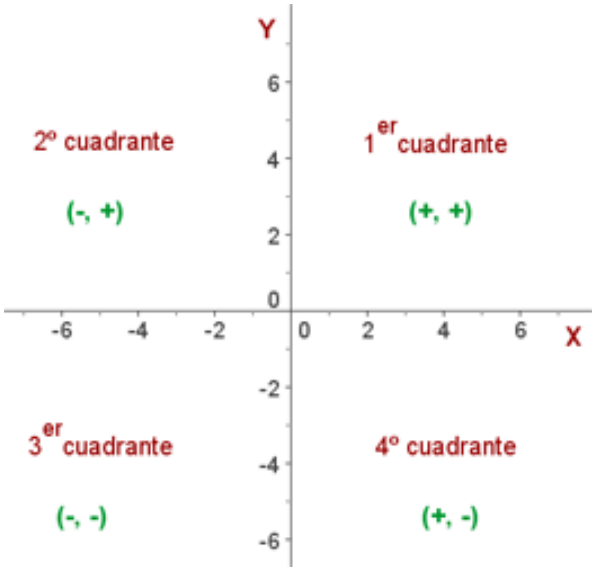


Capítulo 26.- Condiciones compuestas con operadores lógicos – 7

Problema propuesto

Escribir un programa que pida ingresar las coordenadas de un punto en el plano, es decir dos valores enteros x e y (distintos a cero).

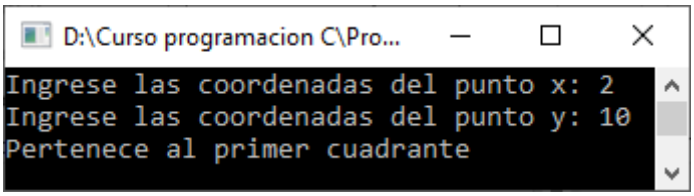
Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto. (1º Cuadrante si x > 0 Y y > 0, 2º Cuadrante x < 0 Y y > 0, etc.)



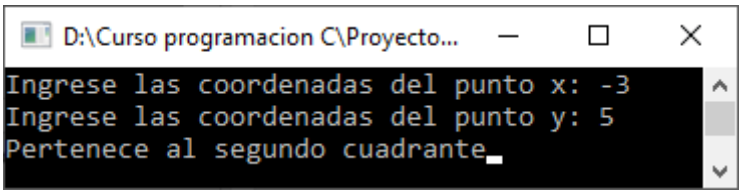

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto25
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x, y;
            string line;
            Console.Write("Ingrese las coordenadas del punto x: ");
            line = Console.ReadLine();
            x = int.Parse(line);
            Console.Write("Ingrese las coordenadas del punto y: ");
            line = Console.ReadLine();
            y = int.Parse(line);
            if(x>0 && y>0)
            {
                Console.Write("Pertenece al primer cuadrante");
            }
            else
```

```
                {
                    if(x<0 && y<0)
                    {
                        Console.Write("Pertenece al tercer cuadrante");
                    }
                    else
                    {
                        if(x>0 && y<0)
                        {
                            Console.Write("Pertenece al cuarto cuadrante");
                        }
                        else
                        {
                            Console.Write("Se encuentra en un eje");
                        }
                    }
                }
            }
            Console.ReadKey();
        }
    }
}
```

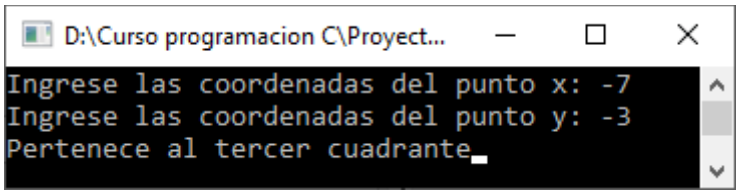
Vamos a ejecutar introduciendo en x 10 e y 2:



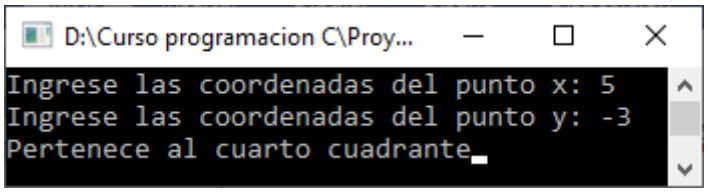
Ahora los valores en x -3 e y 5:



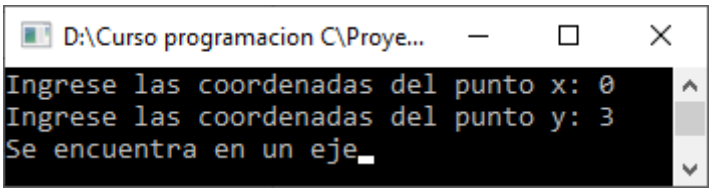
Ahora los valores x -7 e y -3:



Ahora los valores x 5 e y -3:



Ahora pondremos en x o en y un valor cero:


D:\Curso programacion C\Proye...
Ingrese las coordenadas del punto x: 0
Ingrese las coordenadas del punto y: 3
Se encuentra en un eje

Capítulo 27.- Condiciones compuestas con operadores lógicos – 8

Problema propuesto

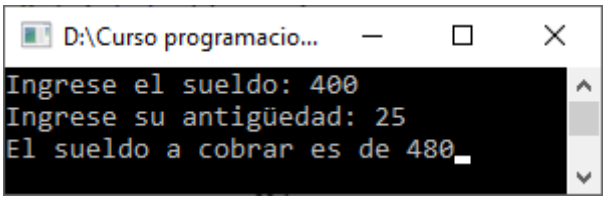
De un operario se conoce el sueldo y los años de antigüedad. Se pide confeccionar un programa que lea los datos de entrada e informe:

a) Si el sueldo es inferior a 500 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20%, mostrar el sueldo a pagar.
b) Si el sueldo es inferior a 500 pero su antigüedad es menor a 10 años, otorgarle un aumento de 5%.
c) Si el sueldo es mayor o igual a 500 mostrar el sueldo en pantalla sin cambios.

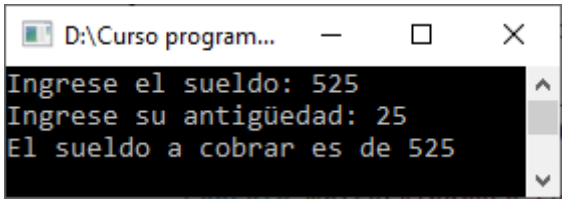
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto26
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int sueldo, antiguedad, totSueldo;
            string line;
            Console.Write("Ingrese el sueldo: ");
            line = Console.ReadLine();
            sueldo = int.Parse(line);
            Console.Write("Ingrese su antigüedad: ");
            line= Console.ReadLine();
            antiguedad = int.Parse(line);
            if (sueldo < 500 && antiguedad >= 10)
            {
                totSueldo = sueldo + (sueldo * 20 / 100);
                Console.Write("El sueldo a cobrar es de ");
                Console.Write(totSueldo);
            }
            else
            {
                if (sueldo < 500 && antiguedad < 10)
                {
                    totSueldo = sueldo + (sueldo * 5 / 100);
                    Console.Write("El sueldo a cobrar es de ");
                    Console.Write(totSueldo);
                }
                else
                {
                    Console.Write("El sueldo a cobrar es de ");
                    Console.Write(sueldo);
                }
            }
```

```
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar:



Ejecutamos de nuevo con un sueldo superior a 500.



Capítulo 28.- Condiciones compuestas con operadores lógicos – 9

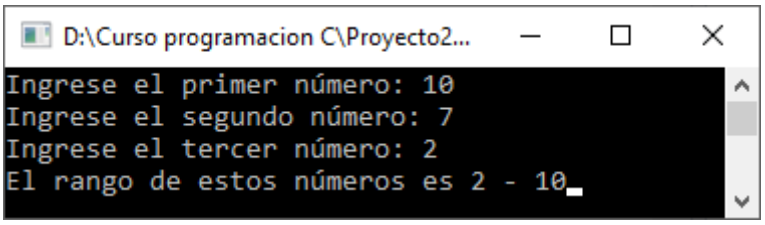
Problema propuesto

Escribir un programa en el cual: dada una lista de tres valores numéricos distintos se calcule e informe su rango de variación (debe mostrar el menor y el mayor de ellos).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto27
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num1, num2, num3, max, min;
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
            if (num1 > num2 && num1 > num3)
            {
                max = num1;
            }
            else
            {
                if (num2 > num3)
                {
                    max = num2;
                }
                else
                {
                    max = num3;
                }
            }
```

```
            if (num1 < num2 && num1 < num3)
            {
                min = num1;
            }
            else
            {
                if (num2 < num3)
                {
                    min = num2;
                }
                else
                {
                    min = num3;
                }

            }
            Console.Write("El rango de estos números es ");
            Console.Write(min);
            Console.Write(" - ");
            Console.Write(max);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 29.- Estructura repetitiva while – 1

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existe otro tipo de estructura tan importante como las anteriores que son las estructuras REPETITIVAS.

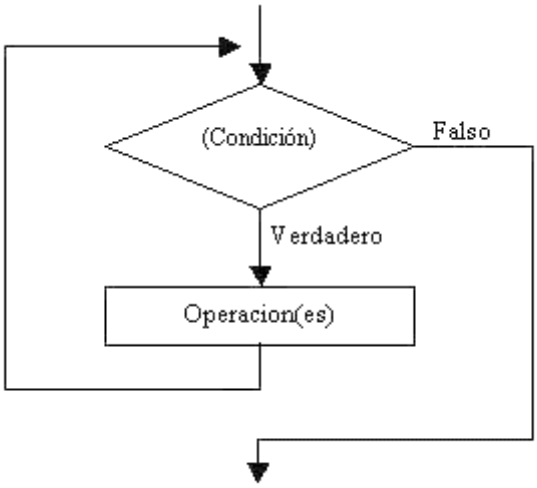
Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

- La o las sentencias se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repita o no las sentencias.

Estructura repetitiva while

Representación gráfica de la estructura while:



No debemos confundir la representación gráfica de la estructura repetitiva while (Mientras) con la estructura condicional if (Si).

Funcionamiento: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos por la rama del Verdadero.

A la rama del verdadero la graficamos en la parte inferior de la estructura repetitiva.

En caso que la condición sea Falsa continúa por la rama del Falso y sale de la estructura repetitiva para continuar con la ejecución del algoritmo.

El bloque se repite MIENTRAS la condición sea verdadera.

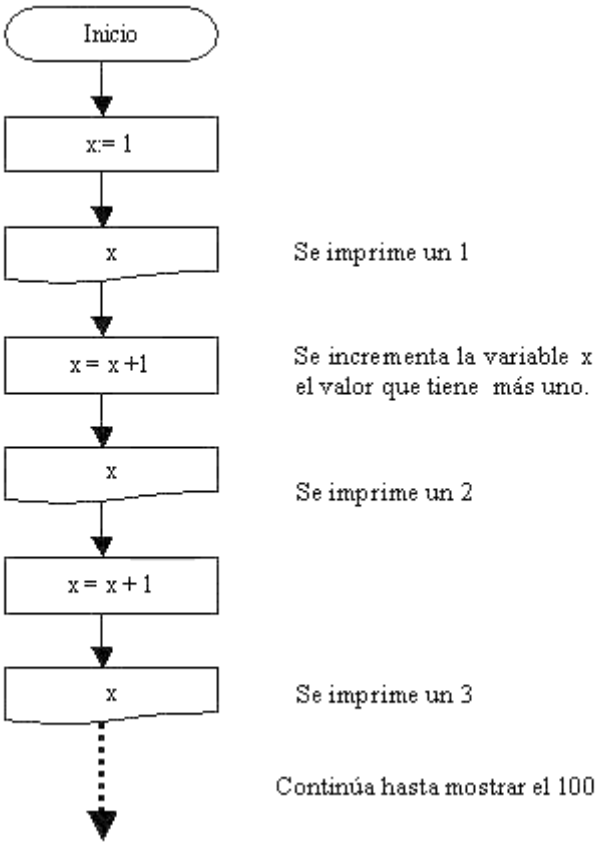
Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, nunca finalizará el programa.

Problema 1:

Realiza un programa que imprima en pantalla los números del 1 al 100.

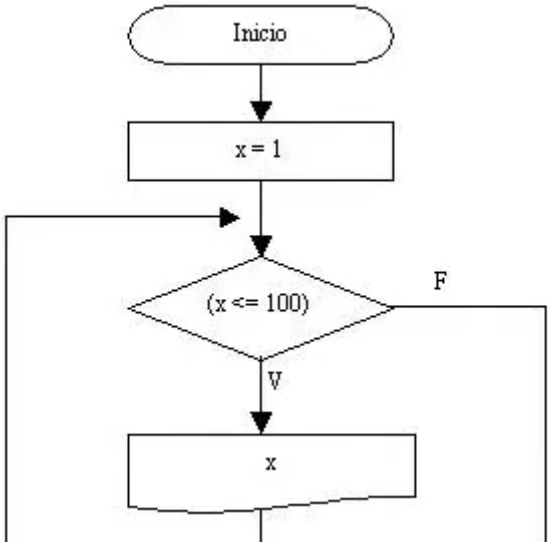
Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.

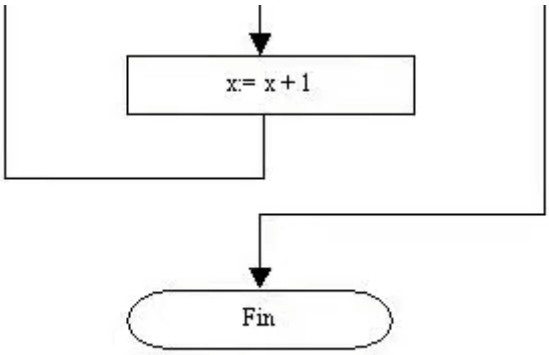
Diagrama de flujo:



Si continuamos con el diagrama no nos alcanzaría los próximas 5 páginas para finalizarlo. Emplear una estructura secuencial para resolver este problema produce un diagrama de flujo y un programa en C# muy largo.

Ahora veamos la solución empleando una estructura repetitiva while:





Es muy importante analizar este diagrama:

La primera operación inicializa la variable x en 1, seguidamente comienza la estructura repetitiva while y disponemos la siguiente condición (x <= 100), se lee MIENTRAS la variable x sea menor o igual a 100.

Al ejecutar la condición retorna VERDADERO porque el contenido de x (1) es menor o igual a 100. Al ser la condición verdadera se ejecuta el bloque de instrucciones que contiene la estructura while. El bloque de instrucciones contiene una salida y una operación.

Se imprime el contenido de x, y seguidamente se incrementa en uno.

La operación x = x + 1 se lee como "en la variable x se guarda el contenido de x más 1". Es decir, si x contiene 1 luego de ejecutarse esta operación se almacenará en x un 2.

Al finalizar el bloque de instrucciones que contiene la estructura repetitiva se verifica nuevamente la condición de la estructura repetitiva y se repite el proceso explicado anteriormente.

Mientras la condición retorne verdadero se ejecuta el bloque de instrucciones; al retornar falso la verificación de la condición se sale de la estructura repetitiva y continua el algoritmo, en este caso finaliza el programa.

Lo más difícil es la definición de la condición de la estructura while y qué bloque de instrucciones se van a repetir. Observar que si, por ejemplo, disponemos la condición x>=100 (si x es mayor o igual a 100) no provoca ningún error sintáctico pero estamos en presencia de un error lógico porque al evaluarse por primera vez la condición retorna falso y no se ejecuta el bloque de instrucciones que queríamos repetir 100 veces.

No existe una RECETA para definir una condición de una estructura repetitiva, sino que se logra con una práctica continua solucionando problemas.

Una vez planteado el diagrama debemos verificar si el mismo es una solución válida al problema (en este caso se debe imprimir los números del 1 al 100 en pantalla), para ello podemos hacer una seguimiento del flujo del diagrama y los valores que toman las variables a lo largo de la ejecución:

```
x
1
2
3
4
.
.
```

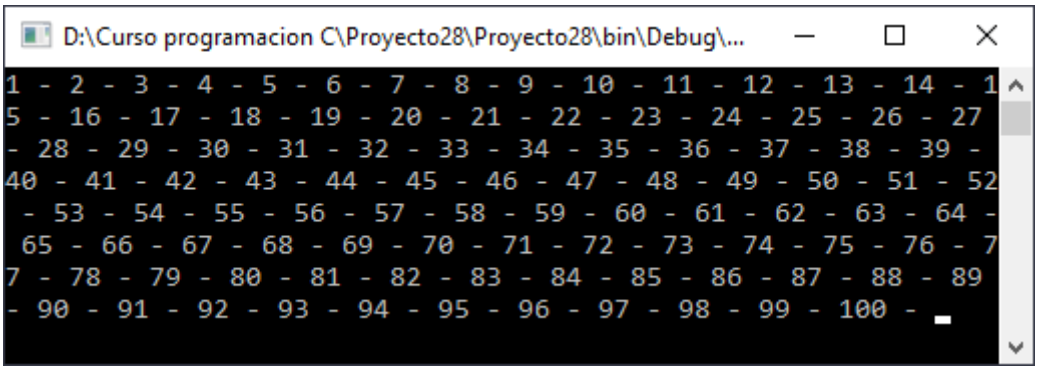
```
        100
        101  Cuando x vale 101 la condición de la estructura repetitiva retorna
falso,
             en este caso finaliza el diagrama.
```

Importante: Podemos observar que el bloque repetitivo puede no ejecutarse ninguna vez si la condición retorna falso por primera vez.

La variable x debe estar inicializada con algún valor antes que se ejecute la operación x)x+1 en caso de no estar inicializada aparece un error de compilación.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto28
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x;
            x = 1;
            while (x<=100)
            {
                Console.Write(x);
                Console.Write(" - ");
                x = x + 1;
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Recordemos que un problema no estará 100% solucionado si no hacemos el programa C# que muestre los resultados buscados.

Probemos algunas modificaciones de este programa y veamos que cambios se debería hacer para:

1- Imprimir los números del 1 al 500.
2- Imprimir los números del 50 al 100.
3- Imprimir los números del -50 al 0.
4- Imprimir los números del 2 al 100 pero de 2 en 2 (2, 4, 6, 8, …., 100).

Respuestas:

1- Debemos cambiar la condición del while con x<=500.
2- Debemos inicializar x con el valor 50.
3- Inicializar x con el valor -50 y fijar la condición x<=0.
4- Inicializar a x con el valor 2 y dentro del bloque repetitivo incrementar a x en 2 (x=x+2).

Capítulo 30.- Estructura repetitiva while – 2

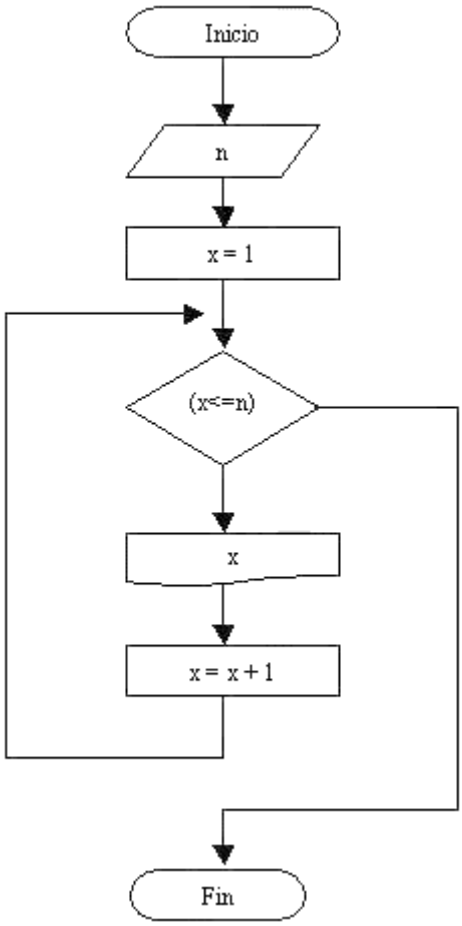
Problema

Escribir un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno.

Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.

Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en C# de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.

Diagrama de flujo:

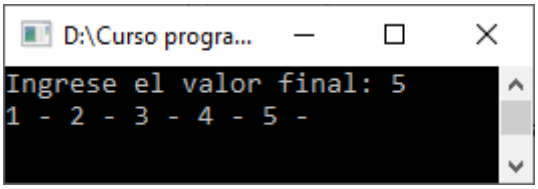


Podemos observar que se ingresa por teclado la variable n. El operador puede cargar cualquier valor.

Si el operador carga 10 el bloque repetitivo se ejecutará 10 veces, ya que la condición es "Mientras x<=n", es decir "mientras x sea menor o igual a 10"; pues x comienza en 1 y se incrementa en uno cada vez que se ejecuta el bloque repetitivo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto30
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int n, x;
            string line;
            Console.Write("Ingrese el valor final: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            x = 1;
            while (x<=n)
            {
                Console.Write(x);
                Console.Write(" - ");
                x = x + 1;
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



A la prueba del diagrama la podemos realizar dándole valores a la variable; por ejemplo, si ingresamos 5 el seguimiento es el siguiente:

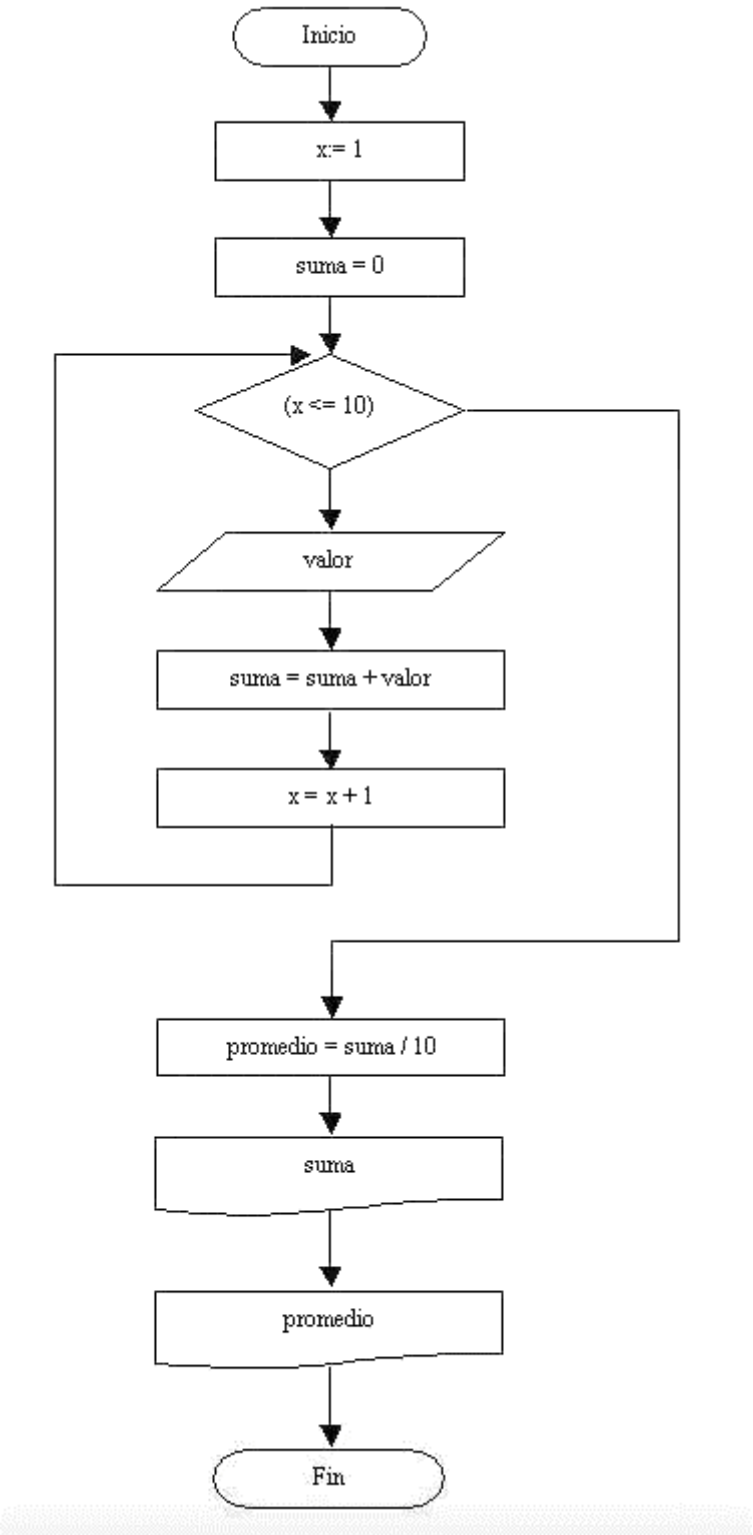
```
n       x
5       1  (Se imprime el contenido de x)
        2       "               "
        3       "               "
        4       "               "
        5       "               "
        6 (Sale del while porque 6 no es menor o igual a 5)
```

Capítulo 31.- Estructura repetitiva while – 3

Problema:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.

Diagrama de flujo:



En este problema, se semejanza de los anteriores, llevamos un CONTDOR llamado x que nos sirve para contar las vueltas que debe repetir el while.

También aparece un ACUMULADOR (un acumulador es un tipo especial de variable que incrementa o decrementa con valores variables durante la ejecución del programa).

Hemos dado el nombre de suma a nuestro acumulador. Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementará con el contenido ingresado en la variable valor.

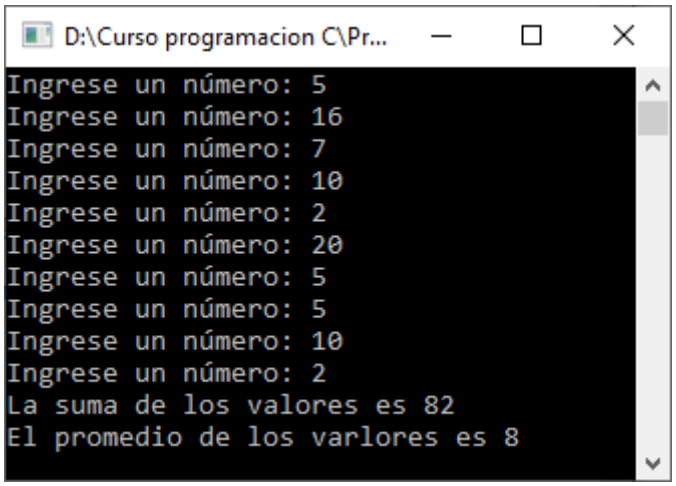
La prueba del diagrama de flujo se realiza dándole valores a las variables:

```
valor           suma            x          promedio
                 0              1
(Antes de entrar a la estructura repetitiva estos son los valores).
 5               5              1
16              21              2
 7              28              3
10              38              4
 2              40              5
20              60              6
 5              65              7
 5              70              8
10              80              9
 2              82             10
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto31
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int valor, x, suma, promedio;
            string line;
            x = 1;
            suma = 0;
            while(x<=10)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                valor = int.Parse(line);
                suma = suma + valor;
                x = x + 1;
            }
            promedio = suma / 10;
            Console.Write("La suma de los valores es ");
            Console.WriteLine(suma);
            Console.Write("El promedio de los varlores es ");
            Console.WriteLine(promedio);
            Console.ReadKey();
```

```
        }
    }
}
```

Si ejecutamos este será el resultado:



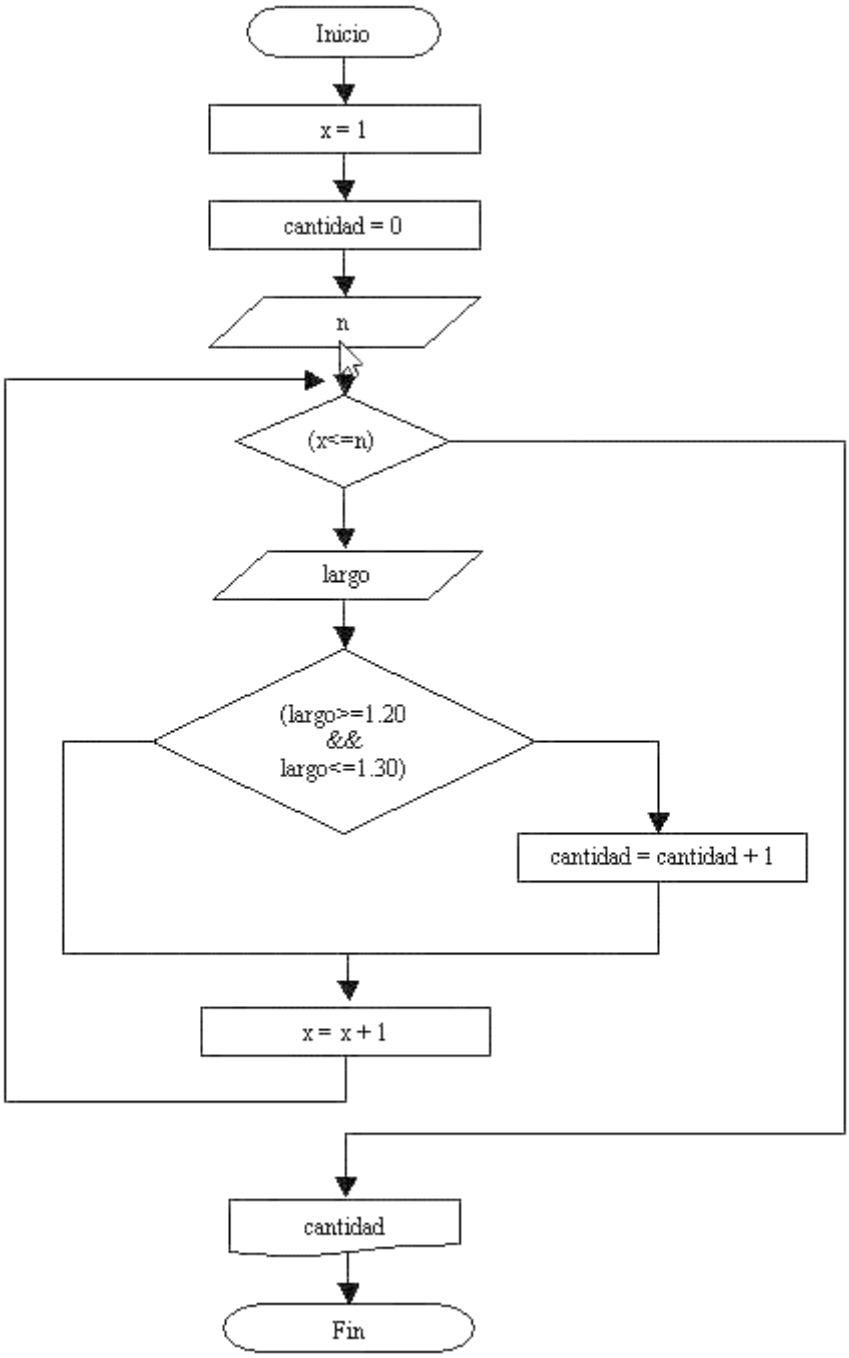
Capítulo 32.- Estructura repetitiva while – 4

Problema

Una planta que fabrica perfiles de hierro posee un lote de n piezas.

Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1,20 y 1,30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.

Diagrama de flujo:



Podemos observar que dentro de una estructura repetitiva puede haber estructuras condicionales (inclusive puede haber otras estructuras repetitivas que veremos más adelante).

En este problema hay que cargar inicialmente la cantidad de piezas a ingresar (n), seguidamente se cargan n valores de largos de piezas.

Cada vez que ingresamos un largo de pieza (largo) verificamos si es una medida correcta (debe estar entre 1.20 y 1.30 el largo para que sea correcta), en aso de ser correcta la CONTAMOS (incrementamos la variable cantidad en 1).

Al contador cantidad lo inicializamos en cero porque inicialmente no se ha cargado ningún largo de medida.

Cuando salimos de la estructura repetitiva porque se han cargado n largos de piezas mostramos por pantalla el contador cantidad (que representan la cantidad de piezas aptas).

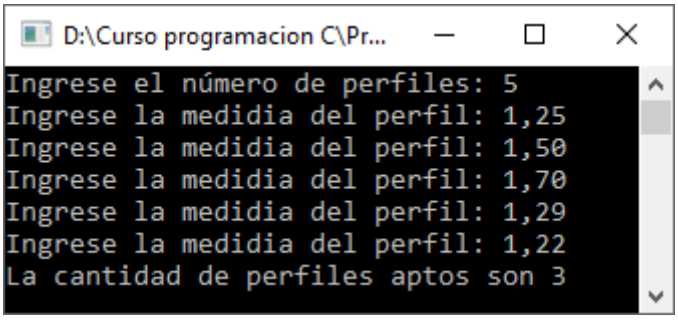
En este problema tenemos dos CONTADORES:

x	(Cuenta la cantidad de piezas cargadas hasta el momento)
cantidad	(Cuenta los perfiles de hierro aptos)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto32
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x, n, cantidad;
            float largo;
            string line;
            x = 1;
            cantidad = 0;
            Console.Write("Ingrese el número de perfiles: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            while (x<=n)
            {
                Console.Write("Ingrese la medidia del perfil: ");
                line = Console.ReadLine();
                largo = float.Parse(line);
                if (largo >= 1.20 && largo <= 1.30)
                {
                    cantidad = cantidad + 1;
                }
                x = x + 1;
            }
```

```
            Console.Write("La cantidad de perfiles aptos son ");
            Console.Write(cantidad);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



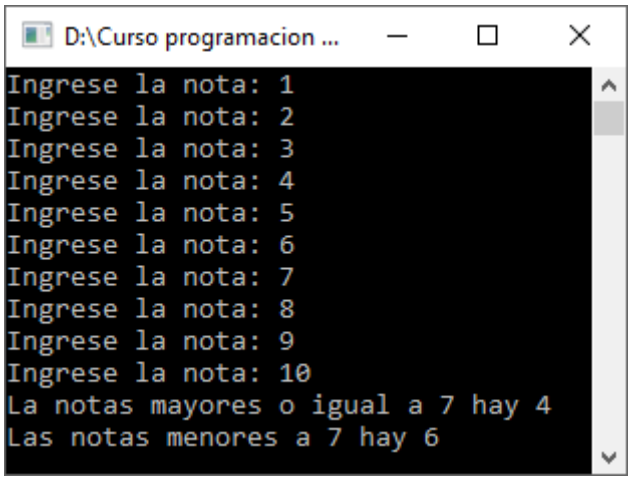
Capítulo 33.- Estructura repetitiva while – 5

Problema propuesto

Escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto33
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x, nota, mayor, menor;
            string line;
            x = 1;
            mayor = 0;
            menor = 0;
            while (x<=10)
            {
                Console.Write("Ingrese la nota: ");
                line = Console.ReadLine();
                nota = int.Parse(line);
                if (nota >=7)
                {
                    mayor=mayor+1;
                }
                else
                {
                    menor = menor + 1;
                }
                x = x + 1;
            }
            Console.Write("La notas mayores o igual a 7 hay ");
            Console.WriteLine(mayor);
            Console.Write("Las notas menores a 7 hay ");
            Console.Write(menor);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Curso programacion ...
Ingrese la nota: 1
Ingrese la nota: 2
Ingrese la nota: 3
Ingrese la nota: 4
Ingrese la nota: 5
Ingrese la nota: 6
Ingrese la nota: 7
Ingrese la nota: 8
Ingrese la nota: 9
Ingrese la nota: 10
La notas mayores o igual a 7 hay 4
Las notas menores a 7 hay 6

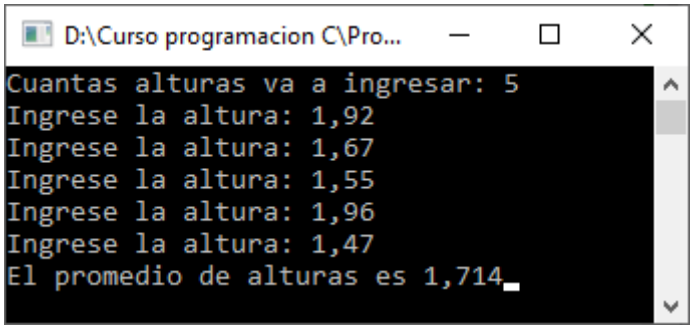
Capítulo 34.- Estructura repetitiva while – 6

Problema propuesto

Se ingresan un conjunto de n alturas de personas por teclado. Mostrar la altura promedio de las personas.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto34
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            float altura, suma, promedio;
            int x, n;
            string line;
            x = 1;
            suma = 0;
            Console.Write("Cuantas alturas va a ingresar: ");
            line = Console.ReadLine();
            n=int.Parse(line);
            while(x<=n)
            {
                Console.Write("Ingrese la altura: ");
                line = Console.ReadLine();
                altura = float.Parse(line);
                suma = suma + altura;
                x = x + 1;
            }
            promedio = suma / n;
            Console.Write("El promedio de alturas es ");
            Console.Write(promedio);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 35.- Estructura repetitiva while – 7

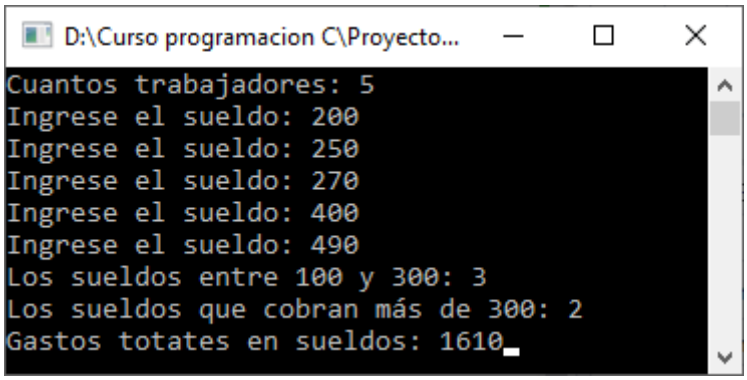
Problema propuesto

En una empresa trabajan n empleados cuyos sueldos oscilan entre $100 y $500, realizar un programa que lea los sueldos que cobran cada empleado e informe cuántos empleados cobran entre $100 y $300 y cuántos cobran más de $300. Además el programa deberá informar el importe que gasta la empresa en sueldos al personal.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto35
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int n, x, sueldo, max, min, total;
            string line;
            min = 0;
            max = 0;
            total = 0;
            x = 1;
            Console.Write("Cuantos trabajadores: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            while (x <=n)
            {
                Console.Write("Ingrese el sueldo: ");
                line = Console.ReadLine() ;
                sueldo = int.Parse(line);
                total = total + sueldo;
                if(sueldo<=300)
                {
                    min = min + 1;
                }
                else
                {
                    max = max + 1;
                }
                x = x + 1;
            }
```

```
            Console.Write("Los sueldos entre 100 y 300: ");
            Console.WriteLine(min);
            Console.Write("Los sueldos que cobran más de 300: ");
            Console.WriteLine(max);
            Console.Write("Gastos totates en sueldos: ");
            Console.Write(total);
            Console.ReadKey();
        }
    }
}
```

Ejecutamos este será el resultado:



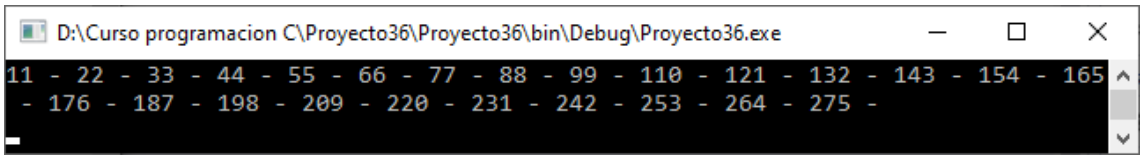
Capítulo 36.- Estructura repetitiva while – 8

Problema propuesto

Realizar un programa que imprima 25 términos de la serie 11 – 22 – 33 – 44, etc. (No se ingresan valores por teclado).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto36
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x = 1;
            while(x<=25)
            {
                Console.Write(x * 11);
                Console.Write(" - ");
                x = x + 1;
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



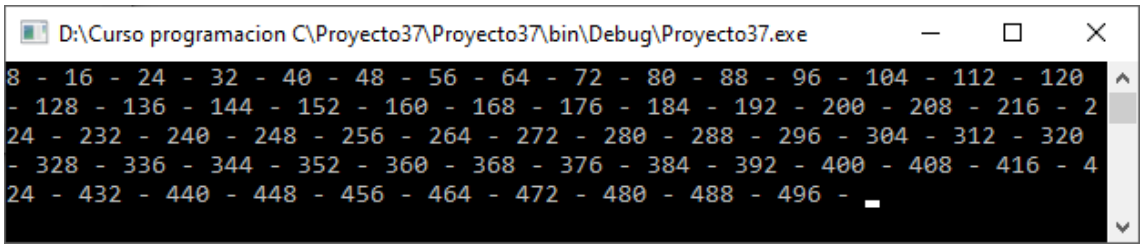
Capítulo 37.- Estructura repetitiva while – 9

Problema propuesto

Mostrar los múltiplos de 8 hasta el valor 500. Debe aparecer 8 – 16 – 24, etc.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto37
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x = 8;
            while (x <= 500)
            {
                Console.Write(x);
                Console.Write(" - ");
                x = x + 8;
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 38.- Estructura repetitiva while – 10

Problema propuesto

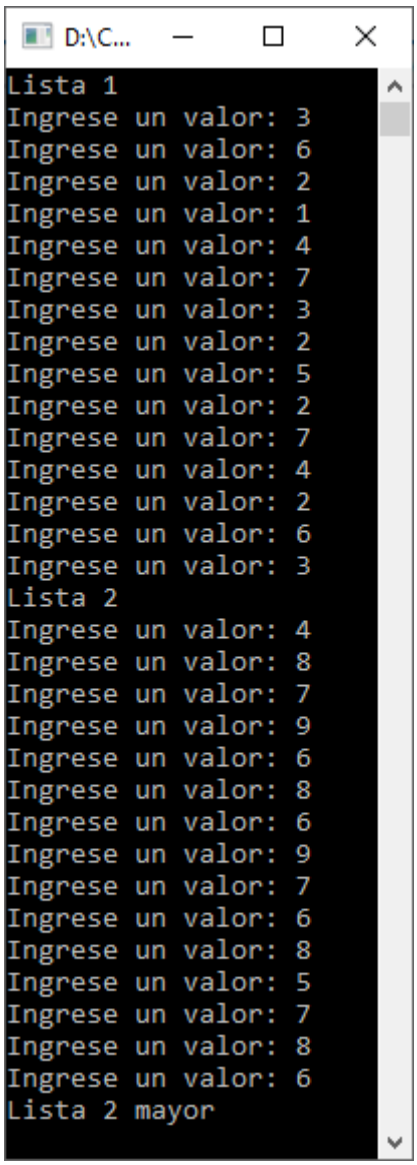
Realizar un programa que permita cargar dos listas de 15 valores cada una. Informar con un mensaje cual de las dos listas tienen un valor acumulado mayor (mensajes "Lista 1 mayor", "Lista 2 mayor", "Listas iguales").

Tener en cuenta que pueden haber dos o más estructuras repetitivas en un algoritmo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto38
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x, num, suma1, suma2;
            string line;
            suma1 = 0;
            suma2 = 0;
            x = 1;
            Console.WriteLine("Lista 1");
            while (x <= 15)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                num = int.Parse(line);
                suma1 = suma1 + num;
                x = x + 1;
            }
            x = 1;
            Console.WriteLine("Lista 2");
            while (x <= 15)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                num = int.Parse(line);
                suma2 = suma2 + num;
                x = x + 1;
            }
            if (suma1 == suma2)
            {
                Console.Write("Listas iguales");
            }
            else
            {
```

```
                if (suma1>suma2)
                {
                    Console.Write("Lista 1 mayor");
                }
                else
                {
                    Console.Write("Lista 2 mayor");
                }
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 39.- Estructura repetitiva while – 11

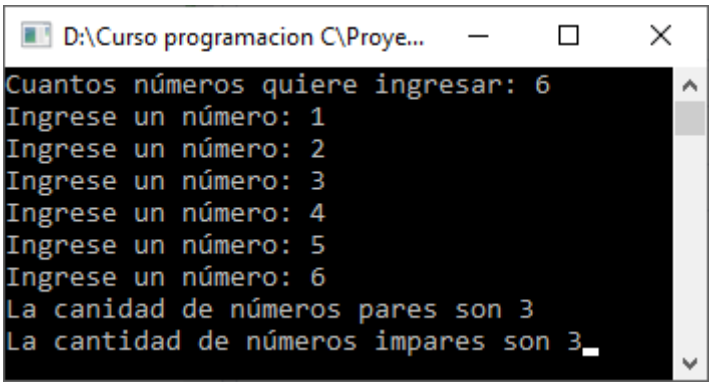
Problema propuesto

Desarrollar un programa que permita cargar n números enteros y luego nos informe cuántos valores fueron pares y cuantos impares.
Emplear el operador "%" en la condición de la estructura condicional:

if (valor%2==0) //Si el if da verdadero luego es par.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto39
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int n, x, num, pares, impares;
            string line;
            pares = 0;
            impares = 0;
            x = 1;
            Console.Write("Cuantos números quiere ingresar: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            while (x <=n)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine() ;
                num = int.Parse(line);
                if (num%2 == 0)
                {
                    pares = pares + 1;
                }
                else
                {
                    impares = impares + 1;
                }
                x = x + 1;
            }
            Console.Write("La canidad de números pares son ");
            Console.WriteLine(pares);
            Console.Write("La cantidad de números impares son ");
            Console.Write(impares);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

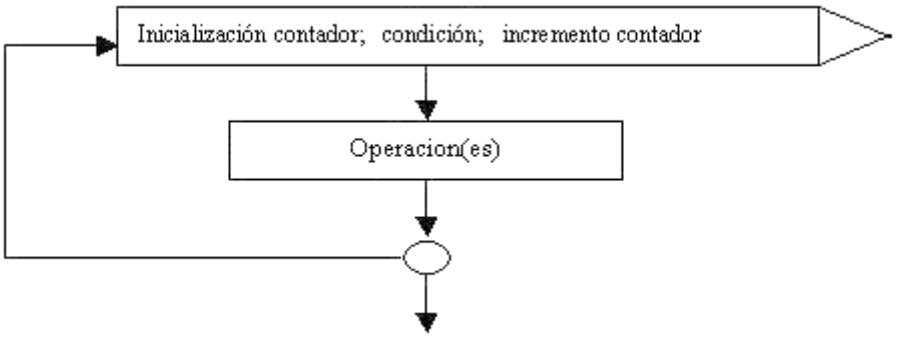


Capítulo 40.- Estructura repetitiva for – 1

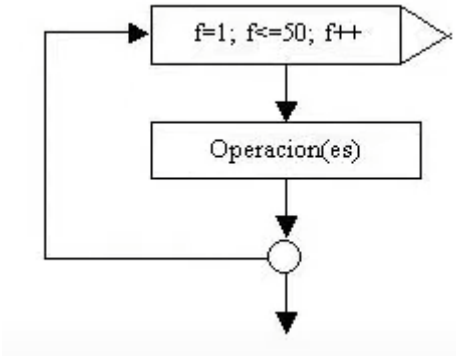
Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while. Pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Ejemplo: Cargar 10 números, ingresar 5 notas de alumnos, etc. Conocemos de antemano la cantidad de veces que queremos que el bloque se repita. Veremos, sin embargo, que en el lenguaje C# la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Representación gráfica:



En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole al dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa).
Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de "inicializar contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercera sección. Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.
Si conocemos la cantidad de veces que se repite el bloque es muy sencillo emplear un for, por ejemplo si queremos que se repita 50 veces el bloque de instrucciones se puede hacerse así:



La variable del for puede tener cualquier nombre. En este ejemplo se ha definido con el nobre de f.

Analicemos el ejemplo:

- La variable f toma inicialmente el valor 1.
- Se controla automáticamente el valor de la condición: como f vale 1 y esto es menor que 50, la condición da verdadero.
- Como la condición fue verdadera, se ejecutan la/s instrucciones.
- Al finalizar de ejecutarlas, se retorna a la instrucción f=f+1, por lo que la variable f se incrementa en uno.
- Se vuelve a controlar (automáticamente) si f es menor o igual a 50. Como ahora su valor es 2, se ejecuta nuevamente el bloque de instrucciones e incrementa la variable del for al terminar el mismo.
- El proceso se repetirá hasta que la variable f sea incrementada al valor 51. En este momento la condición será falsa, y el ciclo se detendrá.

La variable f PUEDE ser modificada dentro del bloque del operaciones del for, aunque esto podría causar problemas de lógica si el programados es inexperto.

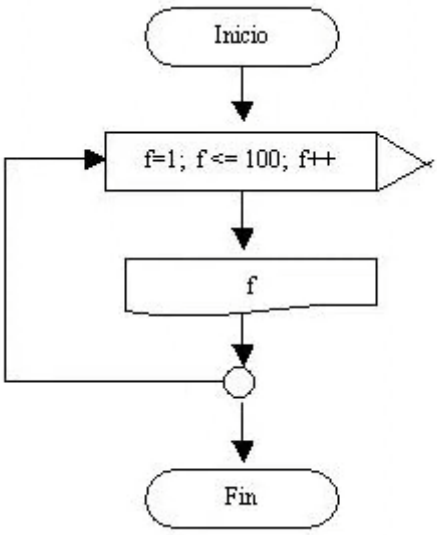
La variable f puede ser inicializada en cualquier valor y finalizar en cualquier valor. Además, no es obligatorio que la instrucción de modificación sea un incremento de tipo contador (f++).

Cualquier instrucción que modifique el valor de la variable es válida. Si por ejemplo se escribe f=f+2 en lugar de f++, el valor de f será incrementado de a 2 en cada vuelta, y no de a 1. En este caso, esto significa que el ciclo no efectuará las 50 vueltas sino sólo 25.

Problema

Realiza un programa que imprima en pantalla los números del 1 al 100.

Diagrama de flujo:



Podemos observar y comparar con el problema realizado con el while. Con la estructura while el CONTADOR x sirve para contar las vueltas. Con el for el CONTADOR f cumple dicha función.

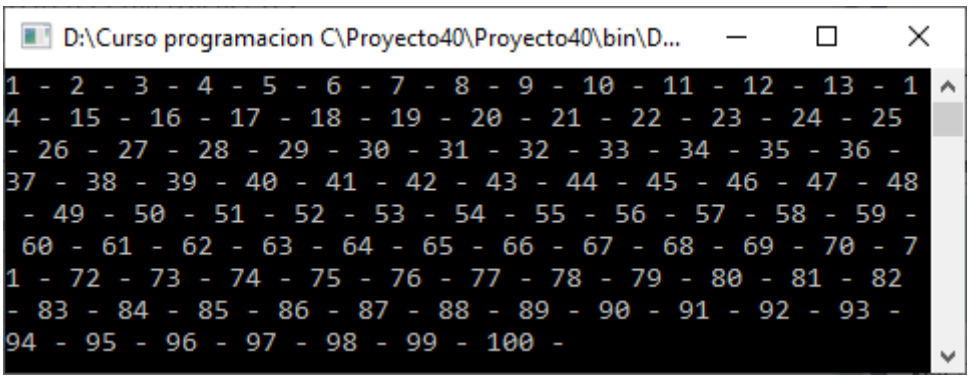
Inicialmente f vale 1 como no es superior a 100 se ejecuta el bloque, imprimimos el contenido de f, al finalizar el bloque repetitivo se incrementa la variable f en 1, como 2 no es superior a 100 se repite el bloque de instrucciones.

Cuando la variable del for llega a 101 sale de la estructura repetitiva y continúa la ejecución del algoritmo que se indica después del círculo.

La variable f (o como sea que se decida llamar) debe estar definida como una variable más.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto40
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f;
            for (f=1; f<=100;  f++)
            {
                Console.Write(f);
                Console.Write(" - ");
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

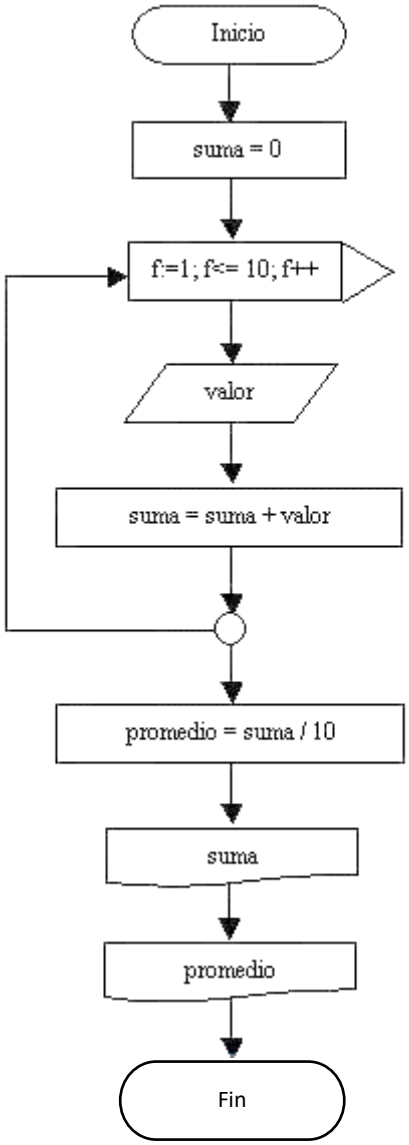


Capítulo 41.- Estructura repetitiva for – 2

Problema

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollamos, lo resolveremos empleando la estructura for.

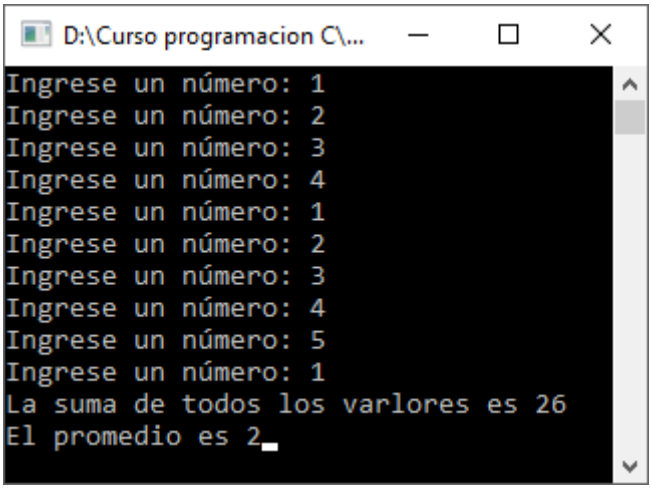
Diagrama de flujo:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto41
{
    0 referencias
```

```
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        int x, valor, suma, promedio;
        string line;
        suma = 0;
        for (x=1; x<=10; x++)
        {
            Console.Write("Ingrese un número: ");
            line = Console.ReadLine();
            valor = int.Parse(line);
            suma = suma + valor;

        }
        promedio = suma / 10;
        Console.Write("La suma de todos los varlores es ");
        Console.WriteLine(suma);
        Console.Write("El promedio es ");
        Console.Write(promedio);
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 42.- Estructura repetitiva for – 3

Problema

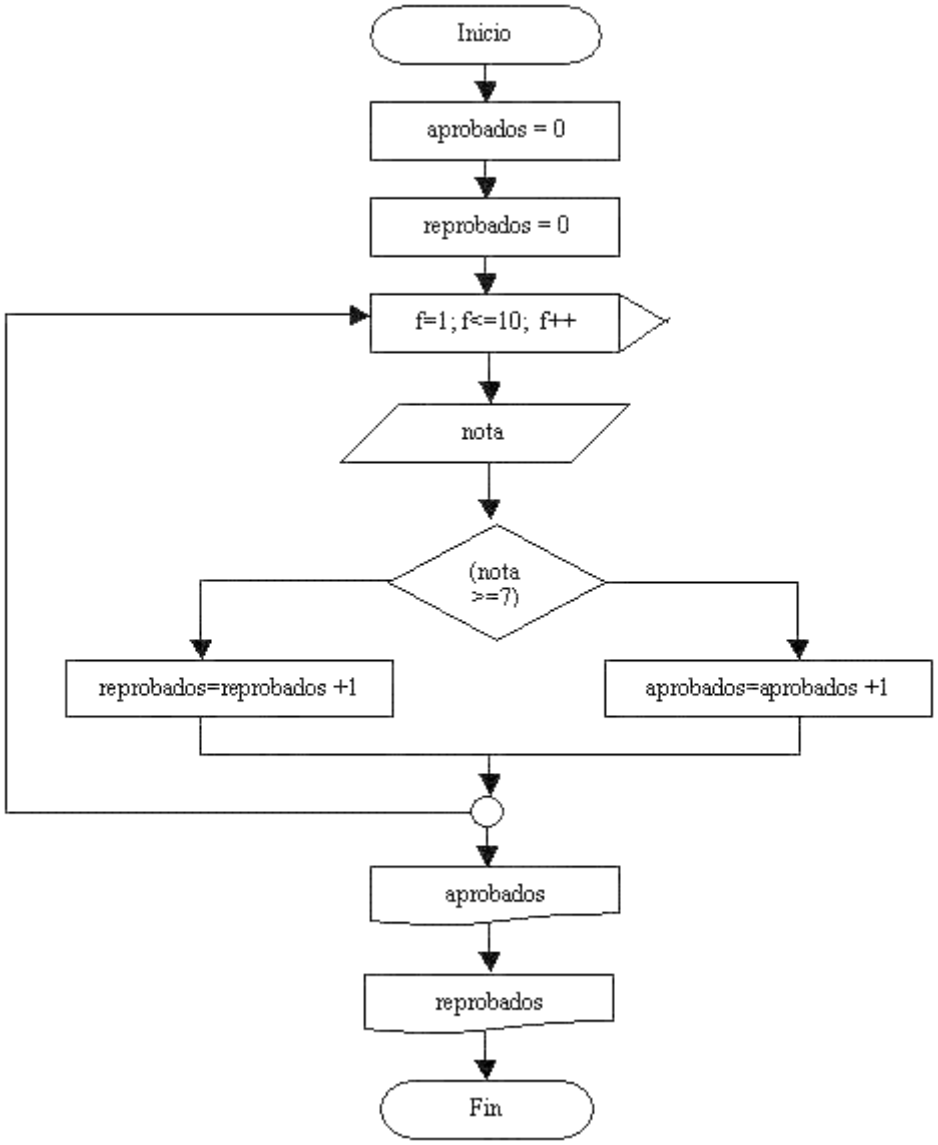
Escribir un programa que lea 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuántos menores.

Para resolver este problema se requiere tres contadores:

aprobados (Cuenta la cantidad de alumnos aprobados).
reprobados (Cuenta la cantidad de alumnos reprobados).
f (es el contador del for).

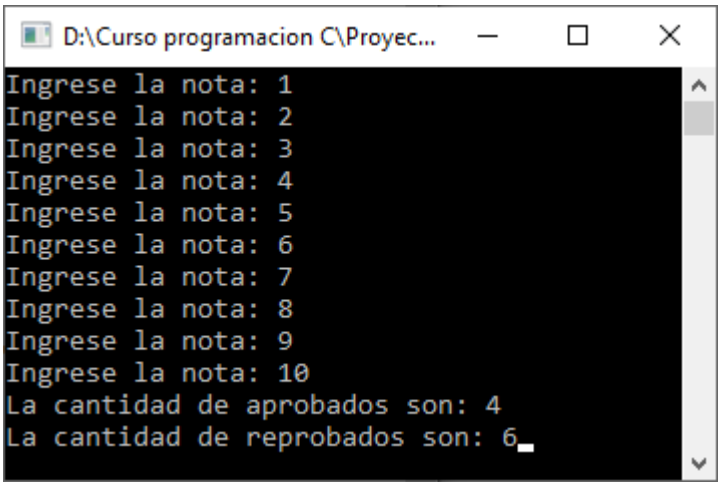
Dentro de la estructura repetitiva debemos hacer la carga de la variable nota y verificar con una estructura condicional si el contenido de la variable nota es mayor o igual a 7 para incrementar el contador aprobados, en caso de que la condición retorne falso debemos incrementar la variable reprobados.

Diagrama de flujo:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto42
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int nota, f, aprobados, reprobados;
            string line;
            aprobados = 0;
            reprobados = 0;
            for (f=1; f<=10; f++)
            {
                Console.Write("Ingrese la nota: ");
                line = Console.ReadLine();
                nota = int.Parse(line);
                if (nota>=7)
                {
                    aprobados = aprobados + 1;

                }
                else
                {
                    reprobados = reprobados + 1;
                }
            }
            Console.Write("La cantidad de aprobados son: ");
            Console.WriteLine(aprobados);
            Console.Write("La cantidad de reprobados son: ");
            Console.Write(reprobados);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

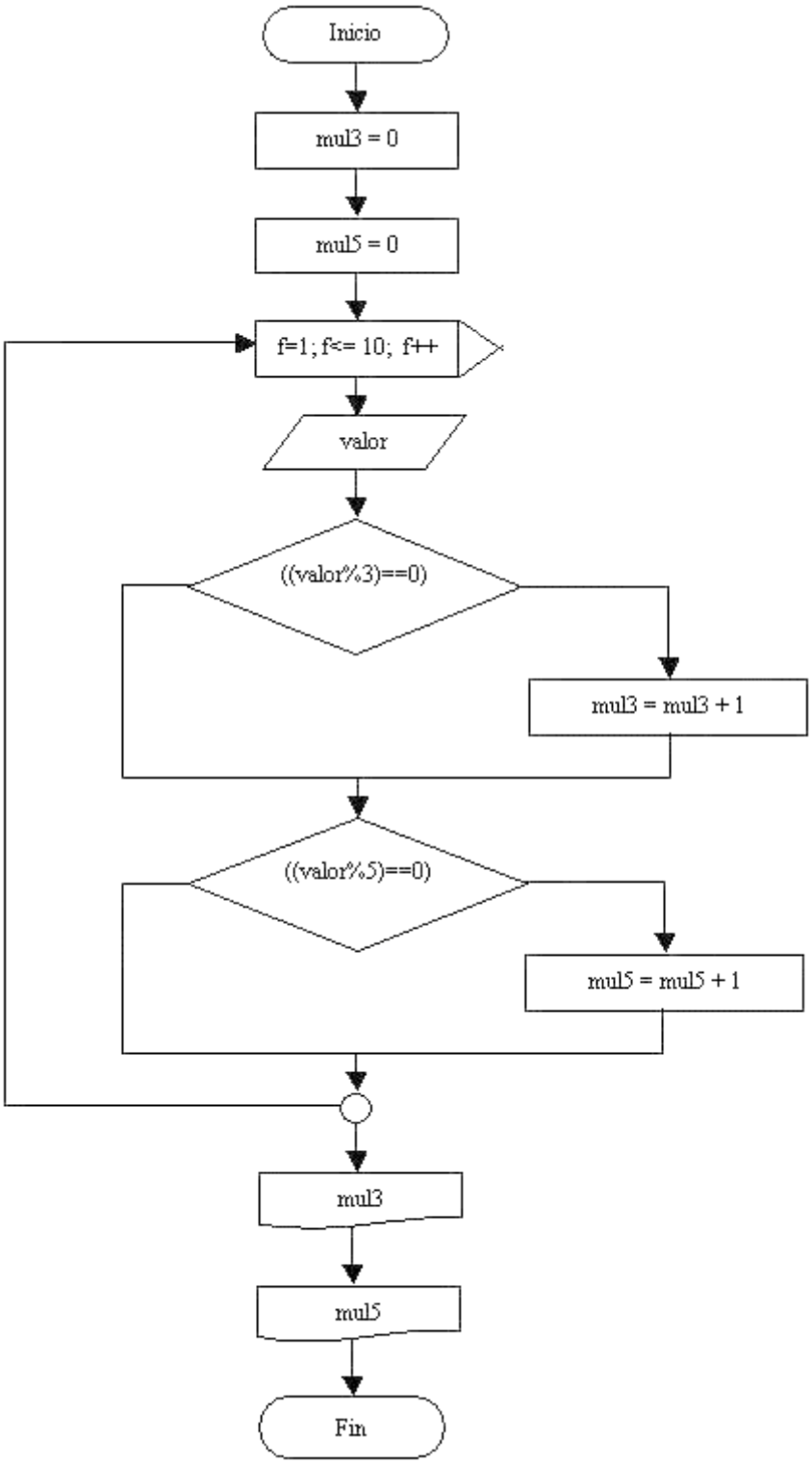


Capítulo 43.- Estructura repetitiva for – 4

Problema

Escribir un programa que lea 10 números enteros y luego muestre cuántos valores ingresados fueron múltiplos de 3 y cuántos de 5. Debemos tener en cuenta que hay números que son múltiplos de 3 y de 5 a la vez.

Diagrama de flujo:



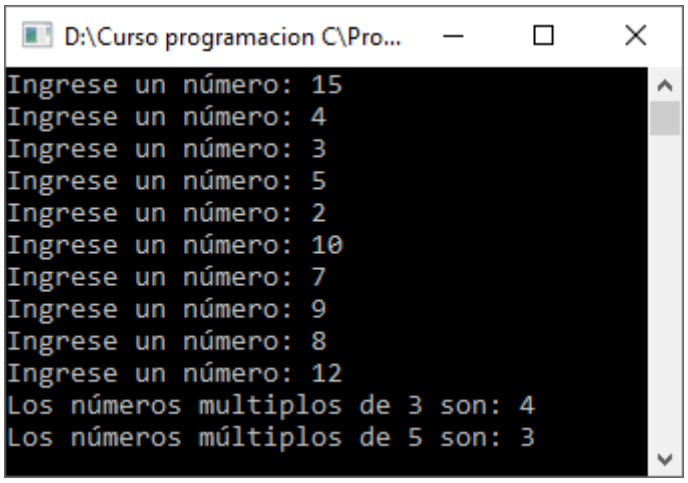
Tengamos en cuenta que el operador matemático % retorna el resto de dividir un valor por otro, en caso: valor%3 retorna el resto de dividir el valor que ingresamos por teclado, por tres.

Veamos si ingresamos 6 el resto de dividirlo por 3 es 0, si ingresamos 12 el resto de dividirlo por 3 es 0. Generalizando: cuando el resto de dividir por 3 al que ingresamos por teclado es cero, se trata de un múltiplo de dicho valor.

Ahora bien ¿por qué no hemos dispuesto una estructura if anidada? Porque hay valores que son múltiplos de 3 y de 5 a la vez. Por lo tanto con if anidado no podríamos analizar los dos casos. Es importante darse cuenta cuando conviene emplear if anidados y cuando no debe emplearse.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto43
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int x, numero, mult3, mult5;
            string line;
            mult3 = 0;
            mult5 = 0;
            for (x=1;x<=10; x++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                numero = int.Parse(line);
                if(numero%3==0)
                {
                    mult3 = mult3 + 1;
                }
                if(numero%5==0)
                {
                    mult5 = mult5 + 1;
                }
            }
            Console.Write("Los números multiplos de 3 son: ");
            Console.WriteLine(mult3);
            Console.Write("Los números múltiplos de 5 son: ");
            Console.Write(mult5);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Curso programacion C\Pro...
Ingrese un número: 15
Ingrese un número: 4
Ingrese un número: 3
Ingrese un número: 5
Ingrese un número: 2
Ingrese un número: 10
Ingrese un número: 7
Ingrese un número: 9
Ingrese un número: 8
Ingrese un número: 12
Los números multiplos de 3 son: 4
Los números múltiplos de 5 son: 3

Capítulo 44.- Estructura repetitiva for – 5

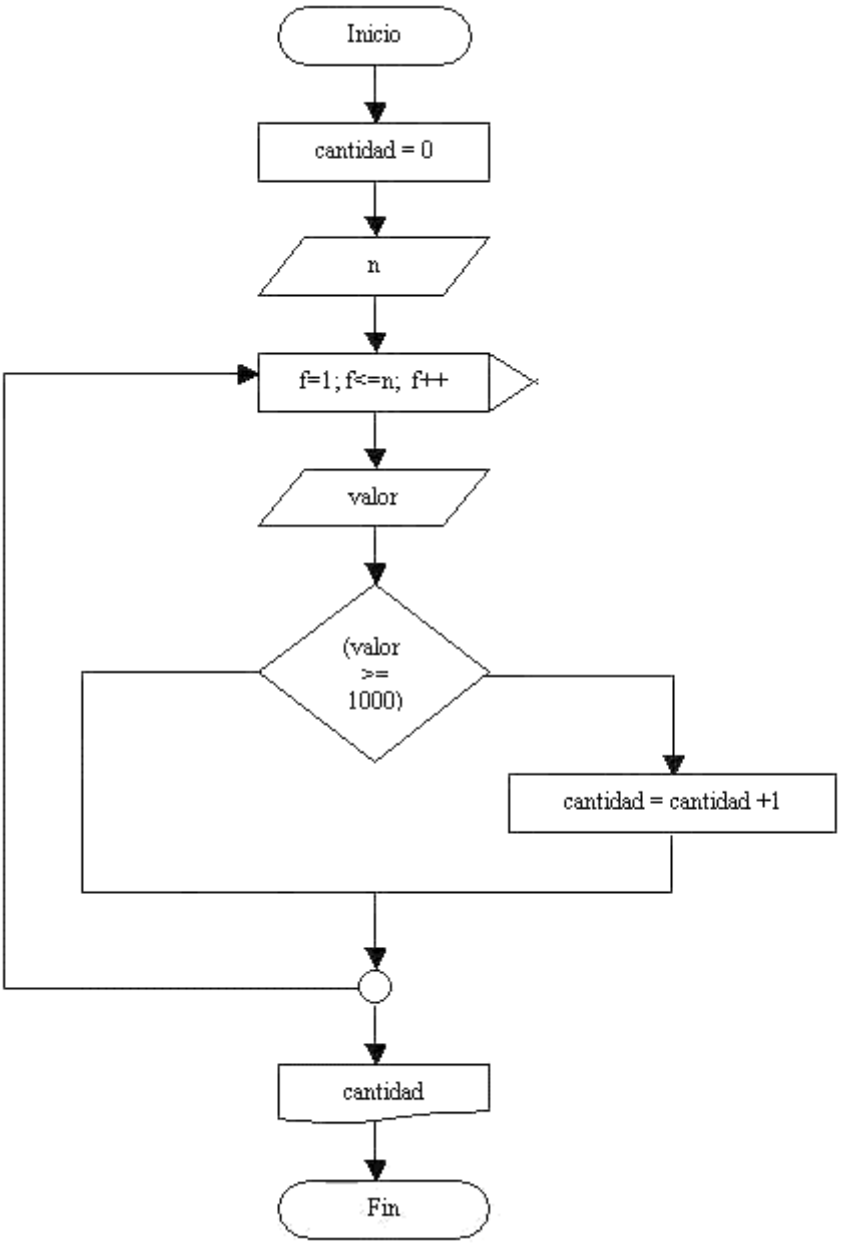
Problema

Escribir un programa que lea n números enteros y calcule la cantidad de valores mayores o iguales a 1000.

Este tipo de problema también se puede resolver empleando la estructura repetitiva for. Lo primero que se hace es cargar una variable que indique la cantidad de valores a ingresar. Dicha variable se carga antes de entrar en la estructura for.

La estructura for permite que el valor inicial o final dependa de una variable cargada previamente por teclado.

Diagrama de flujo:

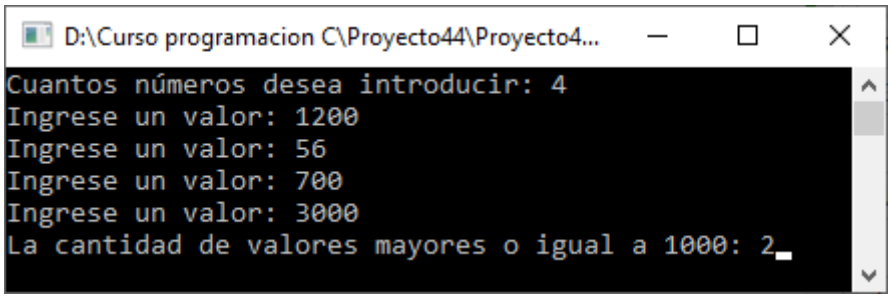


Tenemos un contador llamado cantidad y f que es el contador del for.

La variable entera n se carga previo el inicio del for, por lo que podemos fijar el valor final del for con la variable n.
Por ejemplo si el operador carga 5 en n la estructura repetitiva se ejecutará 5 veces.
La variable valor se ingresa dentro de la estructura repetitiva, y se verifica si el valor de la misma es mayor o igual a 1000, en dicho caso se incrementa en uno el contador cantidad.
Fuera de la estructura repetitiva imprimimos el contador cantidad que tiene almacenado la cantidad de valores ingresados mayores o iguales a 1000.

```
using System;
using System.Collections.Generic;
using System.Diagnostics.Eventing.Reader;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto44
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, n, cantidad, valor;
            string line;
            cantidad = 0;
            Console.Write("Cuantos números desea introducir: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            for (f=1; f<=n; f++)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                valor = int.Parse(line);
                if (valor>=1000)
                {
                    cantidad = cantidad + 1;
                }
            }
            Console.Write("La cantidad de valores mayores o igual a 1000: ");
            Console.Write(cantidad);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 45.- Estructura repetitiva for – 6

Problema propuesto

Confeccionar un programa que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programa deberá informar:

a) De cada triángulo la medida de su base, su altura y su superficie.

b) La cantidad de triángulos cuya superficie es mayor a 12.

```
namespace Proyecto45
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int n, f, ba, al, superficie, cantidad;
            string line;
            cantidad = 0;
            Console.Write("Cuantos triangulos vamos a ingresar: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            for (f=1; f<=n; f++)
            {
                Console.Write("Ingrese la base del triángulo: ");
                line= Console.ReadLine();
                ba=int.Parse(line);
                Console.Write("Ingrese la altua del triángulo: ");
                line=Console.ReadLine();
                al=int.Parse(line);
                superficie = ba * al / 2;
                Console.Write("La superficie es: ");
                Console.WriteLine(superficie);
                if (superficie>12)
                {
                    cantidad = cantidad + 1;
                }
            }
            Console.Write("La cantidad de triángulos con superficie mayor a 12 es ");
            Console.Write(cantidad);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

```
D:\Curso programacion C\Proyecto45\Proyecto45\bin\Debu...
Cuantos triangulos vamos a ingresar: 4
Ingrese la base del triángulo: 3
Ingrese la altua del triángulo: 5
La superficie es: 7
Ingrese la base del triángulo: 10
Ingrese la altua del triángulo: 2
La superficie es: 10
Ingrese la base del triángulo: 10
Ingrese la altua del triángulo: 20
La superficie es: 100
Ingrese la base del triángulo: 14
Ingrese la altua del triángulo: 18
La superficie es: 126
La cantidad de triángulos con superficie mayor a 12 es 2
```

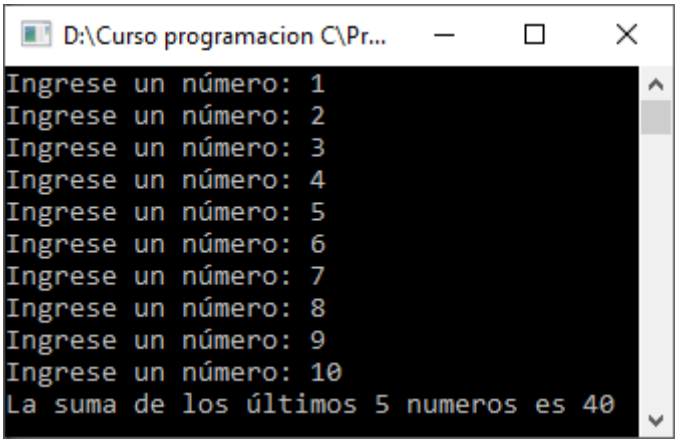
Capítulo 46.- Estructura repetitiva for – 7

Problema propuesto

Desarrolla un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto46
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, num, suma;
            string line;
            suma = 0;
            for (f=1; f<=10;  f++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                num = int.Parse(line);
                if (f > 5)
                {
                    suma = suma + num;
                }

            }
            Console.Write("La suma de los últimos 5 numeros es ");
            Console.Write(suma);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Otra posible solución puede ser:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto46
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, num, suma;
            string line;
            suma = 0;
            for (f=1; f<=5;  f++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                num = int.Parse(line);
            }
            for (f = 1; f <= 5; f++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                num = int.Parse(line);
                suma = suma + num;
            }
            Console.Write("La suma de los últimos 5 numeros es ");
            Console.Write(suma);
            Console.ReadKey();
        }
    }
}
```

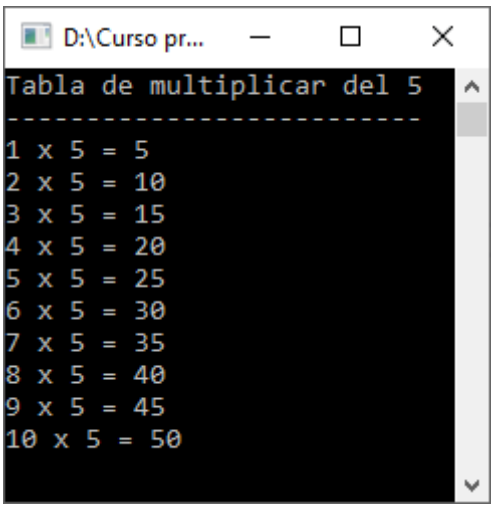
Capítulo 47.- Estructura repetitiva for – 8

Problema propuesto

Desarrollar un programa que muestre la tabla de multiplicar del 5 (del 5 al 50).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto47
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, tabla, total;
            tabla = 5;
            Console.Write("Tabla de multiplicar del ");
            Console.WriteLine(tabla);
            Console.WriteLine("-------------------------");
            for (f=1; f<=10;  f++)
            {
                Console.Write(f);
                Console.Write(" x ");
                Console.Write(tabla);
                Console.Write(" = ");
                total = f * tabla;
                Console.WriteLine(total);
            }

            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 48.- Estructura repetitiva for – 9

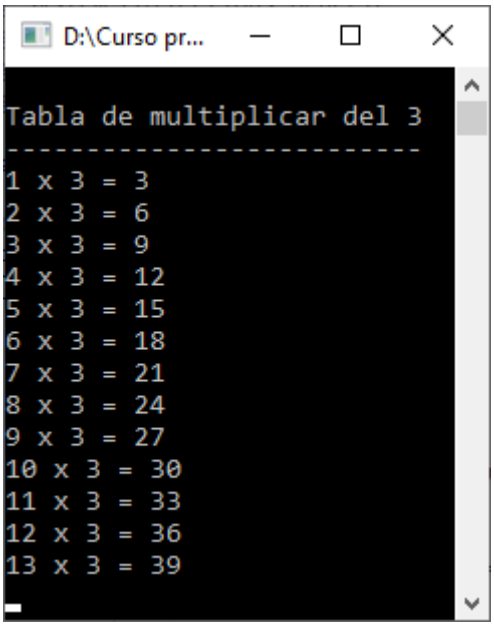
Problema propuesto

Confeccionar un programa que permita ingresar un valor del 1 al 10 y nos muestre la tabla de multiplicar del mismo (los primeros 13 términos.

Ejemplo: Si ingreso 3 deberá mostrar la tabla del 3 hasta el resultado 39.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto48
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, tabla, total;
            string line;
            Console.Write("Que tabla desea consultar: ");
            line = Console.ReadLine();
            tabla = int.Parse(line);
            if (tabla >= 1 && tabla <=10)
            {
                Console.Write("Tabla de multiplicar del ");
                Console.WriteLine(tabla);
                Console.WriteLine("--------------------------");
                for (f = 1; f <= 13; f++)
                {
                    Console.Write(f);
                    Console.Write(" x ");
                    Console.Write(tabla);
                    Console.Write(" = ");
                    total = f * tabla;
                    Console.WriteLine(total);
                }
            }
            else
            {
                Console.WriteLine("Ingresó un valor fuera de rango");
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Curso pr...
Tabla de multiplicar del 3
1 x 3 = 3
2 x 3 = 6
3 x 3 = 9
4 x 3 = 12
5 x 3 = 15
6 x 3 = 18
7 x 3 = 21
8 x 3 = 24
9 x 3 = 27
10 x 3 = 30
11 x 3 = 33
12 x 3 = 36
13 x 3 = 39

Capítulo 49.- Estructura repetitiva for – 10

Problema propuesto

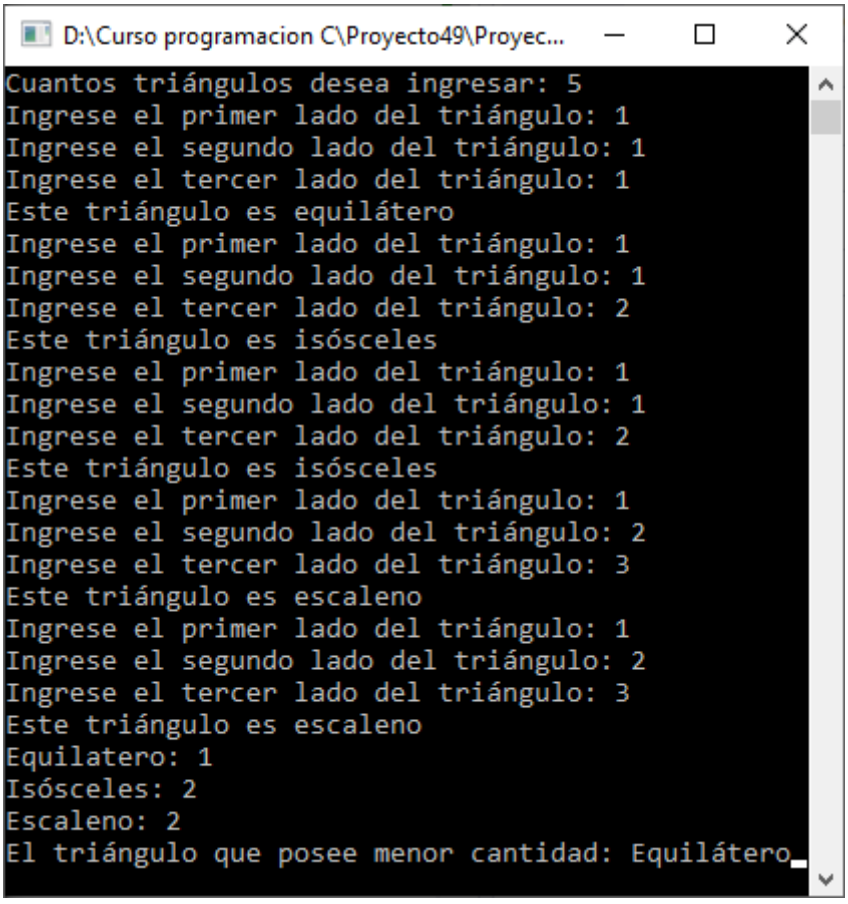
Realizar un programa que lea los lados de n triángulos, e informar:

a) De cada uno de ellos, que tipo de triángulo es: equilátero (tres lados iguales), isósceles (dos lados iguales), o escaleno (ningún lado igual).
b) Cantidad de triángulos de cada tipo.
c) Tipo de triángulo que posee menor cantidad.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto49
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int n, f, lado1, lado2, lado3;
            int equi, isos, esca;
            string line;
            equi = 0;
            isos = 0;
            esca = 0;
            Console.Write("Cuantos triángulos desea ingresar: ");
            line = Console.ReadLine();
            n=int.Parse(line);
            for (f=1; f<=n; f++)
            {
                Console.Write("Ingrese el primer lado del triángulo: ");
                line= Console.ReadLine();
                lado1 = int.Parse(line);
                Console.Write("Ingrese el segundo lado del triángulo: ");
                line = Console.ReadLine();
                lado2 = int.Parse(line);
                Console.Write("Ingrese el tercer lado del triángulo: ");
                line = Console.ReadLine();
                lado3 = int.Parse(line);
                if(lado1 == lado2 && lado1==lado3)
                {
                    Console.WriteLine("Este triángulo es equilátero");
                    equi=equi+1;
                }
                else
                {
                    if(lado1==lado2 || lado1==lado3 || lado2==lado3)
                    {
                        Console.WriteLine("Este triángulo es isósceles");
                        isos=isos+1;
                    }
                    else
                    {
```

```
                    Console.WriteLine("Este triángulo es escaleno");
                    esca=esca+1;
                }
            }
        }
        Console.Write("Equilatero: ");
        Console.WriteLine(equi);
        Console.Write("Isósceles: ");
        Console.WriteLine(isos);
        Console.Write("Escaleno: ");
        Console.WriteLine(esca);
        if(equi<isos && equi<esca)
        {
            Console.Write("El triángulo que posee menor cantidad: Equilátero");
        }
        else
        {
            if(isos<esca)
            {
                Console.Write("El triángulo que posee menor cantidad: Isósceles");
            }
            else
            {
                Console.Write("El triángulo que posee menor cantidad: Escaleno");
            }
        }

        Console.ReadKey();

    }
}
}
```

Si ejecutamos este será el resultado:

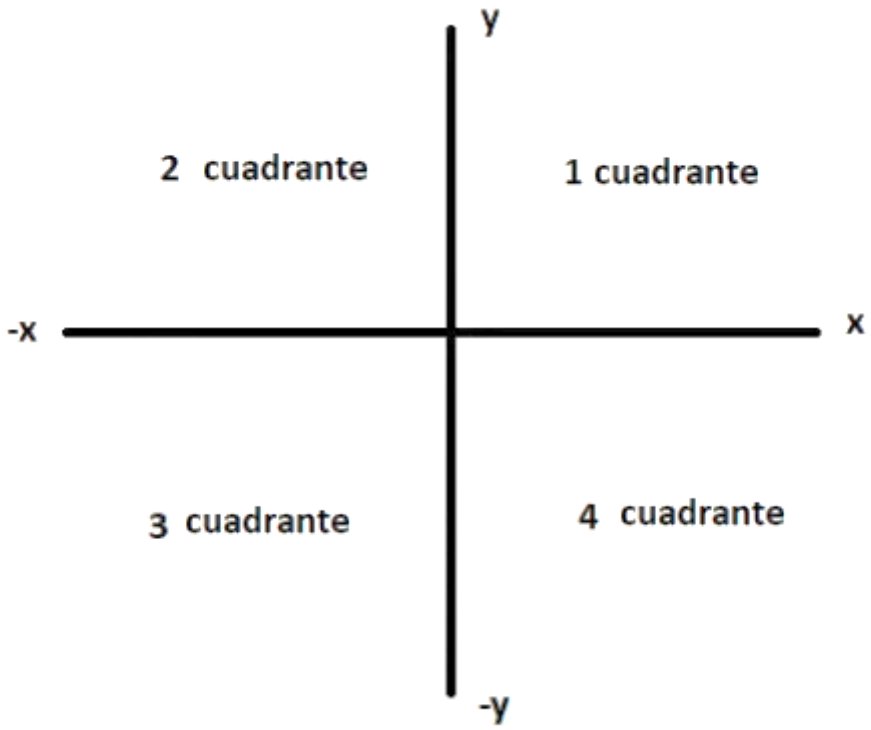


Capítulo 50.- Estructura repetitiva for – 11

Problema propuesto

Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano.

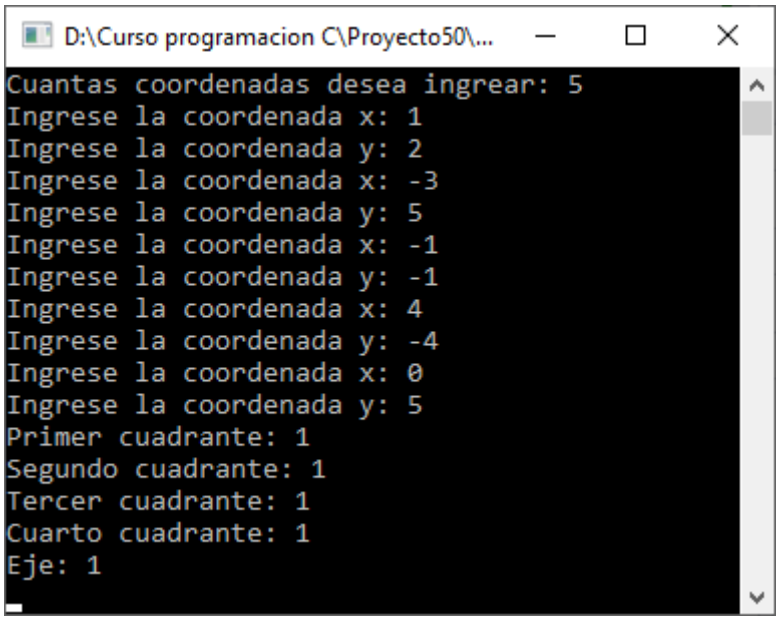
Informar cuántos puntos se han ingresado en el primer, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que se ingrese la cantidad de puntos a procesar.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto50
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, n, x, y;
            int primer, segundo, tercero, cuarto, eje;
            primer = 0;
            segundo = 0;
            tercero = 0;
            cuarto = 0;
            eje = 0;
            string line;
            Console.Write("Cuantas coordenadas desea ingrear: ");
```

```
            line = Console.ReadLine();
            n=int.Parse(line);
            for (f=1; f<=n; f++)
            {
                Console.Write("Ingrese la coordenada x: ");
                line = Console.ReadLine();
                x= int.Parse(line);
                Console.Write("Ingrese la coordenada y: ");
                line= Console.ReadLine();
                y= int.Parse(line);
                if (x>0 && y>0)
                {
                    primer = primer + 1;
                }
                else
                {
                    if(x<0 && y>0)
                    {
                        segundo = segundo + 1;

                    }
                    else
                    {
                        if(x<0 && y<0)
                        {
                            tercero= tercero + 1;
                        }
                        else
                        {
                            if (x>0 && y<0)
                            {
                                cuarto = cuarto + 1;
                            }else
                            {
                                eje= eje + 1;
                            }
                        }

                    }
                }

            }
            Console.Write("Primer cuadrante: ");
            Console.WriteLine(primer);
            Console.Write("Segundo cuadrante: ");
            Console.WriteLine(segundo);
            Console.Write("Tercer cuadrante: ");
            Console.WriteLine(tercero);
            Console.Write("Cuarto cuadrante: ");
            Console.WriteLine(cuarto);
            Console.Write("Eje: ");
            Console.WriteLine(eje);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 51.- Estructura repetitiva for – 12

Problema propuesto

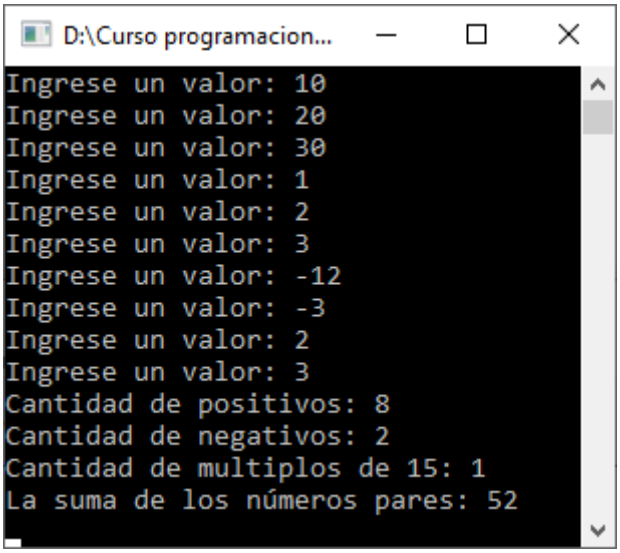
Se realiza la carga de 10 valores por teclado. Se desea conocer:

a) La cantidad de valores ingresados negativos.
b) La cantidad de valores ingresados positivos.
c) La cantidad de múltiplos de 15.
d) El valor acumulado de los números ingresados que son pares.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto51
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, numero,  negativos, positivos, mult15, totPares;
            negativos = 0;
            positivos = 0;
            mult15 = 0;
            totPares = 0;
            string line;
            for (f=1; f<=10; f++)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                numero = int.Parse(line);
                if (numero>0)
                {
                    positivos = positivos + 1;
                }
                else
                {
                    if (numero<0)
                    {
                        negativos = negativos + 1;
                    }
                }
                if(numero%15==0)
                {
                    mult15 = mult15 + 1;
                }
                if(numero%2==0)
                {
                    totPares = totPares + numero;
                }
            }
```

```
            Console.Write("Cantidad de positivos: ");
            Console.WriteLine(positivos);
            Console.Write("Cantidad de negativos: ");
            Console.WriteLine(negativos);
            Console.Write("Cantidad de multiplos de 15: ");
            Console.WriteLine(mult15);
            Console.Write("La suma de los números pares: ");
            Console.WriteLine(totPares);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 52.- Estructura repetitiva for – 13

Problema propuesto

Se cuenta con la siguiente información:

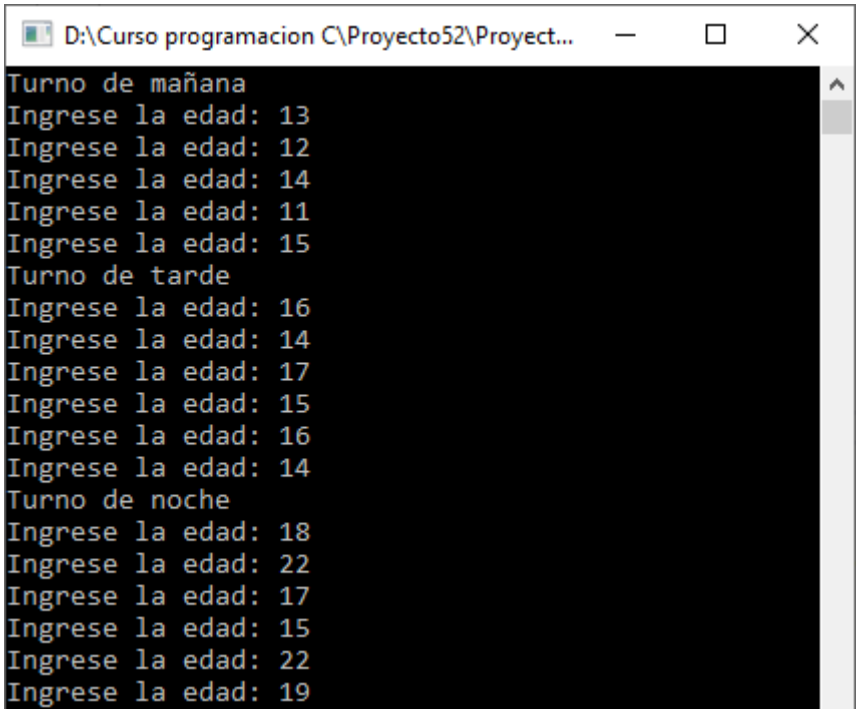
Las edades de 50 estudiantes del turno de mañana.
Las edades de 60 estudiantes del turno de tarde.
Las edades de 110 estudiantes del turno de noche.
Las edades de cada estudiante deben ingresarse por teclado.

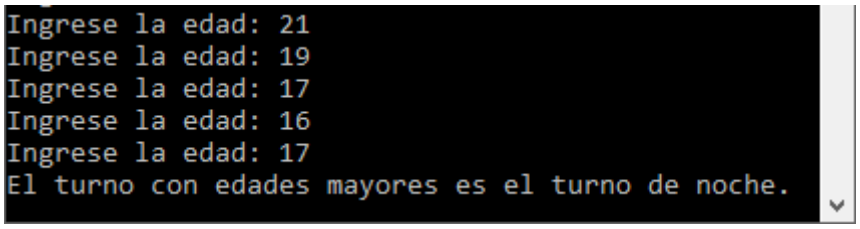
a) Obtener el promedio de las edades de cada turno (tres promedios).
b) Imprimir dichos promedios (promedio de cada turno).
c) Mostrar por pantalla un mensaje que indique cual de los tres turnos tienen un promedio de edades mayor.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Net.Http.Headers;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto52
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int f, edad;
            int suMa, suTa, suNo;
            int proMa, proTa, proNo;
            String line;
            suMa = 0;
            suTa = 0;
            suNo = 0;
            proMa = 0;
            proTa = 0;
            proNo = 0;
            Console.WriteLine("Turno de mañana");
            for (f=1; f<=50; f++)
            {
                Console.Write("Ingrese la edad: ");
                line = Console.ReadLine();
                edad=int.Parse(line);
                suMa = suMa + edad;
            }
```

```
            Console.WriteLine("Turno de tarde");
            for (f=1; f<=60; f++)
            {
                Console.Write("Ingrese la edad: ");
                line = Console.ReadLine();
                edad = int.Parse(line);
                suTa = suTa + edad;
            }
            Console.WriteLine("Turno de noche");
            for (f=1; f<=110; f++)
            {
                Console.Write("Ingrese la edad: ");
                line = Console.ReadLine();
                edad = int.Parse(line);
                suNo = suNo + edad;
            }
            proMa = suMa / 50;
            proTa = suTa / 60;
            proNo = suNo / 110;

            if(proMa>proTa && proMa>proNo)
            {
                Console.Write("El turno con edades mayores es el turno de mañana.");
            }
            else
            {
                if(proTa>proNo)
                {
                    Console.Write("El turno con edades mayores es el turno de tarde.");
                }
                else
                {
                    Console.Write("El turno con edades mayores es el turno de noche.");
                }
            }
            Console.ReadKey();
        }
    }
}
```

Para realizar la prueba vamos a cambiar turno mañana con 5 alumnos, turno tarde con 6 alumnos y turno noche con 11 alumnos, cuando ejecutemos este será el resultado:




Ingrese la edad: 21
Ingrese la edad: 19
Ingrese la edad: 17
Ingrese la edad: 16
Ingrese la edad: 17
El turno con edades mayores es el turno de noche.

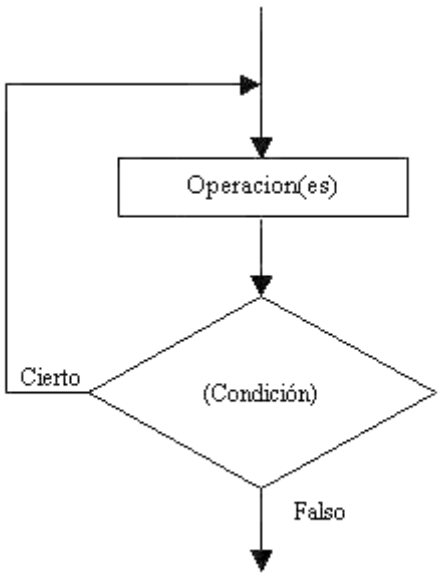
Capítulo 53.- Estructura repetitiva do while – 1

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podría no ejecutar el bloque.

Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

Representación gráfica:



El bloque de operaciones MIENTRAS que la condición sea Verdadera.

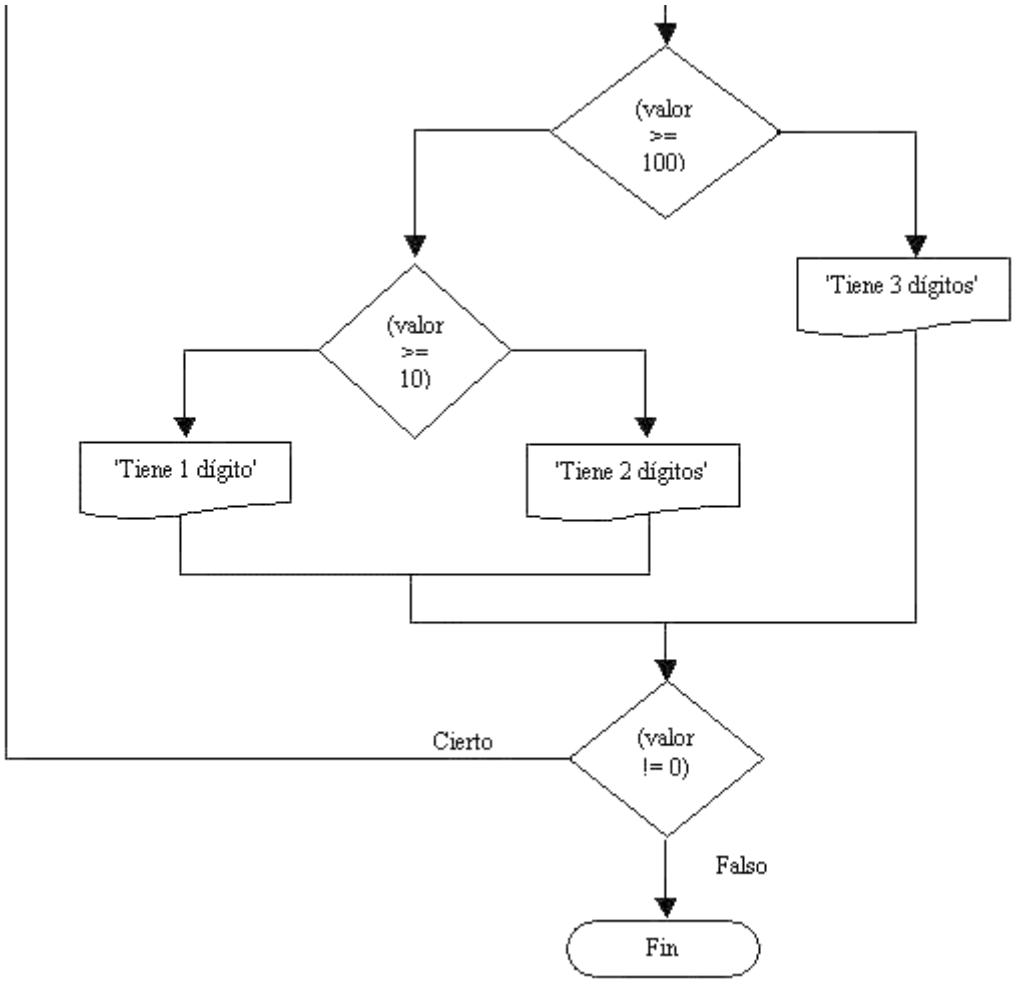
Si la condición retorna Falso el ciclo se detiene. En C#, todos los ciclos repiten por verdadero y cortan por falso.

Es importante analizar y ver las operaciones se ejecutan como mínimo una vez.

Problema

Escribir un programa que solicite la carga de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se cargue el valor 0.

Diagrama de flujo:



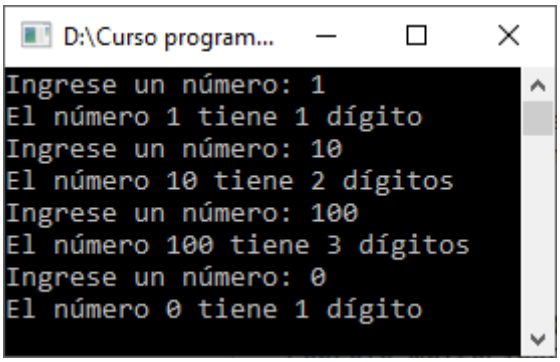
Ho hay que confundir los rombos de la estructuras condicionales con los de las estructuras repetitivas do while.

En este problema por lo menos se caga un valor. Si se carga un valor mayor o igual a 100 se trata un número de tres dígitos. so es mayor o igual a 10 se trata de un valor de dos dígitos, en caso contrario se trata un valor de un dígito. Este bloque se repite hasta que se ingresa en la variable valor el número 0 con lo que la condición de la estructura do while retorna falso y sale del bloque repetitivo finalizando el programa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto53
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num;
            string line;
```

```
            do
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                num =int.Parse(line);
                if(num>=100)
                {
                    Console.Write("El número ");
                    Console.Write(num);
                    Console.WriteLine(" tiene 3 dígitos");
                }
                else
                {
                    if(num>=10)
                    {
                        Console.Write("El número ");
                        Console.Write(num);
                        Console.WriteLine(" tiene 2 dígitos");
                    }
                    else
                    {
                        Console.Write("El número ");
                        Console.Write(num);
                        Console.WriteLine(" tiene 1 dígito");
                    }
                }
            } while (num != 0);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



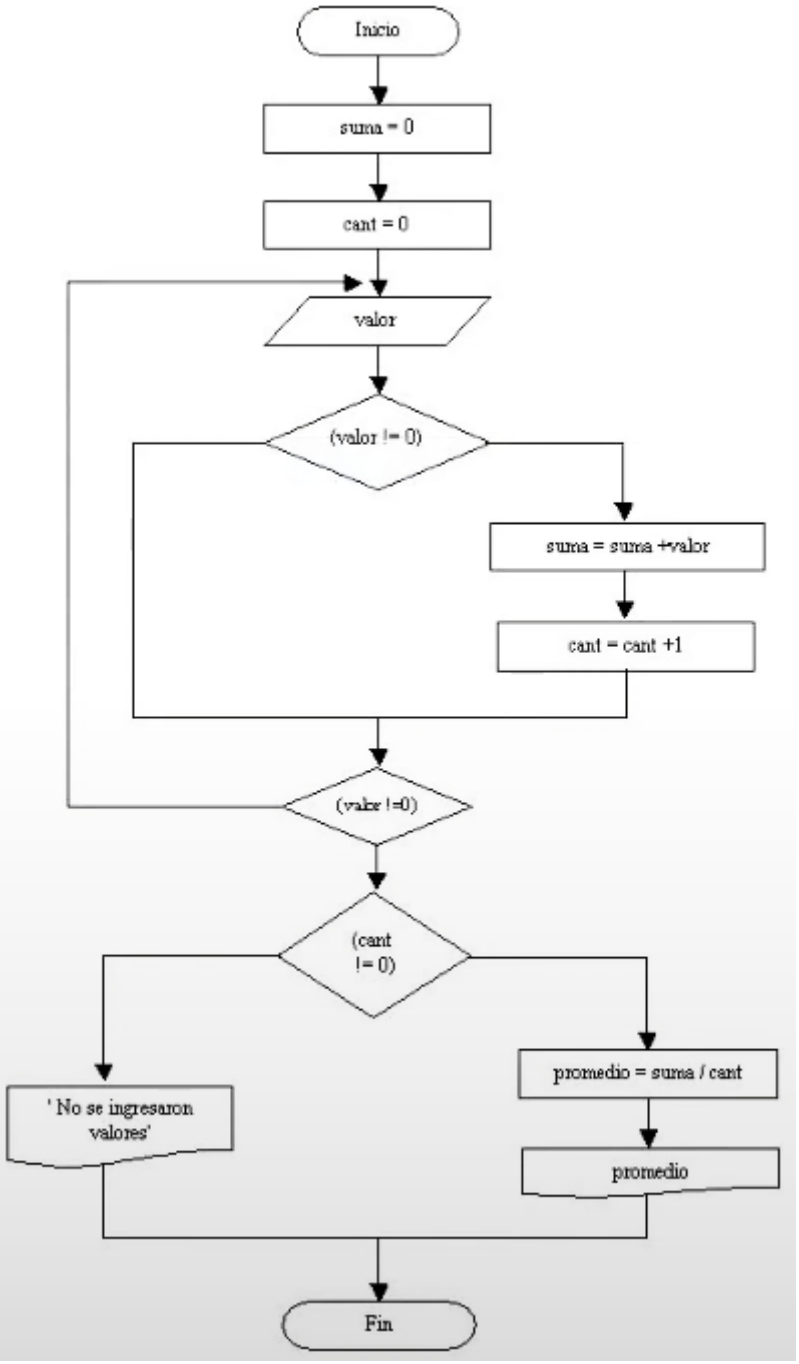
Capítulo 54.- Estructura repetitiva do while – 2

Problema

Escribir un programa que solicite la carga de números por teclado, obtener su promedio. Finalizar la carga de valores cuando se cargue el valor 0.

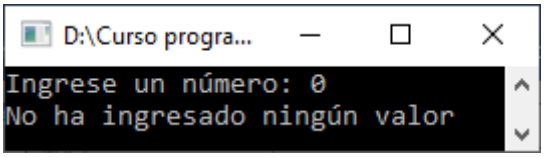
Cuando la finalización depende de algún valor ingresado por el operador conviene el empleo de la estructura do while, por lo menos se cargará un valor (en el caso más extremo se carga 0, que indica la finalización de la carga de valores).

Diagrama de flujo:

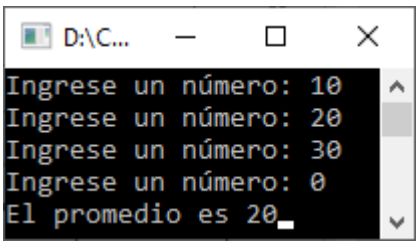


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto54
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num, cant, suma, promedio;
            string line;
            cant = 0;
            suma = 0;
            do
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                num = int.Parse(line);
                if (num!=0)
                {
                    suma = suma + num;
                    cant = cant + 1;
                }
            }while (num!=0);
                  if (cant!=0)
            {
                promedio = suma / cant;
                Console.Write("El promedio es ");
                Console.Write(promedio);
            }else
            {
                Console.Write("No ha ingresado ningún valor");
            }
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar contestando con 0 en el primer valor:



Ejecutamos de nuevo introduciendo varios valores antes de introducir el 0.



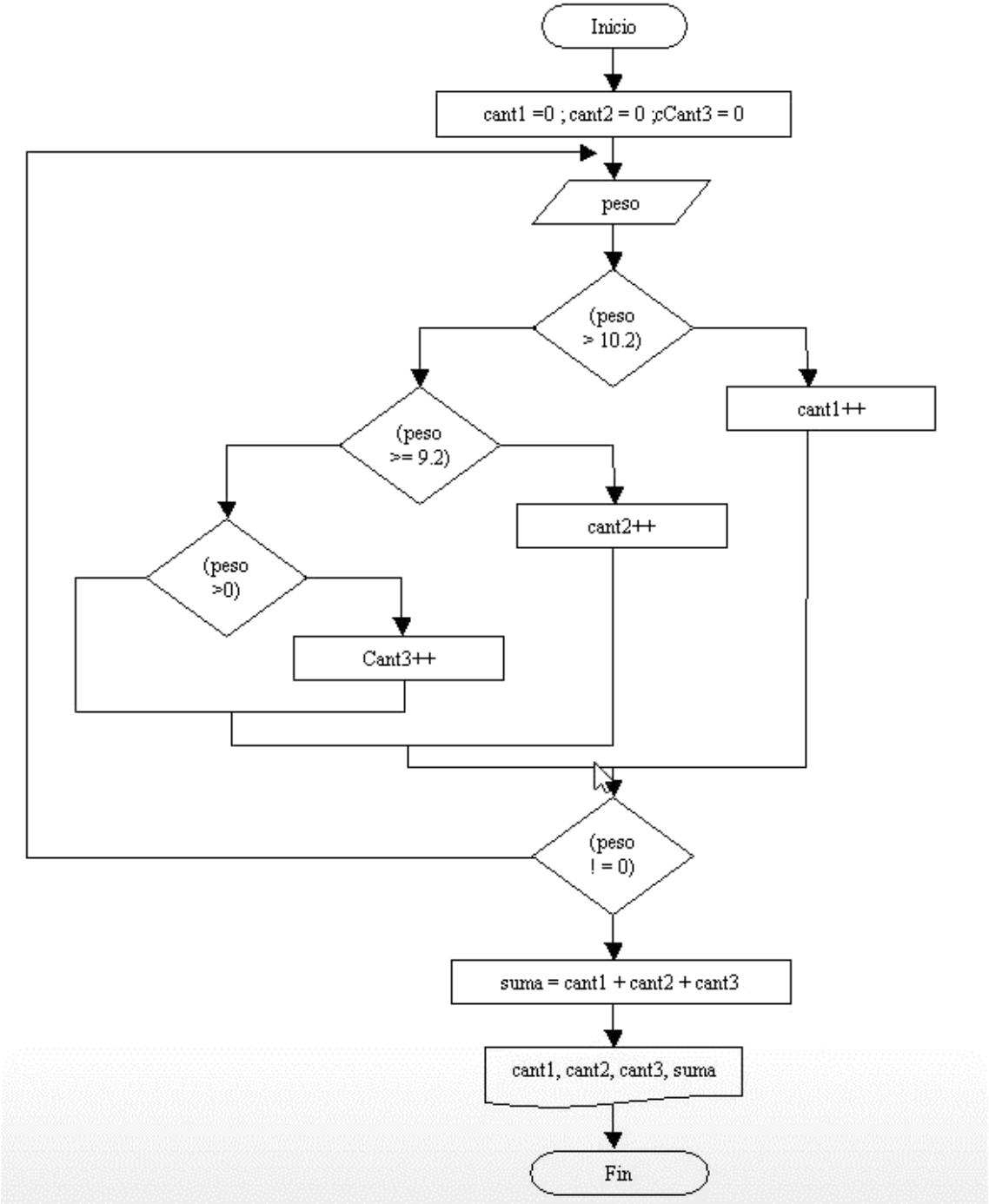
Capítulo 55.- Estructura repetitiva do while – 3

Problema

Realizar un programa que permita ingresar el peso (en kilogramos) de piezas. El proceso termina cuando ingresamos el valor 0. Se debe informar:

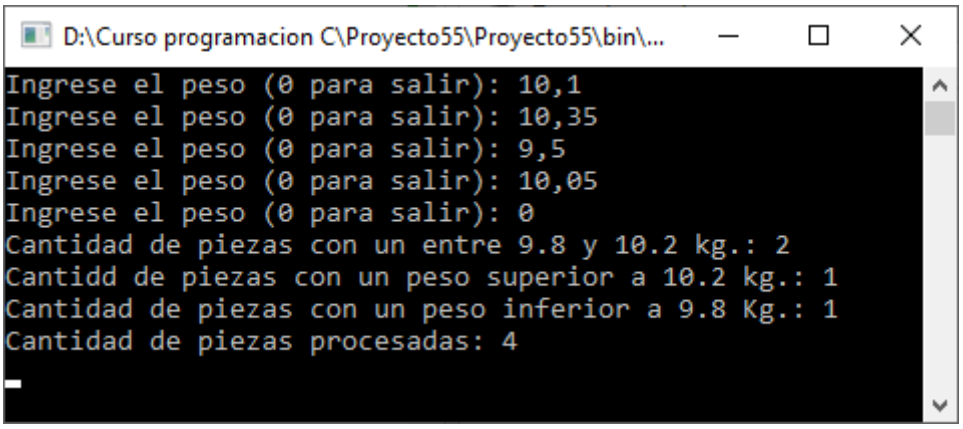
a) Cuantas piezas tienen un peso entre 9.8 kg. y 10.2 Kg.?, cuantas con más de 10.2 Kg.? y cuantas con menos de 9.8 Kg.?.
b) La cantidad de piezas procesadas.

Diagrama de flujo:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto55
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int cant1, cant2, cant3, suma;
            float peso;
            string line;
            cant1 = 0;
            cant2 = 0;
            cant3 = 0;
            suma = 0;
            do
            {
                Console.Write("Ingrese el peso (0 para salir): ");
                line = Console.ReadLine();
                peso=float.Parse(line);
                if(peso!=0)
                {
                    if (peso>=9.8 && peso <=10.2)
                    {
                        cant1=cant1+1;
                    }
                    else
                    {
                        if (peso >10.2)
                        {
                            cant2=cant2+1;
                        }
                        else
                        {
                            cant3=cant3+1;
                        }
                    }
                }
            }while (peso!=0);
            Console.Write("Cantidad de piezas con un entre 9.8 y 10.2 kg.: ");
            Console.WriteLine(cant1);
            Console.Write("Cantidd de piezas con un peso superior a 10.2 kg.: ");
            Console.WriteLine(cant2);
            Console.Write("Cantidad de piezas con un peso inferior a 9.8 Kg.: ");
            Console.WriteLine(cant3);
            suma=cant1+cant2+ cant3;
            Console.Write("Cantidad de piezas procesadas: ");
            Console.WriteLine(suma);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Curso programacion C\Proyecto55\Proyecto55\bin\...
Ingrese el peso (0 para salir): 10,1
Ingrese el peso (0 para salir): 10,35
Ingrese el peso (0 para salir): 9,5
Ingrese el peso (0 para salir): 10,05
Ingrese el peso (0 para salir): 0
Cantidad de piezas con un entre 9.8 y 10.2 kg.: 2
Cantidd de piezas con un peso superior a 10.2 kg.: 1
Cantidad de piezas con un peso inferior a 9.8 Kg.: 1
Cantidad de piezas procesadas: 4

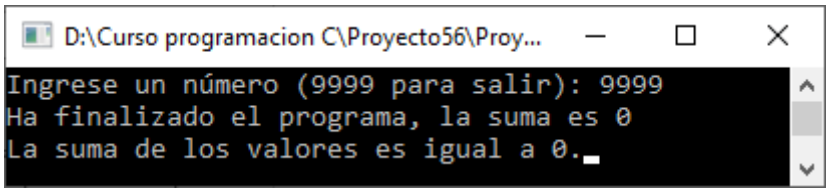
Capítulo 56.- Estructura repetitiva do while – 4

Problema propuesto

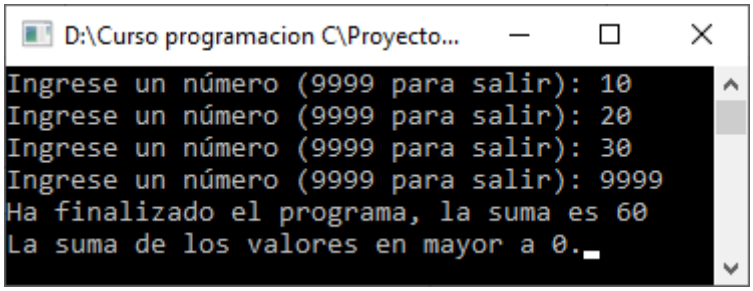
Realizar un programa que acumule (sume) valores ingresados por teclado hasta ingresar el 9999 (no sumar dicho valor, indicar que ha finalizado el programa). Imprimir el valor acumulado e informar si dicho valor es cero, mayor a cero o menor a cero.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto56
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int num, suma;
            string line;
            suma = 0;
            do
            {
                Console.Write("Ingrese un número (9999 para salir): ");
                line = Console.ReadLine();
                num=int.Parse(line);
                if (num!=9999)
                {
                    suma = suma + num;
                }

            } while (num != 9999);
            Console.Write("Ha finalizado el programa, la suma es ");
            Console.WriteLine(suma);
            if(suma==0)
            {
                Console.Write("La suma de los valores es igual a 0.");
            }
            else
            {
                if (suma>0)
                {
                    Console.Write("La suma de los valores en mayor a 0.");
                }
                else
                {
                    Console.Write("La suma de los valores en menor a 0.");
                }
            }
            Console.ReadKey();
        }
    }
}
```

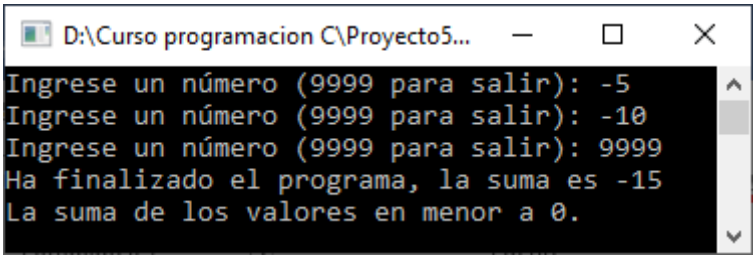
Vamos a ejecutar y de primeras introduciremos el valor 9999.



Ejecutamos de nuevo introduciendo 3 valores positivos antes de ingresar el valor 9999.



Ejecutamos de nuevo introduciendo dos valores negativos antes de ingresar el valor 9999.



Capítulo 57.- Estructura repetitiva do while – 5

Problema propuesto

En un bando se procesan datos de las cuentas corrientes de sus clientes. De cada cuenta corriente se conoce: Número de cuenta y saldo actual. El ingreso de datos debe finalizar al ingresar un valor negativo en el número de cuenta.

Se pide confeccionar un programa que lea los datos de las cuentas corrientes e informe:

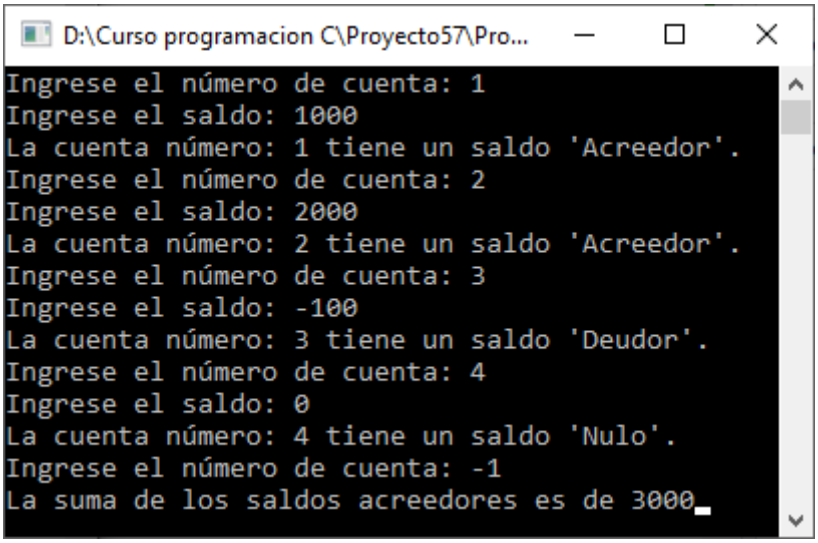
a) De cada cuenta: número de cuenta y el estado de la cuenta según su saldo, sabiendo que:
   'Acreedor' Si el saldo es > 0.
   'Deudor' Si el saldo es < 0.
   'Nulo' Si el saldo es = 0.

b) La suma total de los saldos acreedores.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto57
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int cuenta, saldo, suma;
            string line;
            suma = 0;
            do
            {
                Console.Write("Ingrese el número de cuenta: ");
                line = Console.ReadLine();
                cuenta = int.Parse(line);
                if (cuenta > 0)
                {
                    Console.Write("Ingrese el saldo: ");
                    line = Console.ReadLine();
                    saldo = int.Parse(line);
                    if (saldo > 0)
                    {
                        suma = suma + saldo;
                        Console.Write("La cuenta número: ");
                        Console.Write(cuenta);
                        Console.WriteLine(" tiene un saldo 'Acreedor'.");
                    }
                    else
                    {
                        if (saldo < 0)
                        {
```

```
                    Console.Write("La cuenta número: ");
                    Console.Write(cuenta);
                    Console.WriteLine(" tiene un saldo 'Deudor'.");
                }
                else
                {
                    Console.Write("La cuenta número: ");
                    Console.Write(cuenta);
                    Console.WriteLine(" tiene un saldo 'Nulo'.");
                }
            }

        }

        } while (cuenta > 0);
        Console.Write("La suma de los saldos acreedores es de ");
        Console.Write(suma);
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 58.- Cadena de caracteres

En C# hemos visto que cuando queremos almacenar un valor entero definimos una variable de tipo int, si queremos almacenar un valor con decimales definimos una variable de tipo float. Ahora si queremos almacenar una cadena de caracteres (por ejemplo un nombre de una persona) debemos definir una variable de tipo string.

En realidad hemos estado utilizando en todos los problemas planteados desde el principio la definición de una variable de tipo string donde almacenamos cualquier dato que carga el operador por teclado, esto debido a que la clase Console tiene el método ReadLine que caraga un string.

Más adelante veremos en profundidad y detenimiento los conceptos del manejo de string, por ahora solo nos interesa la mecánica para trabajar con cadena de caracteres.

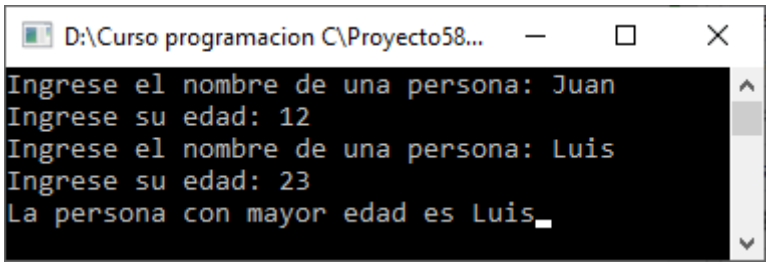
Problema 1:

Solicitar el ingreso del nombre y edad de dos personas. Mostrar el nombre e la persona con mayor edad.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web;

namespace Proyecto58
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            int edad1, edad2;
            string nom1, nom2;
            string line;
            Console.Write("Ingrese el nombre de una persona: ");
            nom1 = Console.ReadLine();
            Console.Write("Ingrese su edad: ");
            line = Console.ReadLine();
            edad1 = int.Parse(line);
            Console.Write("Ingrese el nombre de una persona: ");
            nom2 = Console.ReadLine();
            Console.Write("Ingrese su edad: ");
            line = Console.ReadLine();
            edad2 = int.Parse(line);
            if (edad1 > edad2)
            {
                Console.Write("La persona con mayor edad es ");
                Console.Write(nom1);
            }
            else
            {
```

```
            Console.Write("La persona con mayor edad es ");
            Console.Write(nom2);
        }
        Console.ReadKey();
    }
  }
}
```

Si ejecutamos este será el resultado:

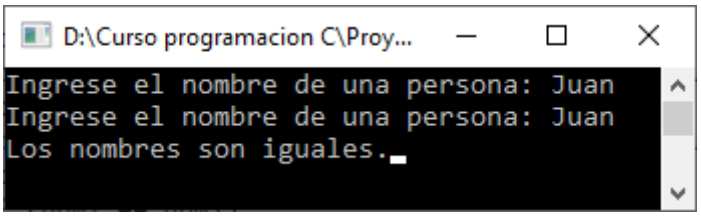


Problema 2:

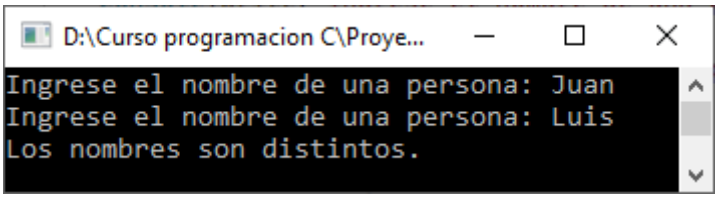
Solicitar el ingreso de dos nombres de personas. Mostrar un mensaje si son iguales o distintos.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto59
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            string nom1, nom2;
            Console.Write("Ingrese el nombre de una persona: ");
            nom1 = Console.ReadLine();
            Console.Write("Ingrese el nombre de una persona: ");
            nom2 = Console.ReadLine();
            if (nom1 == nom2)
            {
                Console.Write("Los nombres son iguales.");
            }
            else
            {
                Console.Write("Los nombres son distintos.");
            }
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar introduciendo dos nombres iguales:



Vamos a ejecutar de nuevo introduciendo nombres distintos:



Capítulo 59.- Declaración de una clase y definición de objetos – 1

La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones).

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos crear una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

La estructura de una clase se:

```
class [nombre de la clase] {
  [atributos o variables de la clase]
  [métodos o funciones de la clase]
  [main]
```

Problema

Confeccionar una clase que permita cargar el nombre y edad de una persona. Mostrar los datos cargados. Imprimir un mensaje si es mayor de edad (edad>=18).

El nombre de la clase debe hacer referencia al concepto (en este caso la hemos llamado Persona):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto60
{
    0 referencias
    class Persona
    {
        private string nombre;
        private int edad;

        0 referencias
        static void Main(string[] args)
        {
        }
    }
}
```

Veremos más adelante que un atributo es normalmente definido con la cláusula private (con esto no permitimos el acceso al atributo desde otra clase).

A los atributos se tienen acceso desde cualquier función o método de la clase (salvo la main).

Luego de definir los atributos de la clase debemos declarar los métodos o funciones de la clase. La sintaxis es parecida a la min (sin la cláusula static):

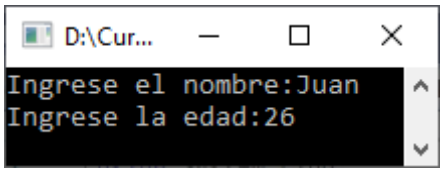
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto60
{
    0 referencias
    class Persona
    {
        private string nombre;
        private int edad;

        0 referencias
        public void Inicializar()
        {
            Console.Write("Ingrese el nombre:");
            nombre = Console.ReadLine();
            string linea;
            Console.Write("Ingrese la edad:");
            linea = Console.ReadLine();
            edad = int.Parse(linea);
        }

        0 referencias
        static void Main(string[] args)
        {
        }
    }
}
```

En el método inicializar (que será el primero que deberemos llamar desde la main) cargamos por teclado los atributos nombre y edad. Como podemos ver el método inicializar puede tener acceso a dos atributos de la clase Persona.

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto60
{
    2 referencias
    class Persona
    {
        private string nombre;
        private int edad;

        1 referencia
        public void Inicializar()
        {
```

```
            Console.Write("Ingrese el nombre:");
            nombre = Console.ReadLine();
            string linea;
            Console.Write("Ingrese la edad:");
            linea = Console.ReadLine();
            edad = int.Parse(linea);
        }

        0 referencias
        static void Main(string[] args)
        {
            Persona persona1 = new Persona();
            persona1.Inicializar();
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar para ver el resultado de llamar un método después de instanciar una clase:



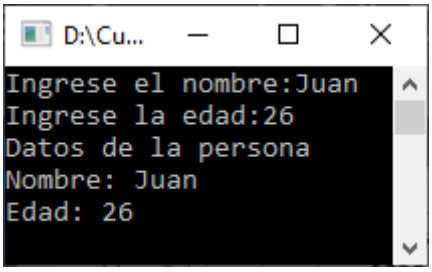
El segundo método tiene por objetivo imprimir el contenido de los atributos nombre y edad (los datos de los atributos se cargaron al ejecutarse previamente inicializar:

```
Console.Write("Nombre:");
Console.WriteLine(nombre);
Console.Write("Edad:");
Console.WriteLine(edad);
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto60
{
    2 referencias
    class Persona
    {
        private string nombre;
        private int edad;

        1 referencia
        public void Inicializar()
        {
            Console.Write("Ingrese el nombre:");
            nombre = Console.ReadLine();
            string linea;
```

```
            Console.Write("Ingrese la edad:");
            linea = Console.ReadLine();
            edad = int.Parse(linea);
        }

        1 referencia
        public void imprimir()
        {
            Console.WriteLine("Datos de la persona");
            Console.Write("Nombre: ");
            Console.WriteLine(nombre);
            Console.Write("Edad: ");
            Console.WriteLine(edad);
        }

        0 referencias
        static void Main(string[] args)
        {
            Persona persona1 = new Persona();
            persona1.Inicializar();
            persona1.imprimir();
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar de nuevo para ir viendo el resultado:

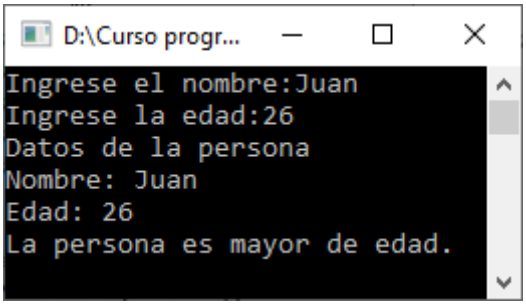


El tercer método tiene por objetivo mostrar un mensaje si la persona es mayor de edad:

```
public void EsMayorEdad()
{
    if (edad >= 18)
    {
        Console.Write("Es mayor de edad");
    }
    else
    {
        Console.Write("No es mayor de edad");
    }
    Console.ReadKey();
}
```

```
        1 referencia
        public void imprimir()
        {
            Console.WriteLine("Datos de la persona");
            Console.Write("Nombre: ");
            Console.WriteLine(nombre);
            Console.Write("Edad: ");
            Console.WriteLine(edad);
        }

        1 referencia
        public void EsMayorEdad()
        {
            if (edad>=18)
            {
                Console.WriteLine("La persona es mayor de edad.");
            }
            else
            {
                Console.WriteLine("La persona no es mayor de edad.");
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            Persona persona1 = new Persona();
            persona1.Inicializar();
            persona1.imprimir();
            persona1.EsMayorEdad();
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar para ver de nuevo el resultado:

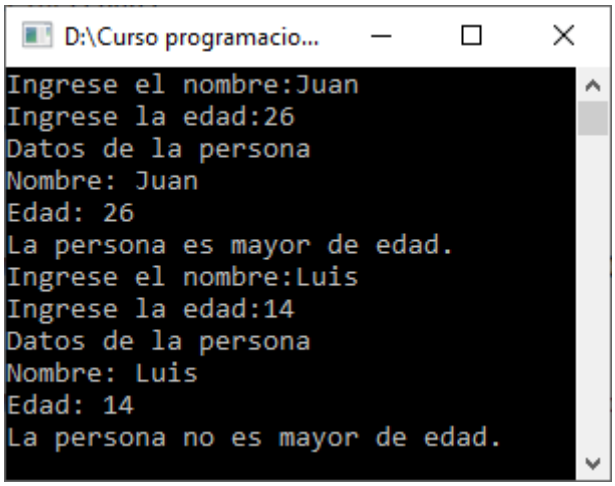


Por último en la main declaramos un segundo objeto de la clase Persona y llamamos a los respectivos métodos.

Ahora vamos a instanciar una segunda clase:

```
        0 referencias
        static void Main(string[] args)
        {
            Persona persona1 = new Persona();
            persona1.Inicializar();
            persona1.imprimir();
            persona1.EsMayorEdad();
            Persona persona2 = new Persona();
            persona2.Inicializar();
            persona2.imprimir();
            persona2.EsMayorEdad();
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar de nuevo:



Capítulo 60.- Declaración de una clase y definición de objetos – 2

Problema

Desarrollar un programa que cargue los lados de un triángulo e implante los siguientes métodos:

Inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no (equilátero es que sus tres lados miden lo mismo).

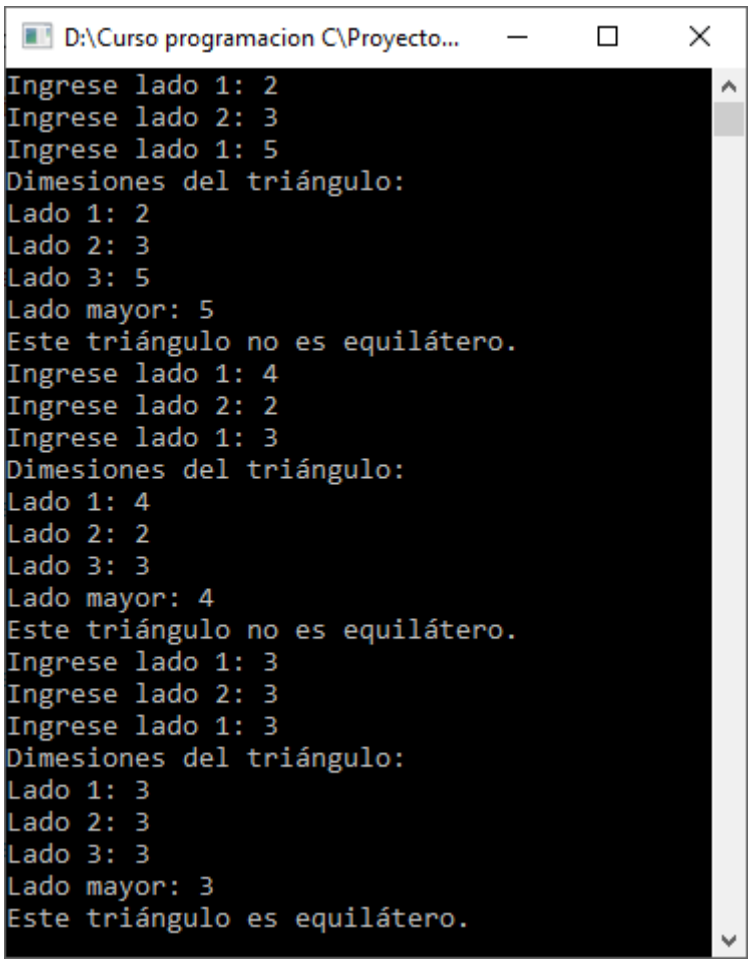
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto61
{
    6 referencias
    class Triangulo
    {
        private int lado1;
        private int lado2;
        private int lado3;

        3 referencias
        public void Inicializar()
        {
            string line;
            Console.Write("Ingrese lado 1: ");
            line = Console.ReadLine();
            lado1 = int.Parse(line);
            Console.Write("Ingrese lado 2: ");
            line = Console.ReadLine();
            lado2 = int.Parse(line);
            Console.Write("Ingrese lado 1: ");
            line = Console.ReadLine();
            lado3 = int.Parse(line);
        }

        3 referencias
        public void imprimir()
        {
            Console.WriteLine("Dimesiones del triángulo:");
            Console.Write("Lado 1: ");
            Console.WriteLine(lado1);
            Console.Write("Lado 2: ");
            Console.WriteLine(lado2);
            Console.Write("Lado 3: ");
            Console.WriteLine(lado3);
        }
```

Si ejecutamos este será el resultado:

```
3 referencias
public void LadoMayor()
{
    Console.Write("Lado mayor: ");
    if (lado1>lado2 & lado1>lado3)
    {
        Console.WriteLine(lado1);
    }
    else
    {
        {
            if (lado2>lado3)
            {
                Console.WriteLine(lado2);
            }
            else
            {
                Console.WriteLine(lado3);
            }
        }
    }
}

3 referencias
public void EsEquilatero()
{
    if (lado1==lado2 && lado1==lado3)
    {
        Console.WriteLine("Este triángulo es equilátero.");
    }
    else
    {
        {
            Console.WriteLine("Este triángulo no es equilátero.");
        }
    }
}
0 referencias
static void Main(string[] args)
{
    Triangulo triangulo1 = new Triangulo();
    triangulo1.Inicializar();
    triangulo1.imprimir();
    triangulo1.LadoMayor();
    triangulo1.EsEquilatero();
    Triangulo triangulo2 = new Triangulo();
    triangulo2.Inicializar();
    triangulo2.imprimir();
    triangulo2.LadoMayor();
    triangulo2.EsEquilatero();
    Triangulo triangulo3 = new Triangulo();
    triangulo3.Inicializar();
    triangulo3.imprimir();
    triangulo3.LadoMayor();
    triangulo3.EsEquilatero();
    Console.ReadKey();
```

```
        }
    }
}
```

Si ejecutamos este será el resultado:

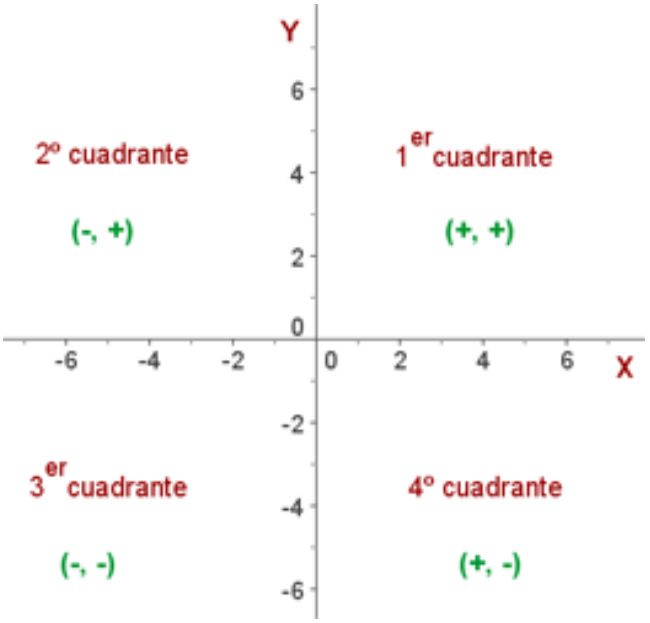


Capítulo 61.- Declaración de una clase y definición de objetos – 3

Problema

Desarrollar una clase que represente un punto en el plano y tenga los siguientes métodos:

- Cargar los valores x e y.
- Imprimir en que cuadrante se encuentra dicho punto (concepto matemático, primer cuadrante se x e y son positivos, si x<0 e y>0 segundo cuadrante, etc.).

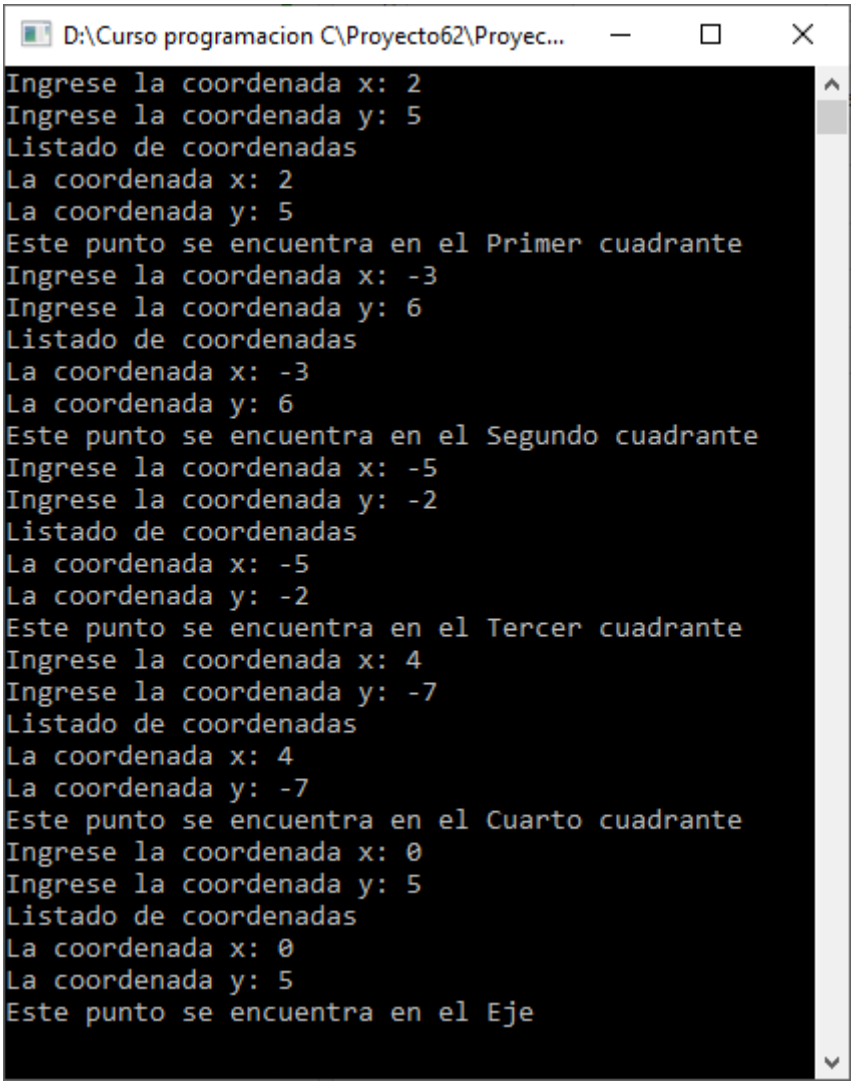



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto62
{
    10 referencias
    class Coordenada
    {
        int x;
        int y;

        5 referencias
        public void inicializar()
        {
            string line;
            Console.Write("Ingrese la coordenada x: ");
            line = Console.ReadLine();
            x = int.Parse(line);
            Console.Write("Ingrese la coordenada y: ");
            line = Console.ReadLine();
            y = int.Parse(line);
        }
```

```
5 referencias
public void imprimir()
{
    Console.WriteLine("Listado de coordenadas");
    Console.Write("La coordenada x: ");
    Console.WriteLine(x);
    Console.Write("La coordenada y: ");
    Console.WriteLine(y);
}

5 referencias
public void cuadrante()
{
    Console.Write("Este punto se encuentra en el ");
    if(x>0 && y>0)
    {
        Console.WriteLine("Primer cuadrante");
    }
    else
    {
        if(x<0 && y >0)
        {
            Console.WriteLine("Segundo cuadrante");
        }
        else
        {
            if(x<0 && y<0)
            {
                Console.WriteLine("Tercer cuadrante");
            }
            else
            {
                if(x>0 && y<0)
                {
                    Console.WriteLine("Cuarto cuadrante");
                }
                else
                {
                    Console.WriteLine("Eje");
                }
            }
        }
    }
}

0 referencias
static void Main(string[] args)
{
    Coordenada coor1 = new Coordenada();
    coor1.inicializar();
    coor1.imprimir();
    coor1.cuadrante();
    Coordenada coor2 = new Coordenada();
    coor2.inicializar();
    coor2.imprimir();
```

```
            coor2.cuadrante();
            Coordenada coor3 = new Coordenada();
            coor3.inicializar();
            coor3.imprimir();
            coor3.cuadrante();
            Coordenada coor4 = new Coordenada();
            coor4.inicializar();
            coor4.imprimir();
            coor4.cuadrante();
            Coordenada coor5 = new Coordenada();
            coor5.inicializar();
            coor5.imprimir();
            coor5.cuadrante();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 62.- Declaración de una clase y definición de objetos – 4

Problema

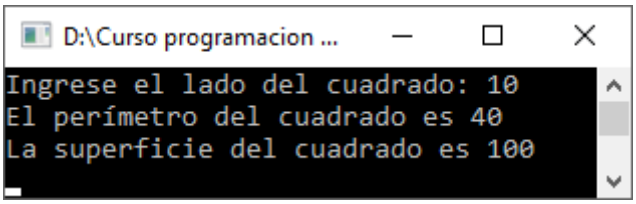
Desarrollar una clase que represente un Cuadrado y tenga los siguientes métodos:

- Cargar el valor de su lado.
- Imprimir su perímetro y su superficie.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto63
{
    2 referencias
    class Cuadrado
    {
        int lado;

        1 referencia
        public void inicialice()
        {
            string line;
            Console.Write("Ingrese el lado del cuadrado: ");
            line = Console.ReadLine();
            lado = int.Parse(line);
        }

        1 referencia
        public void perimetro()
        {
            int perimetro;
            perimetro = lado * 4;
            Console.Write("El perímetro del cuadrado es ");
            Console.WriteLine(perimetro);
        }

        1 referencia
        public void superficie()
        {
            int superficie;
            superficie = lado * lado;
            Console.Write("La superficie del cuadrado es ");
            Console.WriteLine(superficie);
        }


        0 referencias
        static void Main(string[] args)
        {
            Cuadrado cuadrado1 = new Cuadrado();
            cuadrado1.inicialice();
            cuadrado1.perimetro();
```

```
            cuadrado1.superficie();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 63.- Declaración de una clase y definición de objetos – 5

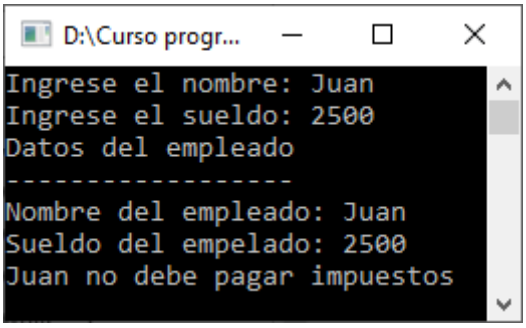
Problema propuesto

Confeccionar una clase que represente un empleado. Definir como atributos su nombre y sueldo. Confeccionar los métodos para la carga, otro para imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000).

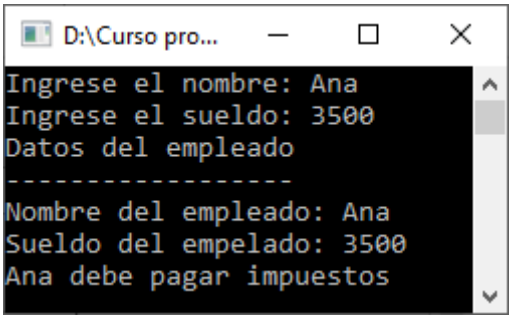
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto64
{
    2 referencias
    class Empleado
    {
        string nombre;
        int sueldo;

        1 referencia
        public void cargar()
        {
            string line;
            Console.Write("Ingrese el nombre: ");
            nombre = Console.ReadLine();
            Console.Write("Ingrese el sueldo: ");
            line = Console.ReadLine();
            sueldo = int.Parse(line);
        }

        1 referencia
        public void imprimir()
        {
            Console.WriteLine("Datos del empleado");
            Console.WriteLine("-------------------");
            Console.Write("Nombre del empleado: ");
            Console.WriteLine(nombre);
            Console.Write("Sueldo del empelado: ");
            Console.WriteLine(sueldo);
        }
        1 referencia
        public void pagarImpuestos()
        {
            if (sueldo>3000)
            {
                Console.Write(nombre);
                Console.WriteLine(" debe pagar impuestos");
            }
            else
            {
                Console.Write(nombre);
                Console.WriteLine(" no debe pagar impuestos");
```

```
        }
    }

    0 referencias
    static void Main(string[] args)
    {
        Empleado empleado1 = new Empleado();
        empleado1.cargar();
        empleado1.imprimir();
        empleado1.pagarImpuestos();
        Console.ReadKey();
    }
}
}
```

Vamos a ejecutar introduciendo un sueldo menor de 3000.



Ejecutamos de nuevo introduciendo un sueldo superior de 3000.



Capítulo 64.- Declaración de una clase y definición de objetos – 6

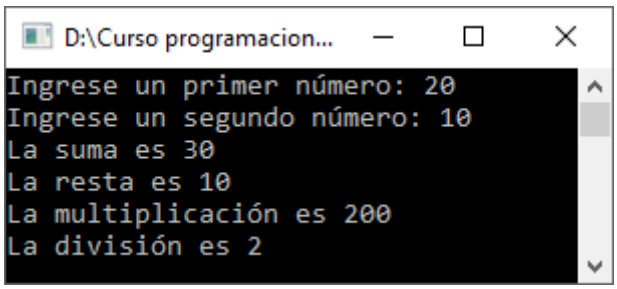
Problema propuesto

Implementar la clase operaciones. Se debe cargar dos valores enteros, calcular suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto65
{
    2 referencias
    class Operaciones
    {
        int num1;
        int num2;

        1 referencia
        public void inicializar()
        {
            string line;
            Console.Write("Ingrese un primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese un segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
        }

        1 referencia
        public void suma()
        {
            int suma = num1 + num2;
            Console.WriteLine("La suma es " + suma);
        }

        1 referencia
        public void resta()
        {
            int resta = num1 - num2;
            Console.WriteLine("La resta es " + resta);
        }

        1 referencia
        public void multiplicacion()
        {
            int multiplicacion = num1 * num2;
            Console.WriteLine("La multiplicación es " + multiplicacion);
        }
        1 referencia
        public void division()
        {
            int division = num1 / num2;
            Console.WriteLine("La división es " + division);
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Operaciones operacion1 = new Operaciones();
            operacion1.inicializar();
            operacion1.suma();
            operacion1.resta();
            operacion1.multiplicacion();
            operacion1.division();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 65.- Declaración de métodos – 1

Cuando uno plantea una clase en lugar de especificar todo el algoritmo en un único método (lo que hicimos en los primeros pasos de este tutorial) es dividir todas las responsabilidades de las clase en un conjunto de métodos.

Un método hemos visto que tiene la siguiente sintaxis:

```
public void [nombre del método]()
{
    [algoritmo]
}
```

Veremos que hay varios tipos de métodos:

Métodos con parámetros

Un método puede tener parámetros:

```
public void [nombre del método]([parámetros])
{
    [algoritmo]
}
```

Los parámetros los podemos imaginar como variables locales al método, pero su valor se inicializa con datos que llegan cuando lo llamamos.

Problema:

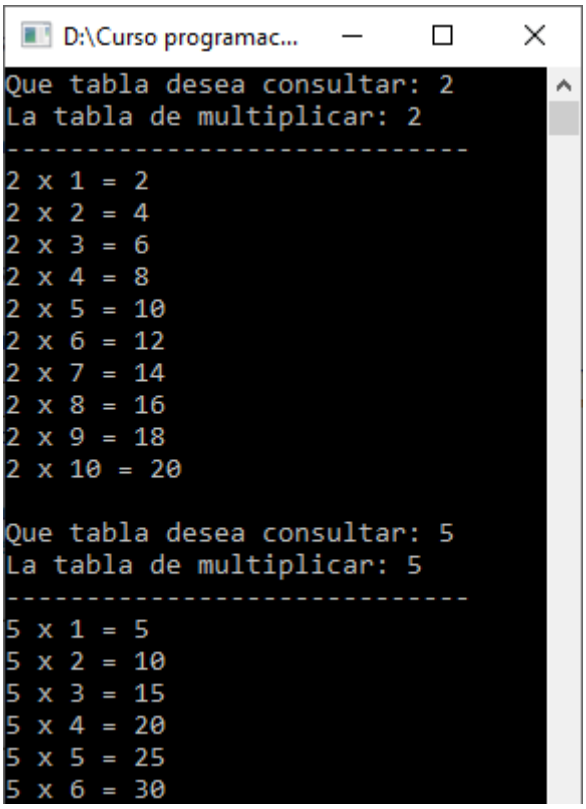
Confeccionar una clase que permita ingresar enteros por teclado y nos muestre la tabla de multiplicar de dicho valor.

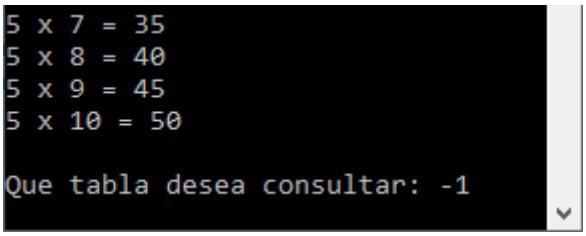
Finalizar el programa al ingresar el -1.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto66
{
    2 referencias
    class Tabla
    {
        1 referencia
        public void cargarValor()
        {
            int tabla;
            string line;
            do
            {
                Console.Write("Que tabla desea consultar: ");
                line = Console.ReadLine();
                tabla = int.Parse(line);
```

```
            if (tabla!=-1)
            {
                mostrarTabla(tabla);
            }

        } while (tabla != -1);
    }

    1 referencia
    public void mostrarTabla(int num)
    {
        int f, resultado;
        Console.WriteLine("La tabla de multiplicar: " + num);
        Console.WriteLine("----------------------------");
        for (f=1; f<=10;  f++)
        {
            resultado = f * num;
            Console.WriteLine(num + " x " + f + " = " +  resultado);
        }
        Console.WriteLine();
    }

    0 referencias
    static void Main(string[] args)
    {
        Tabla tb = new Tabla();
        tb.cargarValor();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:




5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
Que tabla desea consultar: -1

Capítulo 66.- Declaración de métodos – 2

Métodos que retornan un dato

Un método puede retornar un dato:

```
public [tipo de dato] [nombre del método]([parámetros])
{
  [algoritmo]
  return [tipo de dato]
}
```

Cuando un método retorna un dato en vez de indicar la palabra clave void previo al nombre del método indicamos el tipo de dato que retorna. Luego dentro del algoritmo en el momento que queremos que finalice el mismo y retorne el dato empleamos la palabra clave return con el valor respectivo.

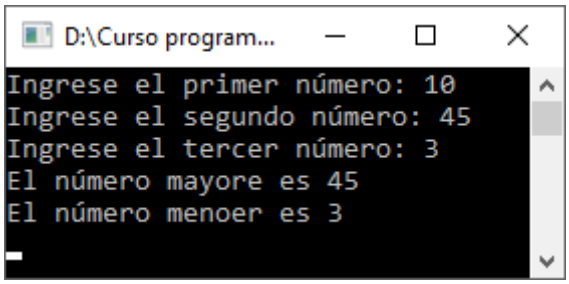
Problema

Confeccionar una clase que permita ingresar tres valores por teclado. Luego mostrar el mayor y el menor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto67
{
    2 referencias
    class Numero
    {
        int num1;
        int num2;
        int num3;

        1 referencia
        public void inicializar()
        {
            string line;
            Console.Write("Ingrese el primer número: ");
            line = Console.ReadLine();
            num1 = int.Parse(line);
            Console.Write("Ingrese el segundo número: ");
            line = Console.ReadLine();
            num2 = int.Parse(line);
            Console.Write("Ingrese el tercer número: ");
            line = Console.ReadLine();
            num3 = int.Parse(line);
        }
```

```
        1 referencia
        public int mayor()
        {
            if (num1 > num2 && num1 > num3)
            {
                return num1;
            }
            else
            {
                if (num2>num3)
                {
                    return num2;
                }
                else
                {
                    return num3;
                }
            }
        }

        1 referencia
        public int menor()
        {
            if (num1 < num2 && num1 < num3)
            {
                return num1;
            }
            else
            {
                if (num2 < num3)
                {
                    return num2;
                }
                else
                {
                    return num3;
                }
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            Numero num1 = new Numero();
            num1.inicializar();
            Console.WriteLine("El número mayore es " + num1.mayor());
            Console.WriteLine("El número menoer es " + num1.menor());
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 67.- Estructura de datos tipo vector – 1

Hemos empleado variables de distinto tipo para el almacenamiento de datos (variables int, float, string). En este capítulo veremos otro tipo de variables que permiten almacenar un conjunto de datos en una única variable.

Un vector es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo.

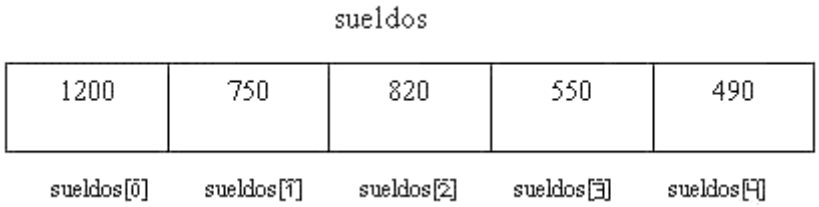
Con un único nombre se define un vector y por medio de un subíndice hacemos referencia a cada elemento del mismo (componente).

Problema

Se desea guardar los sueldos de 5 empleados.

Según lo conocido deberíamos definir 5 variables si queremos tener en un cierto momento 5 sueldos almacenados en la memoria.

Empleando un vector solo se requiere definir un único nombre y accedemos a cada elemento por medio del subíndice.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto68
{
    2 referencias
    class PruebaVector1
    {
        private int[] sueldos;

        1 referencia
        public void Cargar()
        {
            sueldos = new int[5];
            for (int f = 0; f < 5; f++)
            {
                Console.Write("Ingrese el valor del elemento: ");
                string linea;
                linea = Console.ReadLine();
                sueldos[f] = int.Parse(linea);
            }
        }
```

```
        1 referencia
        public void Imprimir()
        {
            for(int f=0; f<5; f++)
            {
                Console.WriteLine(sueldos[f]);
            }
            Console.ReadKey();
        }


        0 referencias
        static void Main(string[] args)
        {
            PruebaVector1 pv = new PruebaVector1();
            pv.Cargar();
            pv.Imprimir();
        }
    }
}
```

Para la declaración de un vector le antecedemos el nombre los corchetes abiertos y cerrados:

```
private int[] sueldos;
```

Lo definimos como atributo de la clase ya que lo utilizaremos en los dos métodos.

En el método de Cargar lo primero que hacemos es crear el vector (en C# los vectores son objetos por lo que es necesario proceder a su creación mediante el operador new):

```
sueldos = new int[5];
```

Cuando creamos el vector indicamos entre corchetes la cantidad de elementos que se pueden almacenar posteriormente en el mismo.

Para cargar cada elemento debemos indicar entre corchetes que elemento del vector estamos accediendo:

```
for (int f = 0; f < 5; f++)
{
    Console.Write("Ingrese valor de la componente:");
    String linea;
    linea = Console.ReadLine();
    sueldos[f] = int.Parse(linea);
}
```

La estructura de programación que más se adapta para cargar en forma completa los elementos de un vector es un for, ya que sabemos de antemano la cantidad de valores a cargar.

Cuando f vale cero estamos accediendo al primer elemento del vector (en nuestro caso sería):

```
sueldos[f] = int.Parse(linea);
```

Lo más común es utilizar una estructura repetitiva for para recorrer cada elemento del vector.

Utilizar el for nos reduce la cantidad de código, si no utilizo un for debería en forma secuencial implementar el siguiente código:

```
string linea;
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[0]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[1]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[2]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[3]=int.Parse(linea);
Console.Write("Ingrese valor de la componente:");
linea=Console.ReadLine();
sueldos[4]=int.Parse(linea);
```

La impresión de los elementos del vector lo hacemos en otro método:

```
public void Imprimir()
{
    for(int f = 0; f < 5; f++)
    {
        Console.WriteLine(sueldos[f]);
    }
    Console.ReadKey();
}
```

Siempre que queremos acceder a un elemento del vector debemos indicar entre corchetes el número de elemento, dicho valor comienza a numerarse en cero y continua hasta un número menor del tamaño del valor en nuestro caso creamos el vector con 5 elementos.

Capítulo 68.- Estructura de datos tipo vector – 2

Problema

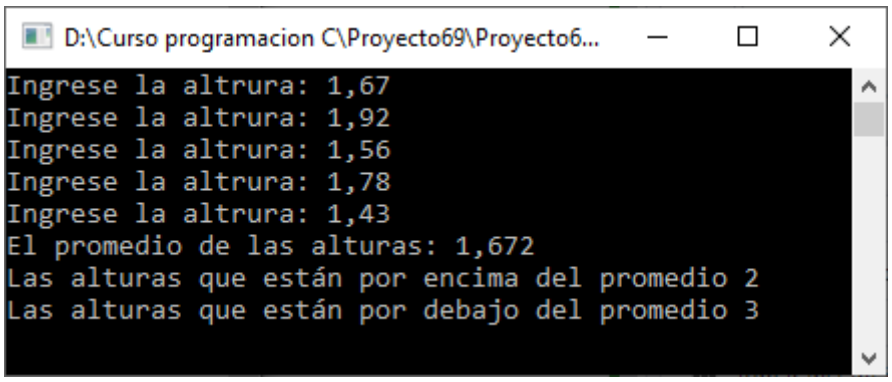
Definir un vector de 5 elementos de tipo float que representan las alturas de 5 personas.

Obtener el promedio de las mismas. Contar cuántas son más altas que el promedio y cuántas más bajas.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto69
{
    2 referencias
    class Alturas
    {
        private float [] altura;

        1 referencia
        public void inicializar()
        {
            altura = new float[5];
            string line;
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese la altrura: ");
                line = Console.ReadLine();
                altura[f]=float.Parse(line);
            }
        }

        2 referencias
        public float promedio()
        {
            float suma, pro;
            suma = 0;
            for (int f=0;f < 5; f++)
            {
                suma = suma + altura[f];
            }
            pro = suma / 5;
            return pro;
        }
```

```
        1 referencia
        public void contar()
        {
            int masAltas, menosAltas;
            masAltas = 0;
            menosAltas = 0;
            float prome = promedio();
            for (int f=0; f<5; f++)
            {
                if (altura[f]>prome)
                {
                    masAltas=masAltas+1;
                }
                else
                {
                    if (altura[f]<prome)
                    {
                        menosAltas=menosAltas+1;
                    }
                }
            }
            Console.WriteLine("Las alturas que están por encima del promedio " + masAltas);
            Console.WriteLine("Las alturas que están por debajo del promedio " +  menosAltas);
            Console.ReadKey();
        }


        0 referencias
        static void Main(string[] args)
        {
            Alturas altura1 = new Alturas();
            altura1.inicializar();
            Console.WriteLine("El promedio de las alturas: "+ altura1.promedio());
            altura1.contar();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 69.- Estructura de datos tipo vector – 3

Problema

Una empresa tiene dos turnos (mañana y tarde) en los que trabajan 8 empleados (4 por la mañana y 4 por la tarde).

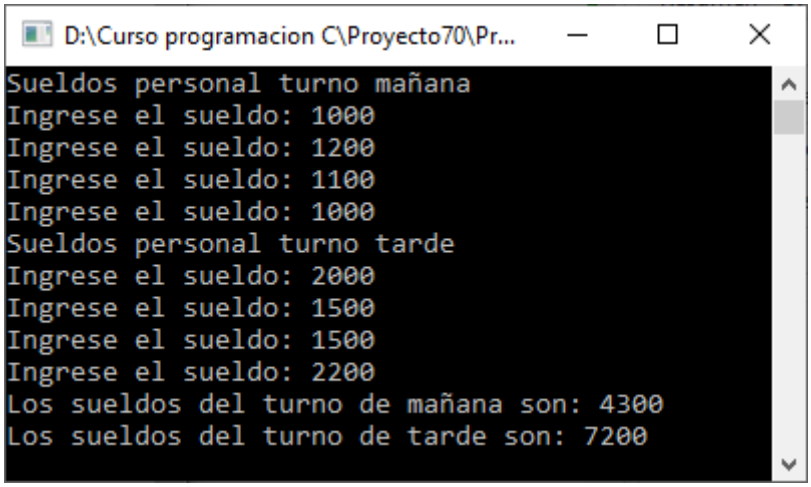
Confeccionar un programa que permita almacenar los sueldos de los empleados agrupados por turno.

Imprimir los gastos en sueldos de cada turno.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto70
{
    2 referencias
    class Turnos
    {
        private int[] turnoMa;
        private int[] turnoTa;

        1 referencia
        public void inicializar()
        {
            turnoMa = new int[4];
            turnoTa = new int[4];
            string line;
            Console.WriteLine("Sueldos personal turno mañana");
            for (int f = 0; f < 4; f++)
            {
                Console.Write("Ingrese el sueldo: ");
                line = Console.ReadLine();
                turnoMa[f] = int.Parse(line);
            }
            Console.WriteLine("Sueldos personal turno tarde");
            for (int f = 0; f < 4; f++)
            {
                Console.Write("Ingrese el sueldo: ");
                line = Console.ReadLine();
                turnoTa[f] = int.Parse(line);
            }
        }

        1 referencia
        public int sumaMa()
        {
            int suma = 0;
            for (int f=0; f<4; f++)
            {
                suma = suma + turnoMa[f];
            }
```

```
            return suma;
        }

        1 referencia
        public int sumaTa()
        {
            int suma = 0;
            for (int f = 0; f < 4; f++)
            {
                suma = suma + turnoTa[f];
            }
            return suma;
        }

        0 referencias
        static void Main(string[] args)
        {
            Turnos turno1 = new Turnos();
            turno1.inicializar();
            Console.WriteLine("Los sueldos del turno de mañana son: " + turno1.sumaMa());
            Console.WriteLine("Los sueldos del turno de tarde son: " + turno1.sumaTa());
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 70.- Estructura de datos tipo vector – 4

Problema propuesto

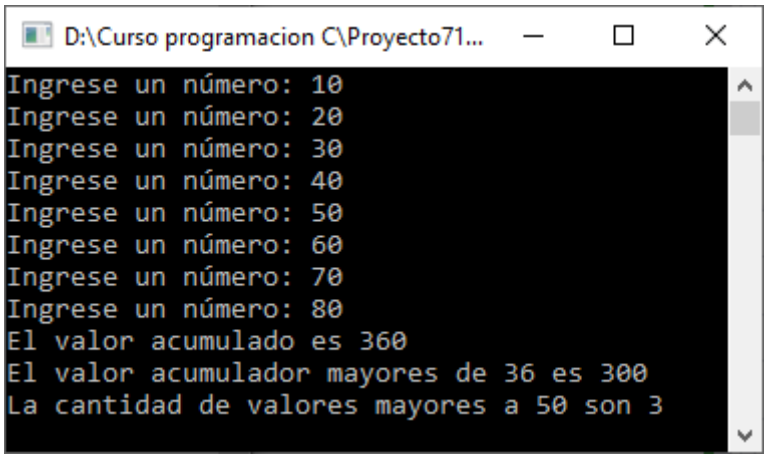
Desarrollar un programa que permita ingresar un vector de 8 elementos, e informe:

- El valor acumulado de todos los elementos del vector.
- El valor acumulado de los elementos del vector que sean mayores a 36.
- Cantidad de valores mayores a 50.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto71
{
    2 referencias
    class Vector
    {
        private int[] numero;

        1 referencia
        public void cargar()
        {
            numero = new int[8];
            string line;
            for (int f = 0; f < 8; f++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                numero[f] = int.Parse(line);
            }
        }

        1 referencia
        public void ValorAcumulado()
        {
            int suma = 0;
            for (int f = 0;f < 8;f++)
            {
                suma = suma + numero[f];
            }
            Console.WriteLine("El valor acumulado es " + suma);
        }
```

```
        1 referencia
        public void ValorAcumuladoMayor36()
        {
            int suma = 0;
            for (int f = 0; f<8; f++)
            {
                if (numero[f] > 36)
                {
                    suma = suma + numero[f];
                }
            }
            Console.WriteLine("El valor acumulador mayores de 36 es " + suma);
        }

        1 referencia
        public void cantidad()
        {
            int cantidad = 0;
            for (int f = 0; f < 8; f++)
            {
                if (numero[f] > 50)
                {
                    cantidad = cantidad + 1;
                }
            }
            Console.WriteLine("La cantidad de valores mayores a 50 son " + cantidad);
        }

        0 referencias
        static void Main(string[] args)
        {
            Vector vec1 = new Vector();
            vec1.cargar();
            vec1.ValorAcumulado();
            vec1.ValorAcumuladoMayor36();
            vec1.cantidad();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 71.- Estructura de datos tipo vector – 5

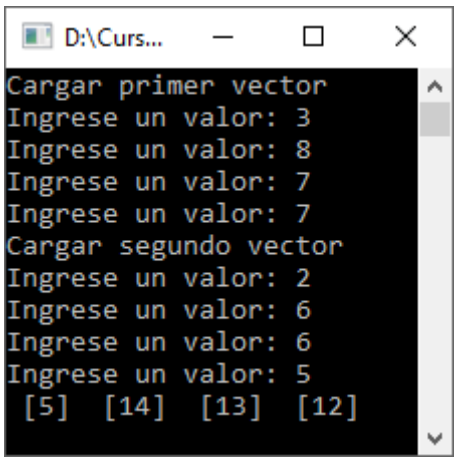
Problema propuesto

Realizar un programa que pida la carga de dos vectores numéricos de enteros de 4 elementos. Obtener la suma de los dos vectores, dicho resultado guardarlo en un tercer vector del mismo tamaño. Suma componente a componente.

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto72
{
    2 referencias
    class Vectores
    {
        private int[] vec1;
        private int[] vec2;
        private int[] vec3;

        1 referencia
        public void cargar()
        {
            vec1 = new int[4];
            vec2 = new int[4];
            string line;
            Console.WriteLine("Cargar primer vector");
            for (int f = 0; f < 4; f++)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                vec1[f] = int.Parse(line);
            }
            Console.WriteLine("Cargar segundo vector");
            for (int f = 0; f < 4; f++)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                vec2[f] = int.Parse(line);
            }
        }

        1 referencia
        public void sumaVectores()
        {
            vec3 = new int[4];
            for (int f = 0;f < 4; f++)
            {
                vec3[f] = vec1[f] + vec2[f];
            }
        }
```

```
        1 referencia
        public void imprimirVector()
        {
            for (int f=0; f<4; f++)
            {
                Console.Write(" [" + vec3[f] + "] ");
            }
            Console.WriteLine();
        }

        0 referencias
        static void Main(string[] args)
        {
            Vectores ve1 = new Vectores();
            ve1.cargar();
            ve1.sumaVectores();
            ve1.imprimirVector();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 72.- Estructura de datos tipo vector – 6

Problema propuesto

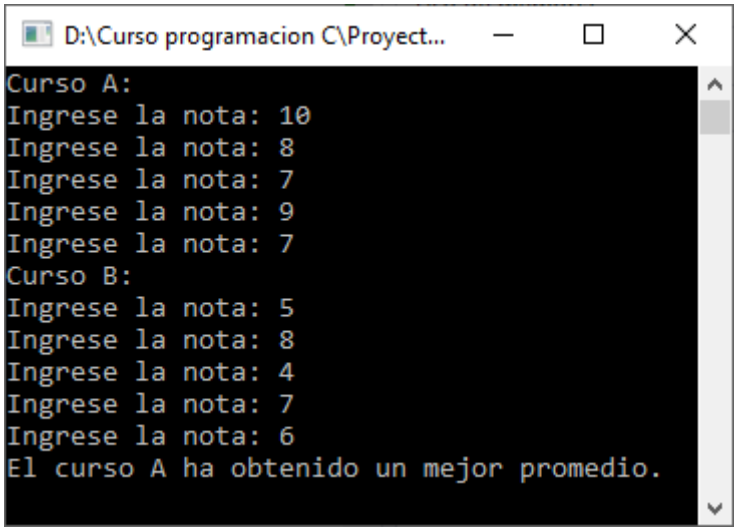
Se tiene las notas del primer parcial de los alumnos de 2 cursos, el curso A y el curso B, cada curso cuenta con 5 alumnos.

Realizar un programa que muestre el curso que obtuvo el mayor promedio general.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto73
{
    2 referencias
    class Curso
    {
        private int[] cursoA;
        private int[] cursoB;

        1 referencia
        void cargar()
        {
            cursoA = new int[5];
            cursoB = new int[5];
            string line;
            Console.WriteLine("Curso A:");
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese la nota: ");
                line = Console.ReadLine();
                cursoA[f] = int.Parse(line);
            }
            Console.WriteLine("Curso B:");
            for (int f = 0; f < 5; f++)
            {
                Console.Write("Ingrese la nota: ");
                line = Console.ReadLine();
                cursoB[f] = int.Parse(line);
            }
        }

        1 referencia
        void mayorNotaPromedio()
        {
            int sumaA=0, sumaB=0;
            int promedioA, promedioB;
            for (int f=0;f < 5; f++)
            {
                sumaA = sumaA + cursoA[f];
                sumaB = sumaB + cursoB[f];
            }
            promedioA = sumaA / 5;
            promedioB = sumaB / 5;
```

```
            if (promedioA > promedioB)
            {
                Console.WriteLine("El curso A ha obtenido un mejor promedio.");
            }
            else
            {
                Console.WriteLine("El curso B ha obtenido un mejor promedio.");
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            Curso curso = new Curso();
            curso.cargar();
            curso.mayorNotaPromedio();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Partiendo de este proyecto te propongo que realices las siguientes modificaciones.
Trata el curso A y curso B como dos instancias.
La función notaPromedio que nos retorne su valor y realizar la comparación en el Main.
Una vez ejecutes el programa debemos obtener los mismos resultados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto73
{
    4 referencias
    class Curso
    {
        private int[] curso;
```

```
        2 referencias
        public void cargar()
        {
            curso = new int[5];
            string line;
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese la nota: ");
                line = Console.ReadLine();
                curso[f] = int.Parse(line);
            }

        }

        2 referencias
        public int NotaPromedio()
        {
            int suma=0;
            int promedio;
            for (int f=0;f < 5; f++)
            {
                suma = suma + curso[f];
            }
            promedio = suma / 5;
            return promedio;
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Curso cursoA = new Curso();
            Console.WriteLine("Curso A: ");
            cursoA.cargar();
            Curso cursoB = new Curso();
            Console.WriteLine("Curso B: ");
            cursoB.cargar();
            if (cursoA.NotaPromedio() > cursoB.NotaPromedio())
            {
                Console.WriteLine("El curso A ha obtenido un mejor promedio.");
            }
            else
            {
                Console.WriteLine("El curso B ha obtenido un mejor promedio.");
            }
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos obtendremos el mismo resultado.

Capítulo 73.- Estructura de datos tipo vector – 7

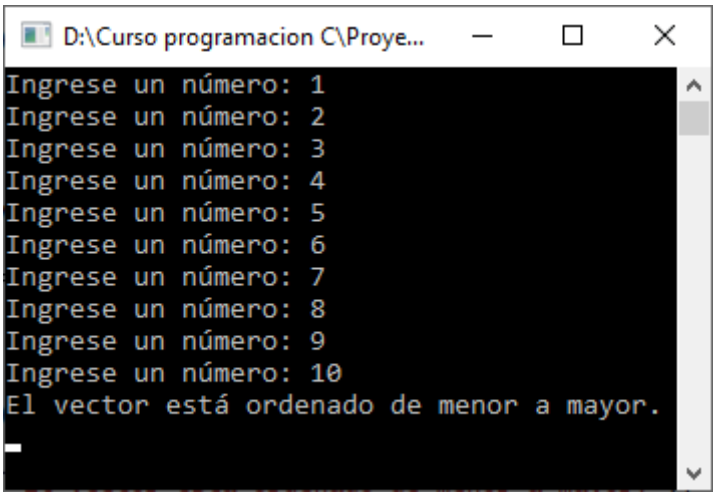
Problema propuesto

Cargar un vector de 10 elementos y verificar posteriormente si el mismo está ordenado de menor a mayor.

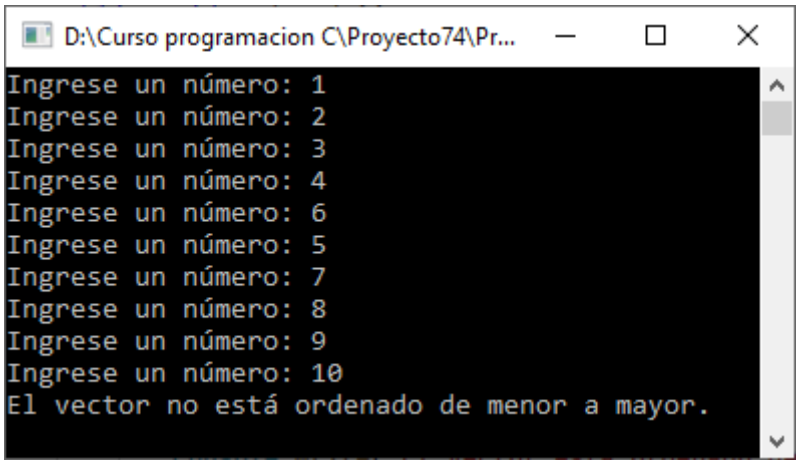
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto74
{
    2 referencias
    class Vector
    {
        private int[] vec;

        1 referencia
        public void cargar()
        {
            vec = new int[10];
            string line;
            for (int f=0; f < 10; f++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                vec[f]=int.Parse(line);
            }
        }

        1 referencia
        public void ordenado()
        {
            int ordenado = 1;
            for (int f=0; f<9; f++)
            {
                if (vec[f] > vec[f+1])
                {
                    ordenado = 0;
                    break;
                }
            }
            if (ordenado == 1)
            {
                Console.Write("El vector está ordenado de menor a mayor.");
            }
            else
            {
                Console.Write("El vector no está ordenado de menor a mayor.");
            }
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Vector v = new Vector();
            v.cargar();
            v.ordenado();
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar introduciendo valores ordenados:



Vamos a ejecutar de nuevo y en este caso no seguiremos un orden en toda la numeración:



Capítulo 74.- Vector (Tamaño de un vector) – 1

Como hemos visto cuando se crea un vector indicamos entre corchetes su tamaño:

```
sueldos=new int[5];
```

Luego cuando tenemos que correr dicho vector disponemos una estructura repetitiva for:

```
for(int f=0;f<5;f++)
{
    Console.Write("Ingrese valor de la componente:");
    string linea;
    linea=Console.ReadLine();
    sueldos[f]=int.Parse(linea);
}
```

Como vemos el for se repite mientras el contador f vale menos de 5. Esta estructura repetitiva es idéntica cada vez que recorremos el vector.

Que pasa ahora si cambiamos el tamaño del vector cuando lo creamos:

```
sueldos=new int[7];
```

Con este tenemos que cambiar todos los for que corren dicho vector. Ahora veremos que un vector al ser un objeto tiene una propiedad llamada Length que almacena su tamaño. Luego podemos modificar todos los for con la siguiente sintaxis:

```
for(int f=0;f<sueldos.Length;f++)
{
    Console.Write("Ingrese valor de la componente:");
    string linea;
    linea=Console.ReadLine();
    sueldos[f]=int.Parse(linea);
}
```

También podemos pedir al usuario que indique el tamaño del vector en tiempo de ejecución, en estos casos se hace imprescindible el empleo de la propiedad Length.

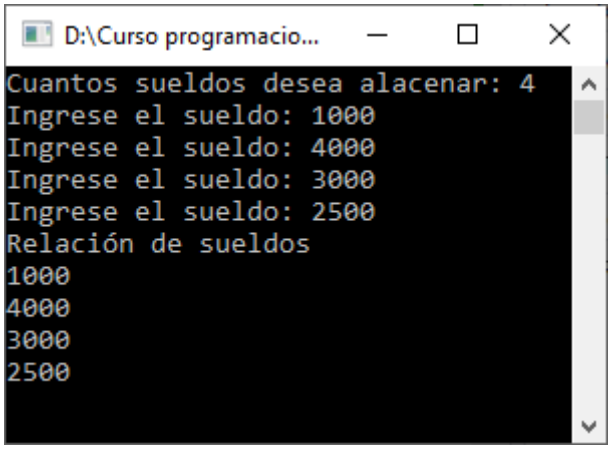
Problema

Se desea almacenar los sueldos de los operarios. Cuando se ejecuta el programa se debe pedir la cantidad de sueldos a ingresar. Luego crear un vector con dicho tamaño.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto75
{
    2 referencias
    class Vector
    {
        private int[] sueldos;
```

```
        1 referencia
        public void cargar()
        {
            int n;
            string line;
            Console.Write("Cuantos sueldos desea alacenar: ");
            line = Console.ReadLine();
            n=int.Parse(line);
            sueldos = new int[n];
            for (int f=0; f<n; f++)
            {
                Console.Write("Ingrese el sueldo: ");
                line= Console.ReadLine();
                sueldos[f] = int.Parse(line);
            }
        }
        1 referencia
        public void imprimir()
        {
            Console.WriteLine("Relación de sueldos");
            for (int f=0;f<sueldos.Length;f++)
            {
                Console.WriteLine(sueldos[f]);
            }
            Console.WriteLine();
        }
        0 referencias
        static void Main(string[] args)
        {
            Vector v = new Vector();
            v.cargar();
            v.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



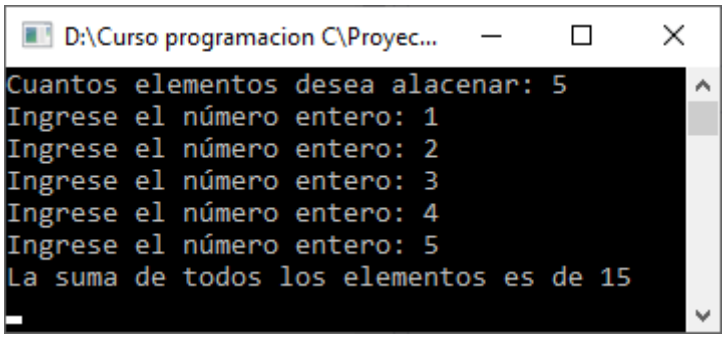
Capítulo 75.- Vector (Tamaño de un vector) – 2

Problema propuesto

Desarrollar un programa que permita ingresar un vector de n elementos, ingresar n por teclado. Luego imprimir la suma de todos sus elementos en otro método.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto76
{
    2 referencias
    class Vector
    {
        private int[] elementos;

        1 referencia
        public void cargar()
        {
            int n;
            string line;
            Console.Write("Cuantos elementos desea alacenar: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            elementos = new int[n];
            for (int f = 0; f < n; f++)
            {
                Console.Write("Ingrese el número entero: ");
                line = Console.ReadLine();
                elementos[f] = int.Parse(line);
            }
        }
        1 referencia
        public void sumar()
        {
            int suma = 0;
            for (int f = 0; f < elementos.Length; f++)
            {
                suma = suma + (elementos[f]);
            }
            Console.WriteLine("La suma de todos los elementos es de " + suma);
        }
        0 referencias
        static void Main(string[] args)
        {
            Vector v = new Vector();
            v.cargar();
            v.sumar();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Curso programacion C\Proyec...
Cuantos elementos desea alacenar: 5
Ingrese el número entero: 1
Ingrese el número entero: 2
Ingrese el número entero: 3
Ingrese el número entero: 4
Ingrese el número entero: 5
La suma de todos los elementos es de 15

Capítulo 76.- Vectores paralelos

Este concepto se da cuando hay una relación entre los elementos de igual subíndice (misma posición) de un vector y el otro.

nombres	Juan	Ana	Marcos	Pablo	Laura
edades	12	21	27	14	21

Si tenemos dos vectores de 5 elementos cada uno. En uno se almacenan los nombres de personas en el otro las edades de dichas personas.

Decimos que el vector nombres es paralelo al vector edades si en el componente 0 de cada vector se almacena información relacionada a una persona (Juan – 12 años).

Es decir hay una relación entre cada componente de los dos vectores.

Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesen los datos almacenados en las estructuras de datos.

Problema

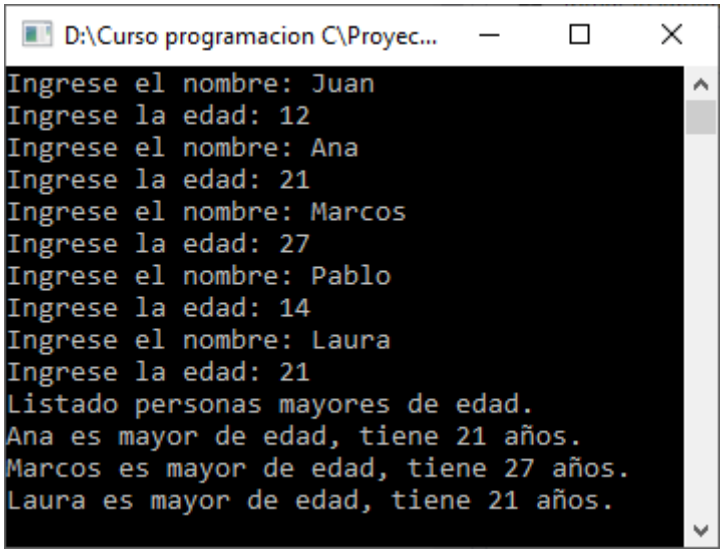
Desarrollar un programa que permita cargar 5 nombres de 5 personas y sus edades respectivas. Luego realizar la carga por teclado de todos los datos.

Imprimir los nombres de las personas mayores de edad (mayores o iguales a 18 años).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto77
{
    2 referencias
    class Personas
    {
        string[] nombre;
        int[] edad;

        1 referencia
        public void cargar()
        {
            nombre = new string[5];
            edad = new int[5];
            string line;
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese el nombre: ");
                nombre[f] = Console.ReadLine();
                Console.Write("Ingrese la edad: ");
                line = Console.ReadLine();
                edad[f]=int.Parse(line);
```

```
        }
    }

    1 referencia
    public void imprimirMayorEdad()
    {
        Console.WriteLine("Listado personas mayores de edad.");
        for (int f=0; f<5; f++)
        {
            if (edad[f]>=18)
            {
                Console.WriteLine(nombre[f] + " es mayor de edad, tiene "
                    + edad[f] + " años.");
            }
        }
    }

    0 referencias
    static void Main(string[] args)
    {
        Personas per1 = new Personas();
        per1.cargar();
        per1.imprimirMayorEdad();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 77.- Vectores (mayor y menor elemento) – 1

es una actividad común la búsqueda del mayor y menor elemento de un vector, lo mismo que su posición.

sueldos

120	750	820	550	490
sueldos[0]	sueldos[1]	sueldos[2]	sueldos[3]	sueldos[4]

El mayor elemento es el 820 y se encuentra en la posición nº 2.

Problema

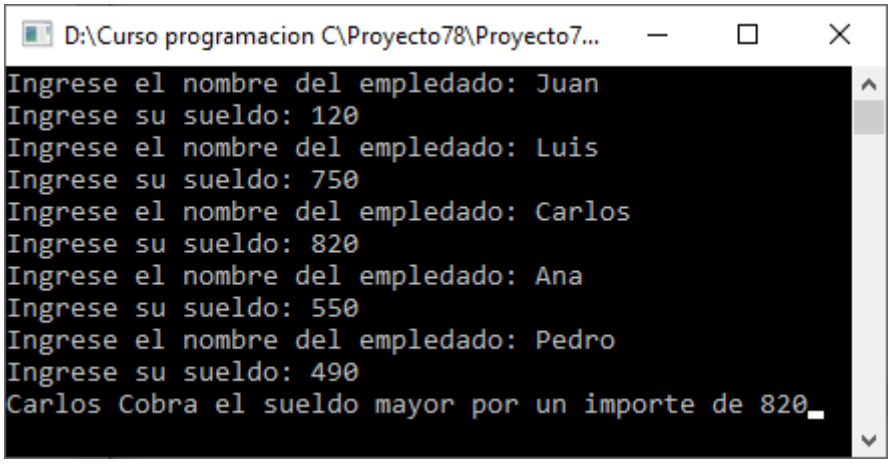
Confeccionar un programa que permita cargar los nombres de 5 operarios y sus sueldos respectivos.

Mostrar el sueldo mayor y en nombre del operario.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto78
{
    2 referencias
    class Empleado
    {
        string[] nombre;
        int[] sueldo;

        1 referencia
        public void cargar()
        {
            nombre = new string[5];
            sueldo = new int[5];
            string line;

            for (int f=0; f<5; f++)
            {
                Console.Write("Ingrese el nombre del empledado: ");
                nombre[f] = Console.ReadLine();
                Console.Write("Ingrese su sueldo: ");
                line = Console.ReadLine();
                sueldo[f] = int.Parse(line);
            }
        }
```

```
        1 referencia
        public void sueldoMayor()
        {
            int pos, mayor;
            mayor = sueldo[0];
            pos = 0;
            for (int f=1; f<5;f++)
            {
                if (mayor < sueldo[f])
                {
                    mayor = sueldo[f];
                    pos = f;
                }
            }
            Console.Write(nombre[pos] + " Cobra el sueldo mayor");
            Console.Write(" por un importe de " + sueldo[pos]);
        }

        0 referencias
        static void Main(string[] args)
        {
            Empleado empl = new Empleado();
            empl.cargar();
            empl.sueldoMayor();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 78.- Vectores (mayor y menor elemento) – 2

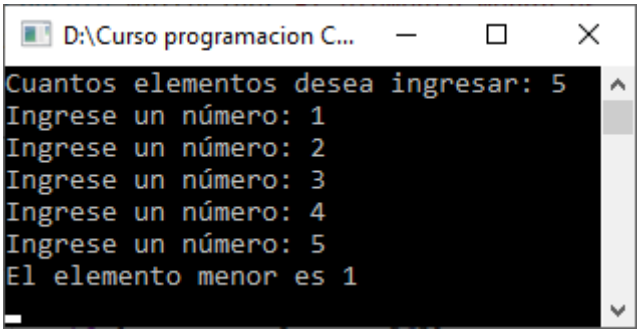
Problema propuesto

Crear un vector de n elementos. Imprimir el menor y un mensaje si se repite dentro del vector.

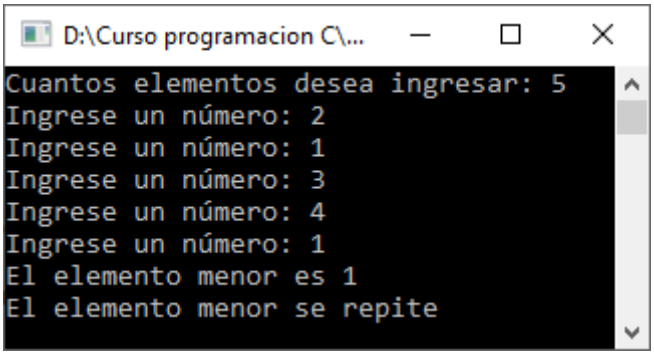
```
using System;
using System.Collections.Generic;
using System.Diagnostics.Eventing.Reader;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Schema;

namespace Proyecto79
{
    2 referencias
    class Vector
    {
        private int[] elemento;
        private int menor;

        1 referencia
        public void cargar()
        {
            int n;
            string line;
            Console.Write("Cuantos elementos desea ingresar: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            elemento = new int[n];
            for(int f = 0; f < n; f++)
            {
                Console.Write("Ingrese un número: ");
                line = Console.ReadLine();
                elemento[f] = int.Parse(line);
            }
        }
        1 referencia
        public void imprimir()
        {

            menor = elemento[0];
            for (int f=1; f<elemento.Length; f++)
            {
                if (elemento[f] < menor)
                {
                    menor = elemento[f];
                }
            }
            Console.WriteLine("El elemento menor es " +  menor);
            repetirMenor();
        }
```

```
        1 referencia
        public void repetirMenor()
        {
            int contador = 0;
            for (int f = 0; f < elemento.Length; f++)
            {
                if (menor == elemento[f])
                {
                    contador++;
                }
            }
            if (contador > 1)
            {
                Console.Write("El elemento menor se repite");
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            Vector v = new Vector();
            v.cargar();
            v.imprimir();
            Console.ReadKey();
        }
    }
}
```

Vamos a ejecutar sin repetir el número menor:



Ejecutamos de nuevo repitiendo el número menor:



Capítulo 79.- Vectores (ordenamiento) – 1

El ordenamiento de un vector se logra intercambiando los elementos de manera que:

vec[0] <= vec[1] <= vec[2] etc.

El contenido del elemento vec[0] sea menor o igual al contenido del elemento vec[1] y así sucesivamente.

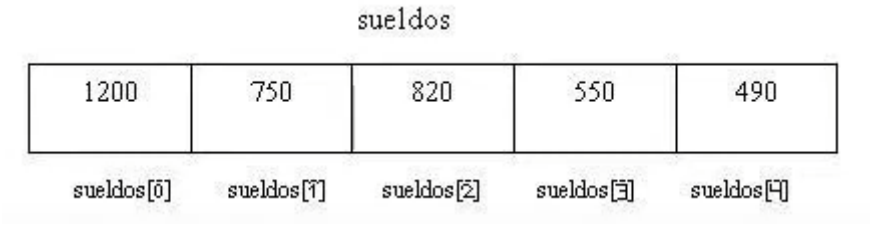
Si se cumple lo dicho anteriormente decimos que el vector está ordenado de menor a mayor.

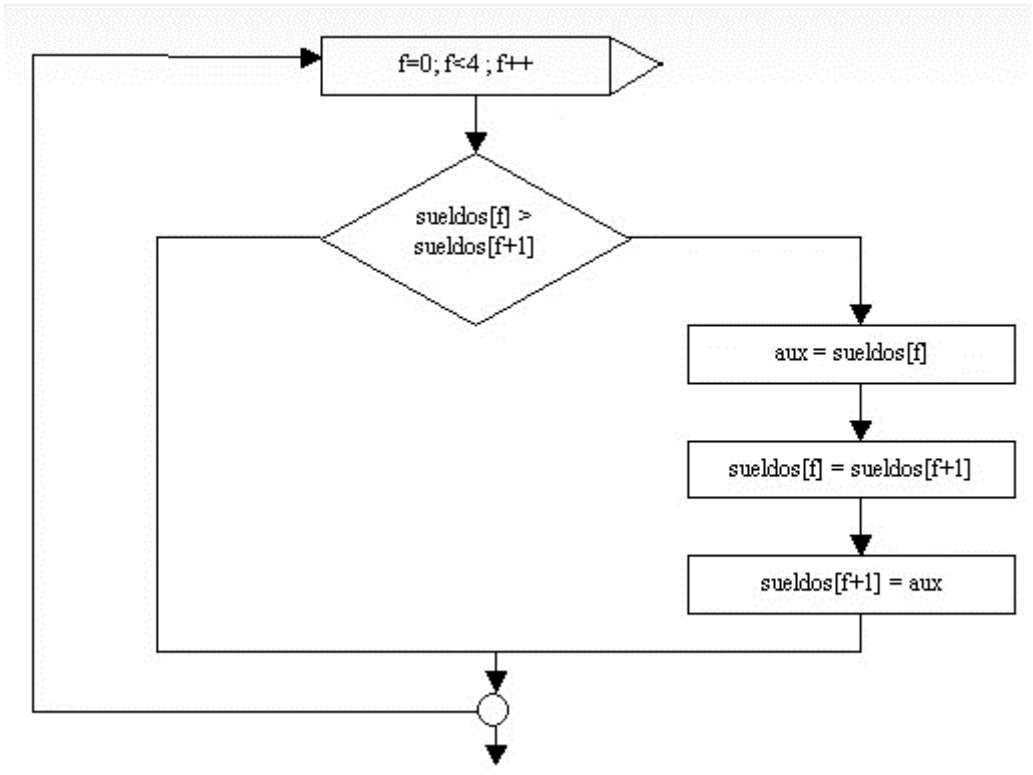
Igualmente podemos ordenar un vector de mayor a menor.

Se puede ordenar tanto vectores con componentes de tipo int, float como string. En este último caso el ordenamiento es alfabético.

Problema

Se debe crear un vector donde almacenar 5 sueldos. Ordenar el vector sueldos de menor a mayor.





Esta primera aproximación tiene por objetivo analizar los intercambios de elementos dentro del vector.

El algoritmo consiste en comparar si la primer elemento es mayor al segundo, en caso que la condición sea verdadera, intercambiamos los contenidos de los elementos.

Vamos a suponer que se ingresan los siguientes valores por teclado:

```
1200
750
820
550
490
```

En este ejemplo: ¿es 1200 mayor a 750? La respuesta es verdadera, por lo tanto intercambiamos el contenido del elemento 0 con el del elemento 1.

Luego comparamos el contenido del elemento 1 con el del elemento 2: ¿Es 1200 mayor a 820?.

La respuesta es verdadera entonces intercambiamos.

Si hay 5 elementos hay que hacer 4 comparaciones, por eso el for se repite 4 veces.

Generalizando: si el vector tiene N elementos hay que hacer N-1 comparaciones.

Cuando	f = 0	f = 1	f = 2	f = 3
	750	750	750	750
	1200	820	820	820
	820	1200	550	550
	550	550	1200	490
	490	490	490	1200

Podemos ver cómo el valor más grande del vector desciende al último elemento. Empleamos una variable auxiliar (aux) para el proceso de intercambio:

```
aux=sueldos[f];
sueldos[f]=sueldos[f+1];
sueldos[f+1]=aux;
```

Al salir del for en este ejemplo el contenido del vector es el siguiente:

```
750
820
550
490
1200
```

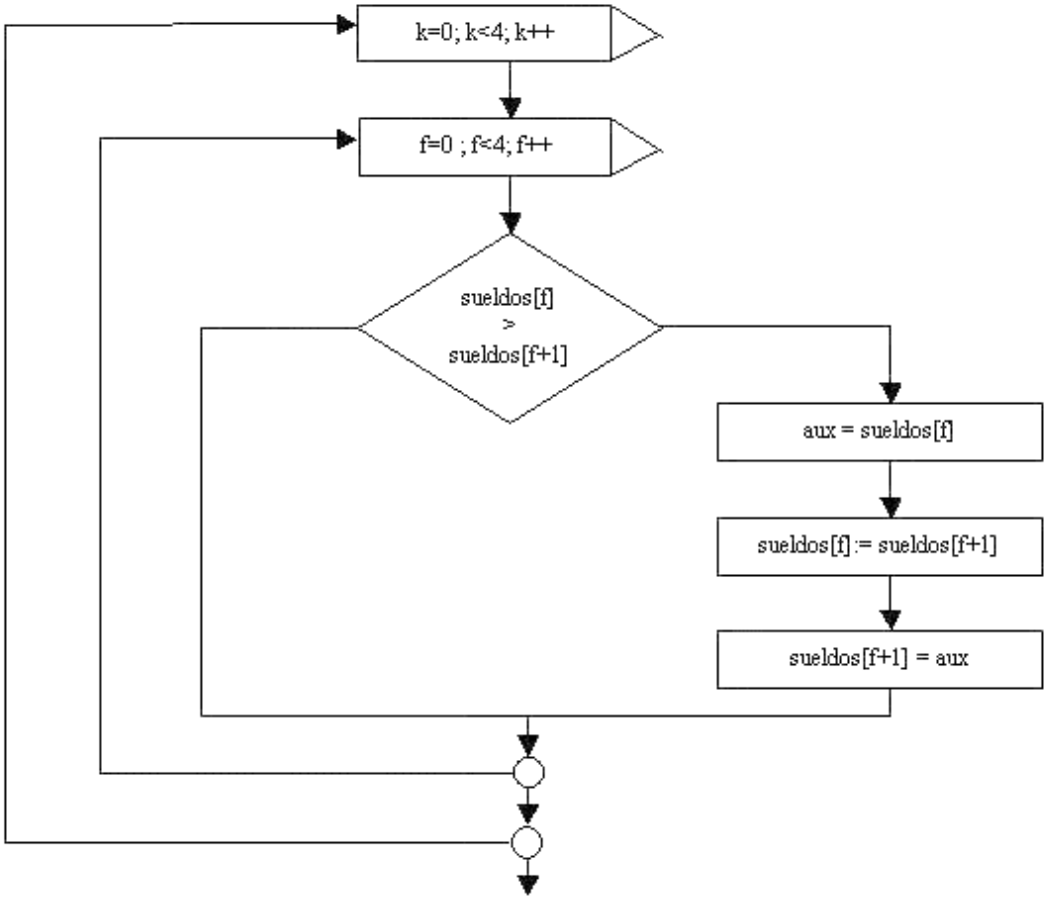
Analizando el algoritmo podemos comprobar que el elemento mayor del vector se ubica ahora en el último lugar.

Podemos definir otros vectores con distintos valores y comprobar que siempre el elemento mayor queda al final.

Pero todavía con este algoritmo no se ordena un vector. Solamente está ordenado el último elemento del vector.

Ahora bien, con los 4 elementos que nos quedan podemos hacer el mismo proceso visto anteriormente, con lo cual quedará ordenado otro elemento del vector. Este proceso lo repetiremos hasta que quede ordenado por completo el vector.

Como debemos repetir el mismo algoritmo podemos englobar todo el bloque en otra estructura repetitiva.



Realicemos una prueba del siguiente algoritmo:

Cuando k = 0

f = 0	f = 1	f = 2	f = 3
750	750	750	750
1200	820	820	820
820	1200	550	550
550	550	1200	490
490	490	490	1200

Cuando k = 1

f = 0	f = 1	f = 2	f = 3
750	750	750	750
820	550	550	550
550	820	490	490
490	490	820	820
1200	1200	1200	1200

Cuando k = 2

f = 0	f = 1	f = 2	f = 3
550	550	550	550
750	490	490	490
490	750	750	750
820	820	820	820
1200	1200	1200	1200

Cuando k = 3

f = 0	f = 1	f = 2	f = 3
490	490	490	490
550	550	550	550
750	750	750	750
820	820	820	820
1200	1200	1200	1200

¿Por qué repetimos 4 veces el for externo?

Como sabemos cada vez que se repite en forma completa el for interno queda ordenada un elemento del vector. A primera vista diríamos que deberíamos repetir el for externo la cantidad de elementos del vector, en este ejemplo el vector sueldos tiene 5 componentes.

Si observamos, cuando quedan dos elementos por ordenar, al ordenar uno de ellos queda el otro automáticamente ordenado (podemos imaginar que si tenemos un vector con 2 elementos no se requiere el for externo, porque este debería repetirse una única vez).

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando continuamos comparándolos.

Ejemplo: En la primera ejecución del for interno el valor 1200 queda ubicado en la posición 4 del vector. En la segunda ejecución comparamos si el 820 es mayor a 1200, lo cual seguramente será falso.

Podemos concluir que la primera vez debemos hacer para este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en una la cantidad de comparaciones.

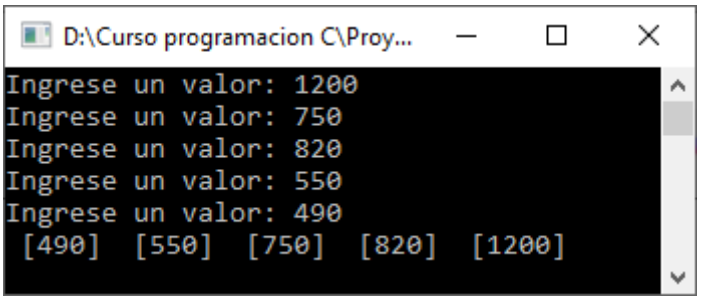
Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del anterior es el plantear un for interno con la siguiente estructura: (f=0; f<4-k; f++).

Es decir restarle el valor del contador del for externo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto80
{
    2 referencias
    class Ordenamiento
    {
        int[] vector;
```

```
        1 referencia
        public void cargar()
        {
            vector = new int[5];
            string line;
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                vector[f]=int.Parse(line);
            }
        }

        1 referencia
        public void ordenar()
        {
            int aux;
            for (int k=0; k<4; k++)
            {
                for (int f=0; f<4-k; f++)
                {
                    if (vector[f] > vector[f + 1])
                    {
                        aux = vector[f];
                        vector[f] = vector[f + 1];
                        vector[f + 1] = aux;
                    }
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f=0; f<5; f++)
            {
                Console.Write(" [" + vector[f] + "] ");
            }
            Console.WriteLine();
        }

        0 referencias
        static void Main(string[] args)
        {
            Ordenamiento or = new Ordenamiento();
            or.cargar();
            or.ordenar();
            or.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Curso programacion C\Proy...
Ingrese un valor: 1200
Ingrese un valor: 750
Ingrese un valor: 820
Ingrese un valor: 550
Ingrese un valor: 490
[490] [550] [750] [820] [1200]

Capítulo 80.- Vectores (ordenamiento) – 2

Ordenamiento de string

También podemos ordenar vectores cuyos elementos sean de tipo String. Para esto no podemos utilizar el operador > sino debemos utilizar un método de la clase String:

```
string cad1="juan";
string cad2="analia";
if (cad1.CompareTo(cad2)>0)
{
    Console.Write(cad1 + " es mayor alfabéticamente que " + cad2);
}
```

El método CompareTo retorna un valor mayor a cero si cad1 es mayor alfabéticamente. En este ejemplo cad1 tiene un valor alfabéticamente mayor a cad2, luego el CompareTo retorna un valor mayor de cero.

Si los dos string son exactamente iguales el método CompareTo retorna un cero, y finalmente si cad1 es menor alfabéticamente retorna un valor menor a cero.

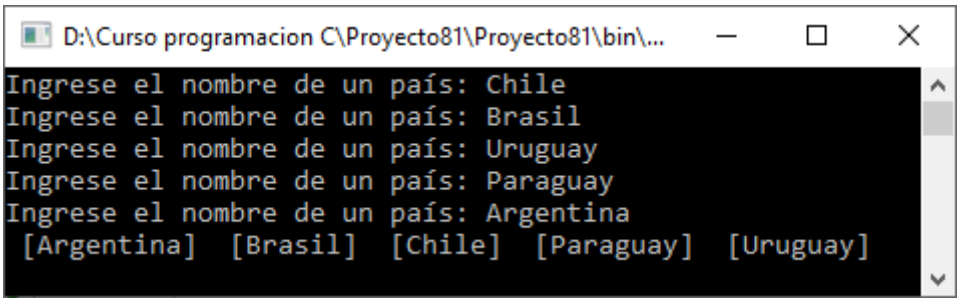
Problema

Definir un vector donde almacenar los nombres de 5 países. Confeccionar el algoritmo de ordenamiento alfabético.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto81
{
    2 referencias
    class Vector
    {
        private string[] pais;

        1 referencia
        public void cargar()
        {
            pais = new string[5];
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese el nombre de un país: ");
                pais[f] = Console.ReadLine();
            }
        }

        1 referencia
        public void ordenar()
        {
            string aux;
            for (int k=0; k<4; k++)
            {
```

```
            for (int f=0; f<4-k; f++)
            {
                if (pais[f].CompareTo(pais[f+1])>0)
                {
                    aux = pais[f];
                    pais[f] = pais[f+1];
                    pais[f+1] = aux;
                }
            }
        }
    }

    1 referencia
    public void imprimir()
    {
        for (int f=0; f<5; f++)
        {
            Console.Write(" [" + pais[f] + "] ");
        }
        Console.WriteLine();
    }

    0 referencias
    static void Main(string[] args)
    {
        Vector v = new Vector();
        v.cargar();
        v.ordenar();
        v.imprimir();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 81.- Vectores (ordenamiento) – 3

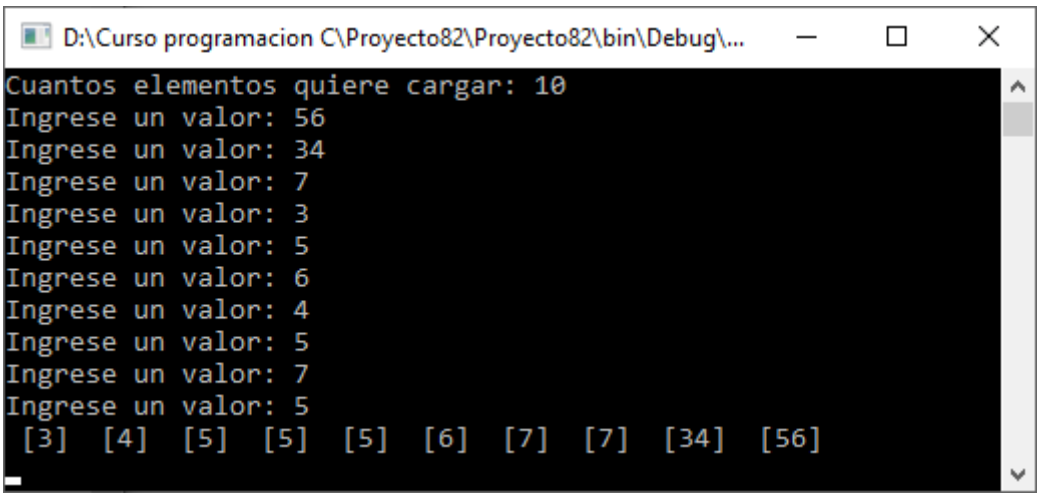
Problema propuesto

Cargar un vector de n elementos de tipo entero. Ordenar posteriormente el vector.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto82
{
    2 referencias
    class Vector
    {
        private int[] numero;

        1 referencia
        public void cargar()
        {
            int n;
            string line;
            Console.Write("Cuantos elementos quiere cargar: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            numero = new int[n];
            for (int f=0; f < n; f++)
            {
                Console.Write("Ingrese un valor: ");
                line = Console.ReadLine();
                numero[f] = int.Parse(line);
            }
        }

        1 referencia
        public void ordenar()
        {
            int aux;
            for (int k=0; k<numero.Length-1; k++)
            {
                for (int f=0; f<numero.Length-1-k ; f++)
                {
                    if (numero[f] > numero[f+1])
                    {
                        aux = numero[f];
                        numero[f] = numero[f+1];
                        numero[f+1] = aux;
                    }
                }
            }
        }
```

```
        1 referencia
        public void imprimir()
        {
            for (int f = 0; f < numero.Length; f++)
            {
                Console.Write(" [" + numero[f] + "] ");
            }
            Console.WriteLine();
        }

        0 referencias
        static void Main(string[] args)
        {
            Vector v = new Vector();
            v.cargar();
            v.ordenar();
            v.imprimir();
            Console.ReadKey();
        }
    }
}
```

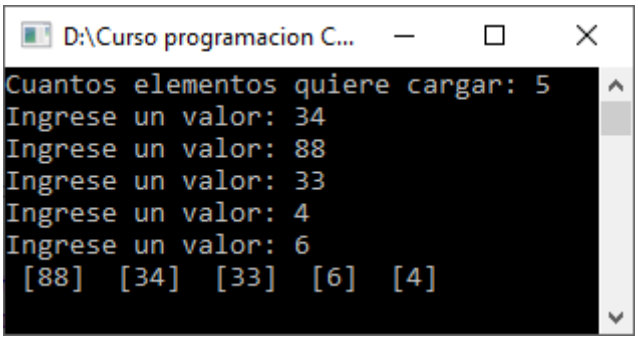
Si ejecutamos este será el resultado:




```
        1 referencia
        public void ordenar()
        {
            int aux;
            for (int k=0; k<numero.Length-1; k++)
            {
                for (int f=0; f<numero.Length-1-k ; f++)
                {
                    if (numero[f] < numero[f+1])
                    {
                        aux = numero[f];
                        numero[f] = numero[f+1];
                        numero[f+1] = aux;
```

```
            }
        }
    }
}
```

Ahora cambiamos el operador mayor por el operador de menor y vamos a ejecutar de nuevo.



La ordenación es en modo decreciente.

Capítulo 82.- Vectores (ordenamiento con vectores paralelos) – 1

Cuando se tienen vectores paralelos y se ordena uno de ellos hay que tener la precaución de intercambiar los elementos de los vectores paralelos.

Problema

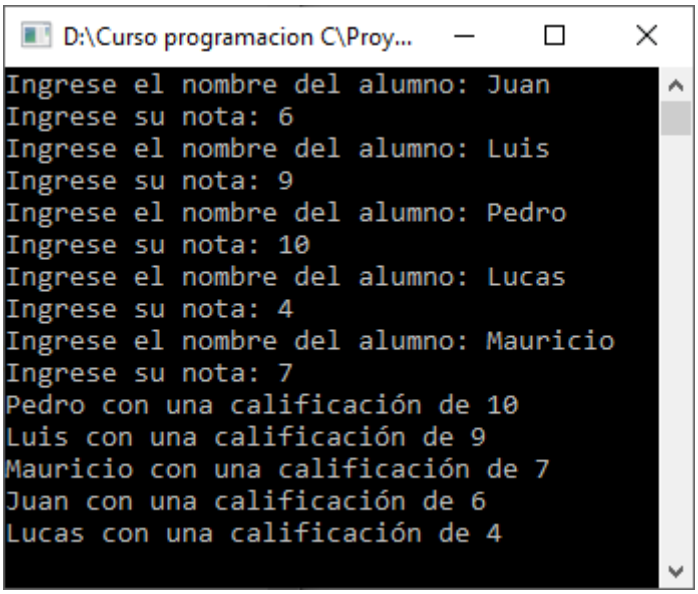
Confeccionar un programa que permita cargar los nombres de 5 alumnos y sus notas respectivas.

Luego ordenar las notas de mayor a menor. Imprimir las notas y los nombres de los alumnos.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto83
{
    2 referencias
    class Alumnos
    {
        private string[] nombre;
        private int[] nota;

        1 referencia
        public void cargar()
        {
            nombre = new string[5];
            nota = new int[5];
            string line;
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese el nombre del alumno: ");
                nombre[f]=Console.ReadLine();
                Console.Write("Ingrese su nota: ");
                line = Console.ReadLine();
                nota[f]= int.Parse(line);
            }
        }

        1 referencia
        public void ordenar()
        {
            int auxNota;
            string auxNombre;
            for (int k=0; k<4; k++)
            {
                for (int f=0; f<4-k; f++)
                {
                    if (nota[f] < nota[f+1])
                    {
                        auxNota = nota[f];
                        nota[f] = nota[f+1];
                        nota[f + 1] = auxNota;
```

```
                    auxNombre = nombre[f];
                    nombre[f] = nombre[f+1];
                    nombre[f+1] = auxNombre;
                }
            }
        }
    }

    1 referencia
    public void imprimir()
    {
        for (int f = 0; f < 5;f++)
        {
            Console.WriteLine(nombre[f] + " con una calificación de " + nota[f]);
        }
        Console.WriteLine();
    }

    0 referencias
    static void Main(string[] args)
    {
        Alumnos alu = new Alumnos();
        alu.cargar();
        alu.ordenar();
        alu.imprimir();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:

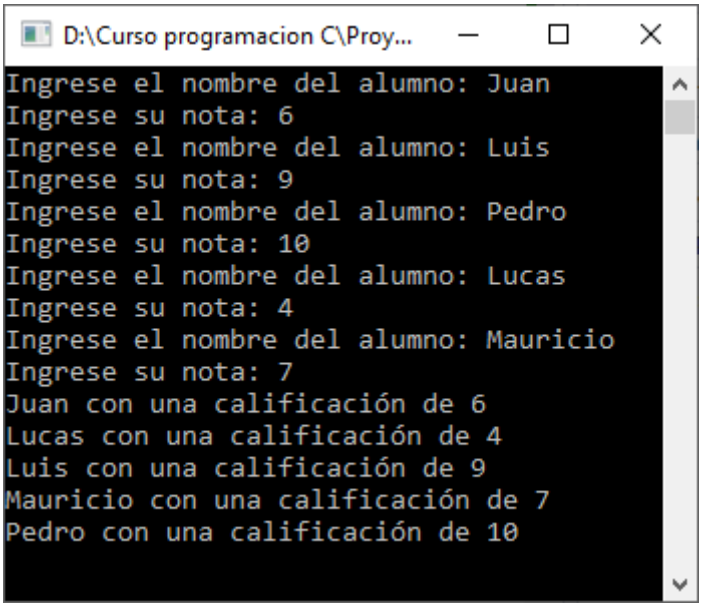


Ahora te propongo que modifiques el método ordenar para que me muestre ordenado alfabéticamente por el nombre del alumno.

```
    1 referencia
    public void ordenar()
    {
        int auxNota;
        string auxNombre;
        for (int k=0; k<4; k++)
        {
```

```
            for (int f=0; f<4-k; f++)
            {
                if (nombre[f].CompareTo(nombre[f+1])>0)
                {
                    auxNota = nota[f];
                    nota[f] = nota[f+1];
                    nota[f + 1] = auxNota;
                    auxNombre = nombre[f];
                    nombre[f] = nombre[f+1];
                    nombre[f+1] = auxNombre;
                }
            }
        }
    }
```

Si ejecutamos este será el resultado:



Capítulo 83.- Vectores (ordenamiento con vectores paralelos) – 2

Problema propuesto

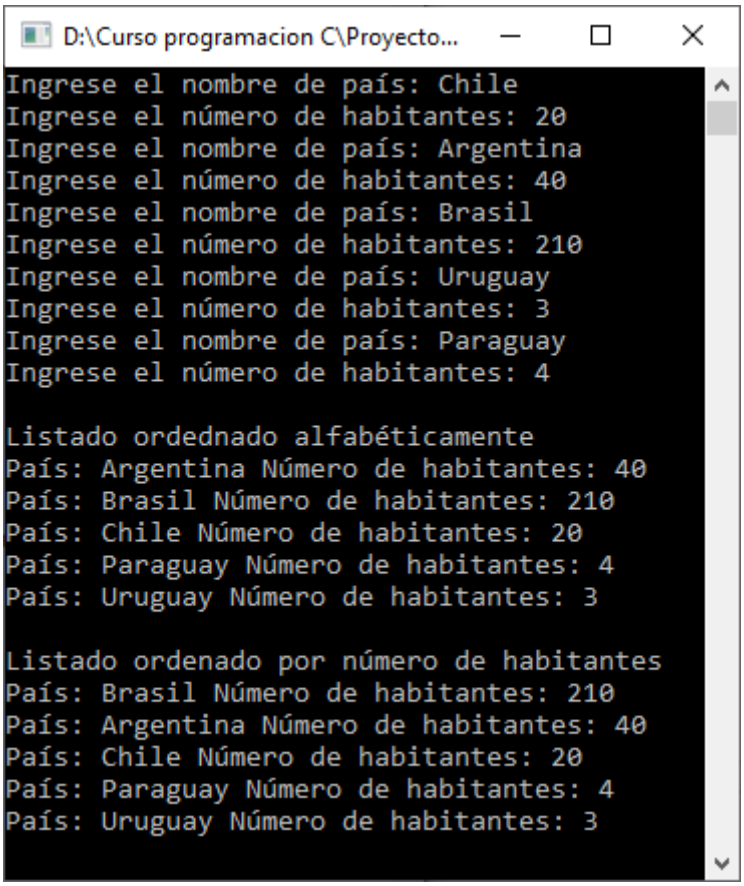
Cargar en un vector los nombres de 5 países y en otro vector paralelo la cantidad de habitantes del mismo. Ordenar alfabéticamente e imprimir los resultados.

Por último ordenar con respecto a la cantidad de habitantes (de mayor a menor) e imprimir nuevamente.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Runtime.CompilerServices;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto84
{
    2 referencias
    class Paises
    {
        private string[] pais;
        private int[] habitantes;

        1 referencia
        public void cargar()
        {
            string line;
            pais = new string[5];
            habitantes = new int[5];
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese el nombre de país: ");
                pais[f]= Console.ReadLine();
                Console.Write("Ingrese el número de habitantes: ");
                line = Console.ReadLine();
                habitantes[f]= int.Parse(line);
            }
            Console.WriteLine();
        }

        1 referencia
        public void ordenAlfabeticamente()
        {
            string auxPais;
            int auxHabitantes;
            for (int k=0; k<4; k++)
            {
                for (int f=0; f<4-k; f++)
                {
                    if (pais[f].CompareTo(pais[f+1])>0)
                    {
                        auxPais = pais[f];
```

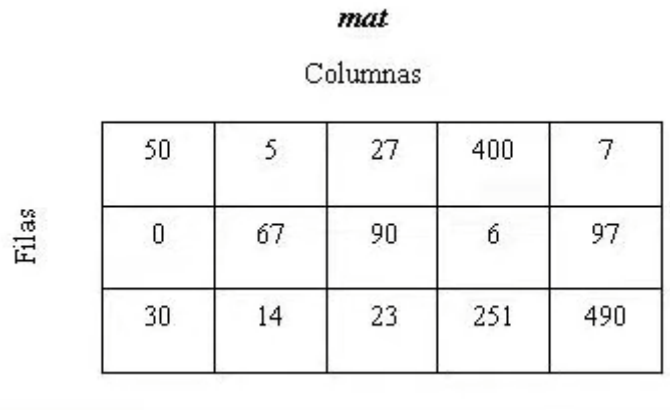
```
                    pais[f] = pais[f+1];
                    pais[f+1] = auxPais;
                    auxHabitantes = habitantes[f];
                    habitantes[f]= habitantes[f + 1];
                    habitantes[f+1]=auxHabitantes;
                }
            }
        }
    }
    1 referencia
    public void ordenarNumHabitantes()
    {
        string auxPais;
        int auxHabitantes;
        for (int k = 0; k < 4; k++)
        {
            for (int f = 0; f < 4 - k; f++)
            {
                if (habitantes[f] < habitantes[f+1])
                {
                    auxPais = pais[f];
                    pais[f] = pais[f + 1];
                    pais[f + 1] = auxPais;
                    auxHabitantes = habitantes[f];
                    habitantes[f] = habitantes[f + 1];
                    habitantes[f + 1] = auxHabitantes;
                }
            }
        }
    }
    2 referencias
    public void imprimir()
    {
        for (int f=0; f < 5; ++f)
        {
            Console.WriteLine("País: " + pais[f] + " Número de habitantes: "
                + habitantes[f]);
        }
        Console.WriteLine();
    }

    0 referencias
    static void Main(string[] args)
    {
        Paises pais1 = new Paises();
        pais1.cargar();
        Console.WriteLine("Listado ordednado alfabéticamente");
        pais1.ordenAlfabeticamente();
        pais1.imprimir();
        Console.WriteLine("Listado ordenado por número de habitantes");
        pais1.ordenarNumHabitantes();
        pais1 .imprimir();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 84.- Estructura de datos tipo matriz – 1

Una matriz es una estructura de datos que permite almacenar un CONJUNTO de datos del MISMO tipo. Con un único nombre se define la matriz por medio de dos subíndices hacemos referencia a cada elemento de la misma (componente).



Hemos graficado una matriz en 3 filas y 5 columnas. Para hacer referencia a cada elemento debemos indicar primero la fila y luego la columna, por ejemplo el elemento 1, 4 se almacena el valor 97.

En este ejemplo almacenamos valores enteros. Todos los elementos de la matriz deben ser del mismo tipo (int, float, string, etc.).

Las filas y las columnas comienzan a numerarse a partir de cero, similar a los vectores.

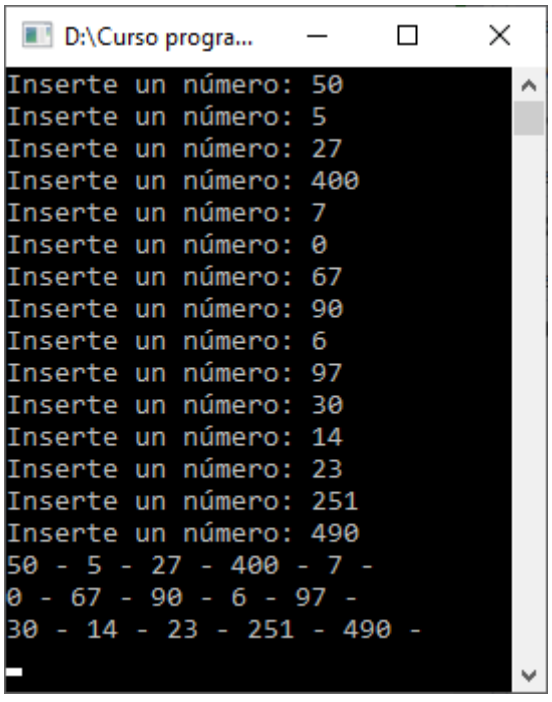
Problema

Crear una matriz de 3 filas por 5 columnas con elementos de tipo int, cargar sus componentes y luego imprimirlas.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization.Formatters;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto85
{
    2 referencias
    class Matriz
    {
        private int[,] mat;

        1 referencia
        public void cargar()
        {
            mat = new int[3,5];
            string line;
            for (int f=0; f<3; f++)
            {
                for (int c=0; c<5; c++)
                {
```

```
                Console.Write("Inserte un número: ");
                line = Console.ReadLine();
                mat[f,c] = int.Parse(line);
            }
        }
    }

    1 referencia
    public void imprimir()
    {
        for (int f=0; f<3; ++f)
        {
            for (int c=0; c<5; ++c)
            {
                Console.Write(mat[f,c] + " - ");
            }
            Console.WriteLine();
        }
    }

    0 referencias
    static void Main(string[] args)
    {
        Matriz m = new Matriz();
        m.cargar();
        m.imprimir();
        Console.ReadKey();
    }
}
}
```

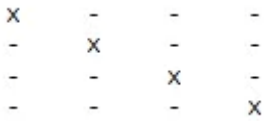
Si ejecutamos este será el resultado:



Capítulo 85.- Estructura de datos tipo matriz – 2

Problema

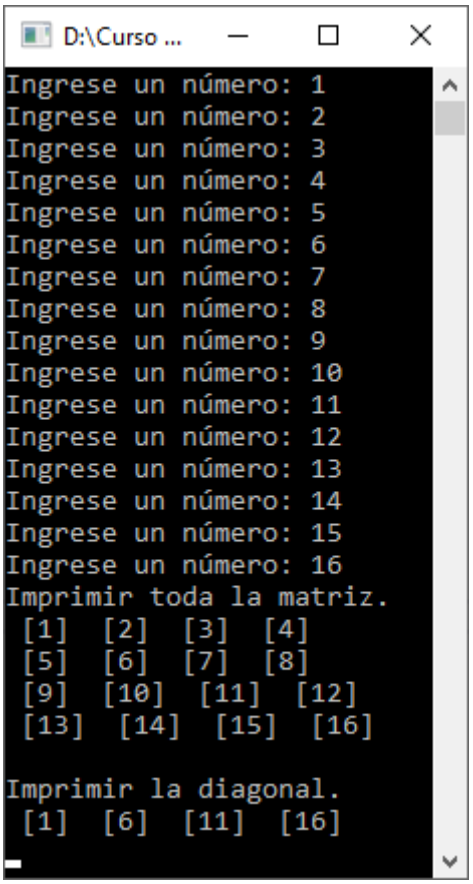
Crear y cargar una matriz de enteros de 4 filas por 4 columnas. Imprimir la diagonal principal.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto86
{
    2 referencias
    class Matriz
    {
        private int[,] mat;

        1 referencia
        public void cargar()
        {
            mat = new int[4, 4];
            string line;
            for (int f=0; f < 4; f++)
            {
                for(int c=0; c < 4; c++)
                {
                    Console.Write("Ingrese un número: ");
                    line = Console.ReadLine();
                    mat[f,c]=int.Parse(line);
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f=0;f < 4; f++)
            {
                for (int c = 0; c < 4; c++)
                {
                    Console.Write(" [" + mat[f, c] + "] ");
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }
```

```
        1 referencia
        public void imprimirDiagonal()
        {
            for (int f=0; f < 4; f++)
            {
                Console.Write(" [" + mat[f, f] + "] ");
            }
            Console.WriteLine();
        }

        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            Console.WriteLine("Imprimir toda la matriz.");
            m.imprimir();
            Console.WriteLine("Imprimir la diagonal.");
            m.imprimirDiagonal();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 86.- Estructura de datos tipo matriz – 3

Problema

Crear y cargar una matriz de 3 filas por 4 columnas. Imprimir la primera fila. Imprimir la última fila e imprimir la primera columna.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto87
{
    2 referencias
    class Matriz
    {
        private int[,] mat;

        1 referencia
        public void cargar()
        {
            mat = new int[3, 4];
            string line;
            for (int f=0; f<3; f++)
            {
                for (int c=0; c<4; c++)
                {
                    Console.Write("Ingrese un número: ");
                    line = Console.ReadLine();
                    mat[f,c]=int.Parse(line);
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f=0; f<3;f++)
            {
                for (int c=0; c<4;c++)
                {
                    Console.Write(" [" + mat[f,c] + "]");
                }
                Console.WriteLine();
            }
        }
```

```
        1 referencia
        public void imprimirPriFila()
        {
            Console.WriteLine("Imprimiendo la primera fila.");
            for (int c=0; c<4;c++)
            {
                Console.Write(" [" + mat[0,c] + "] ");
            }
            Console.WriteLine();
        }

        1 referencia
        public void imprimirUltFila()
        {
            Console.WriteLine("Imprimiendo la última fila.");
            for (int c = 0; c < 4; c++)
            {
                Console.Write(" [" + mat[2,c] + "] ");
            }
            Console.WriteLine();
        }

        1 referencia
        public void imprimirPriColumna()
        {
            Console.WriteLine("Imprimiendo la primera columan.");
            for (int f=0; f<3; f++)
            {
                Console.WriteLine(" [" + mat[f,0] + "] ");
            }
            Console.WriteLine();
        }
        0 referencias
        static void Main(string[] args)
        {
            Matriz ma = new Matriz();
            ma.cargar();
            ma.imprimir();
            ma.imprimirPriFila();
            ma.imprimirUltFila();
            ma.imprimirPriColumna();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

```
D:\Curso program...
Ingrese un número: 1
Ingrese un número: 2
Ingrese un número: 3
Ingrese un número: 4
Ingrese un número: 5
Ingrese un número: 6
Ingrese un número: 7
Ingrese un número: 8
Ingrese un número: 9
Ingrese un número: 10
Ingrese un número: 11
Ingrese un número: 12
 [1] [2] [3] [4]
 [5] [6] [7] [8]
 [9] [10] [11] [12]
Imprimiendo la primera fila.
 [1]  [2]  [3]  [4]
Imprimiendo la última fila.
 [9]  [10]  [11]  [12]
Imprimiendo la primera columan.
 [1]
 [5]
 [9]
```

Capítulo 87.- Estructura de datos tipo matriz – 4

Problema propuesto

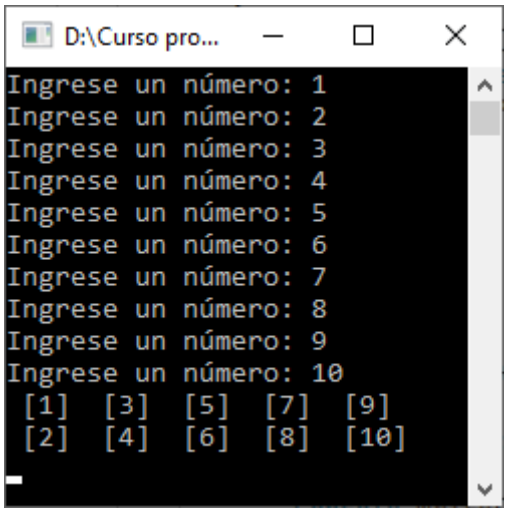
Crear una matriz de 2 filas y 5 columnas. Realizar la carga de componentes por columna (es decir primero ingresar la primera columna, luego la segunda columna y así sucesivamente).

Imprimir luego la matriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto88
{
    2 referencias
    class Matriz
    {
        private int[,] mat;

        1 referencia
        public void cargar()
        {
            mat = new int [2,5];
            string line;
            for (int c=0; c<5; c++)
            {
                for (int f=0; f<2; f++)
                {
                    Console.Write("Ingrese un número: ");
                    line = Console.ReadLine();
                    mat[f,c]=int.Parse(line);
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f=0; f<2; f++)
            {
                for (int c=0; c<5;c++)
                {
                    Console.Write(" [" + mat[f,c] + "] ");
                }
                Console.WriteLine();
            }
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            m.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 88.- Matrices (cantidad de filas y columnas) – 1

Como hemos visto para definir y crear la matriz utilizamos la siguientes sintaxis:

```
int[,] mat;
```

Creación:

```
mat=new int[3,4];
```

Como las matrices son objetos en C# disponemos de un método llamado GetLength que le pasamos como parámetro la dimensión y nos retorna el valor de dicha dimensión.

Si queremos saber la cantidad de filas que tiene la matriz debemos llamar al método GetLength con el valor cero:

```
Console.WriteLine("Cantidad de filas de la matriz:" + mat.GetLength(0));
```

Si queremos saber la cantidad de columnas luego:

```
Console.WriteLine("Cantidad de columnas de la matriz:" + mat.GetLength(1));
```

La primera dimensión son la cantidad de filas y la segunda dimensión son la cantidad de columnas de la matriz.

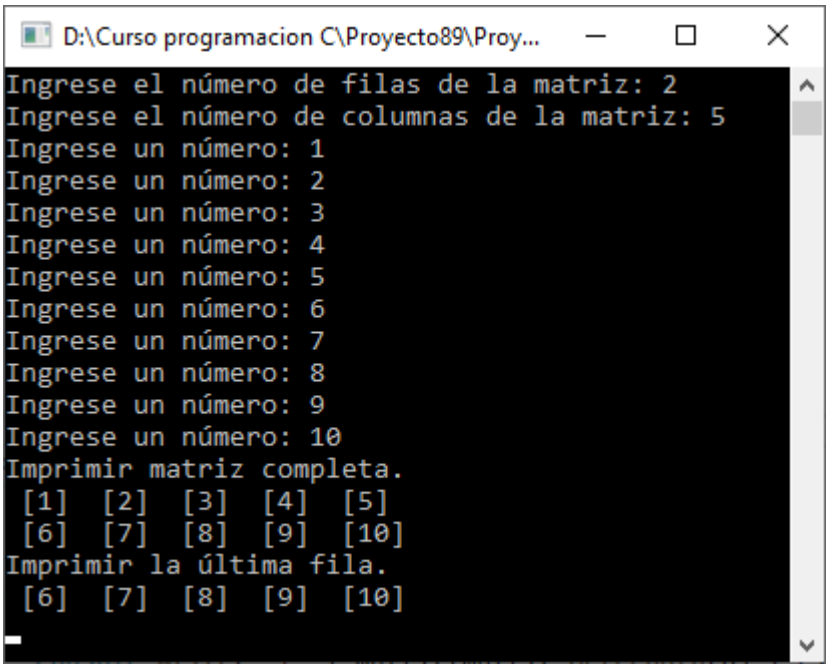
Problema

Crear una matriz de n filas * m columnas (cargar n y m por teclado). Imprimir la matriz completa y la última fila.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto89
{
    2 referencias
    class Matriz
    {
        private int[,] matriz;

        1 referencia
        public void cargar()
        {
            string line;
            int n, m;
            Console.Write("Ingrese el número de filas de la matriz: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            Console.Write("Ingrese el número de columnas de la matriz: ");
            line = Console.ReadLine();
            m = int.Parse(line);
            matriz = new int[n, m];
```

```
            for (int f= 0; f < n; f++)
            {
                for (int c= 0; c < m; c++)
                {
                    Console.Write("Ingrese un número: ");
                    line = Console.ReadLine();
                    matriz[f,c] = int.Parse(line);
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            Console.WriteLine("Imprimir matriz completa.");
            for (int f=0; f<matriz.GetLength(0); f++)
            {
                for (int c= 0; c<matriz.GetLength(1); c++)
                {
                    Console.Write(" [" + matriz[f, c] + "] ");
                }
                Console.WriteLine();
            }
        }

        1 referencia
        public void imprimirUltFila()
        {
            Console.WriteLine("Imprimir la última fila.");
            for (int c = 0; c < matriz.GetLength(1); c++)
            {
                Console.Write(" [" + matriz[matriz.GetLength(0)-1,c] + "] ");
            }
            Console.WriteLine() ;
        }

        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            m.imprimir();
            m.imprimirUltFila();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 89.- Matrices (cantidad de filas y columnas) – 2

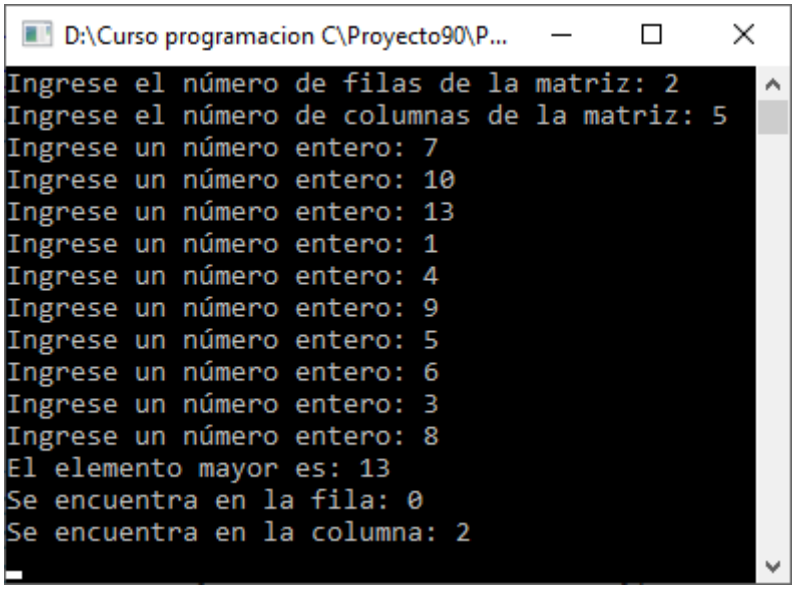
Problema

Crear una matriz de n * m (cargar n y m por teclado) imprimir el mayor elemento y la fila y columna donde se almacena.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto90
{
    2 referencias
    class Matriz
    {
        private int[,] matriz;

        1 referencia
        public void cargar()
        {
            int n, m;
            string line;
            Console.Write("Ingrese el número de filas de la matriz: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            Console.Write("Ingrese el número de columnas de la matriz: ");
            line = Console.ReadLine();
            m = int.Parse(line);
            matriz = new int[n, m];
            for (int f = 0; f < n; f++)
            {
                for (int c = 0; c < m; c++)
                {
                    Console.Write("Ingrese un número entero: ");
                    line = Console.ReadLine();
                    matriz[f,c] = int.Parse(line);
                }
            }
        }

        1 referencia
        public void MayorPosicion()
        {
            int max = 0;
            int fila=0, columna=0;
            for (int f=0; f<matriz.GetLength(0); f++)
            {
                for (int c=0; c<matriz.GetLength(1); c++)
                {
                    if (matriz[f,c]>max)
                    {
                        max = matriz[f,c];
                        fila = f;
                        columna = c;
```

```
                }
            }
        }
        Console.Write("El elemento mayor es: ");
        Console.WriteLine(max);
        Console.Write("Se encuentra en la fila: ");
        Console.WriteLine(fila);
        Console.Write("Se encuentra en la columna: ");
        Console.WriteLine(columna);
    }

    0 referencias
    static void Main(string[] args)
    {
        Matriz m = new Matriz();
        m.cargar();
        m.MayorPosicion();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 90.- Matrices (cantidad de filas y columnas) – 3

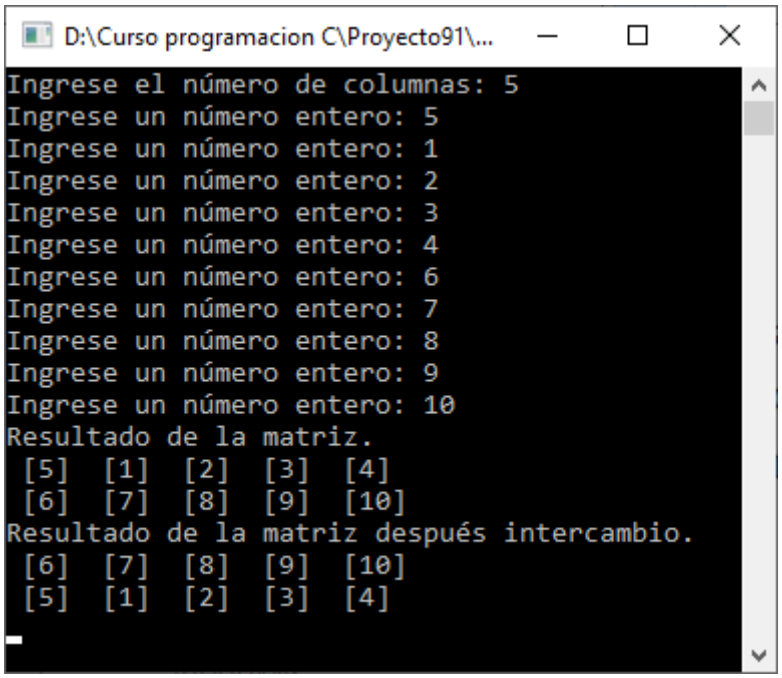
Problema propuesto

Crear una matriz de n filas * m columnas (cargar n y m por teclado) intercambiar la primera fila con la segunda. Imprimir luego la matriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto91
{
    2 referencias
    class Matriz
    {
        private int[,] matriz;

        1 referencia
        public void cargar()
        {
            int n, m;
            string line;
            Console.Write("Ingrese el número de filas: ");
            line = Console.ReadLine();
            n = int.Parse(line);
            Console.Write("Ingrese el número de columnas: ");
            line = Console.ReadLine();
            m = int.Parse(line);
            matriz = new int[n, m];
            for (int f=0; f < n; f++)
            {
                for (int c=0; c < m; c++)
                {
                    Console.Write("Ingrese un número entero: ");
                    line= Console.ReadLine();
                    matriz[f, c] = int.Parse(line);
                }
            }
        }

        1 referencia
        public void intercambio()
        {
            int aux;
            for (int c=0; c<matriz.GetLength(1); c++)
            {
                aux = matriz[0,c];
                matriz[0,c] = matriz[1,c];
                matriz[1,c] = aux;
            }
        }
```

```
        2 referencias
        public void imprimir()
        {
            for (int f=0; f<matriz.GetLength(0); f++)
            {
                for (int c=0;c<matriz.GetLength(1);c++)
                {
                    Console.Write(" [" + matriz[f, c] + "] ");
                }
                Console.WriteLine();
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            Console.WriteLine("Resultado de la matriz.");
            m.imprimir();
            m.intercambio();
            Console.WriteLine("Resultado de la matriz después intercambio.");
            m.imprimir();
            Console.ReadKey();
        }
    }
}
```

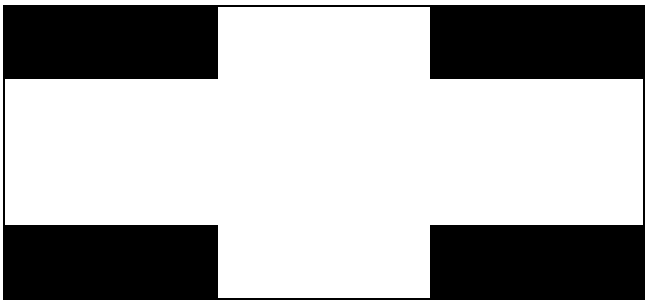
Si ejecutamos este será el resultado:



Capítulo 91.- Matrices (cantidad de filas y columnas) – 4

Problema propuesto

Crear una matriz de n filas * m columnas (cargar n y m por teclado). Imprimir los cuatro valores que es encuentran en los vértices de la misma (mat[0,0] etc.).



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto92
{
    2 referencias
    class Matriz
    {
        private int[,] matriz;

        1 referencia
        public void cargar()
        {
            int fila, columna;
            string line;
            Console.Write("Ingrese el número de filas: ");
            line = Console.ReadLine();
            fila = int.Parse(line);
            Console.Write("Ingrese el número de columnas: ");
            line = Console.ReadLine();
            columna = int.Parse(line);
            matriz = new int[fila, columna];
            for (int f=0; f<fila; f++)
            {
                for (int c=0; c<columna; c++)
                {
                    Console.Write("Ingrese un número: ");
                    line = Console.ReadLine();
                    matriz[f,c] = int.Parse(line);
                }
            }
        }
```

```
        1 referencia
        public void imprimir()
        {
            for (int f=0; f< matriz.GetLength(0); f++)
            {
                for (int c=0;c<matriz.GetLength(1); c++)
                {
                    Console.Write(" [" + matriz[f,c] + "] ");
                }
                Console.WriteLine();
            }
        }

        1 referencia
        public void imprimirVertices()
        {
            Console.WriteLine("Imprimr los vértices.");
            Console.WriteLine("Vertice superior izquerdo: "
                + matriz[0,0]);
            Console.WriteLine("Vertice superior derecho: "
                + matriz[0, matriz.GetLength(1) - 1]);
            Console.WriteLine("Vertice inferior izqueirdo: "
                + matriz[matriz.GetLength(0) - 1, 0]);
            Console.WriteLine("Vertice inferior derecho: "
                + matriz[matriz.GetLength(0) - 1, matriz.GetLength(1) - 1]);
        }

        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            m.imprimir();
            m.imprimirVertices();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

```
D:\Curso programacio...
Ingrese el número de filas: 3
Ingrese el número de columnas: 4
Ingrese un número: 1
Ingrese un número: 2
Ingrese un número: 3
Ingrese un número: 4
Ingrese un número: 5
Ingrese un número: 6
Ingrese un número: 7
Ingrese un número: 8
Ingrese un número: 9
Ingrese un número: 10
Ingrese un número: 11
Ingrese un número: 12
 [1]  [2]  [3]  [4]
 [5]  [6]  [7]  [8]
 [9]  [10]  [11]  [12]
Imprimr los vértices.
Vertice superior izquerdo: 1
Vertice superior derecho: 4
Vertice inferior izqueirdo: 9
Vertice inferior derecho: 12
```

Capítulo 92.- Matrices y vectores paralelos – 1

Dependiendo de la complejidad del problema podemos necesitar el empleo de vectores y matrices paralelos.

Problema

Se tiene la siguiente información:

- Nombre de 4 empleados.
- Ingresos en concepto de sueldo por cada empleado en los últimos 3 meses.

Confeccionar el programa para:

a) Realizar la carga de la información mencionada.
b) Generar un vector que contenga el ingreso acumulado en sueldos en los últimos 3 meses para cada empleado.
c) Mostrar por pantalla el total pagado en sueldos a todos los empleados en los últimos 3 meses.
d) Obtener el nombre del empleado que tuvo el mayor ingreso acumulado.

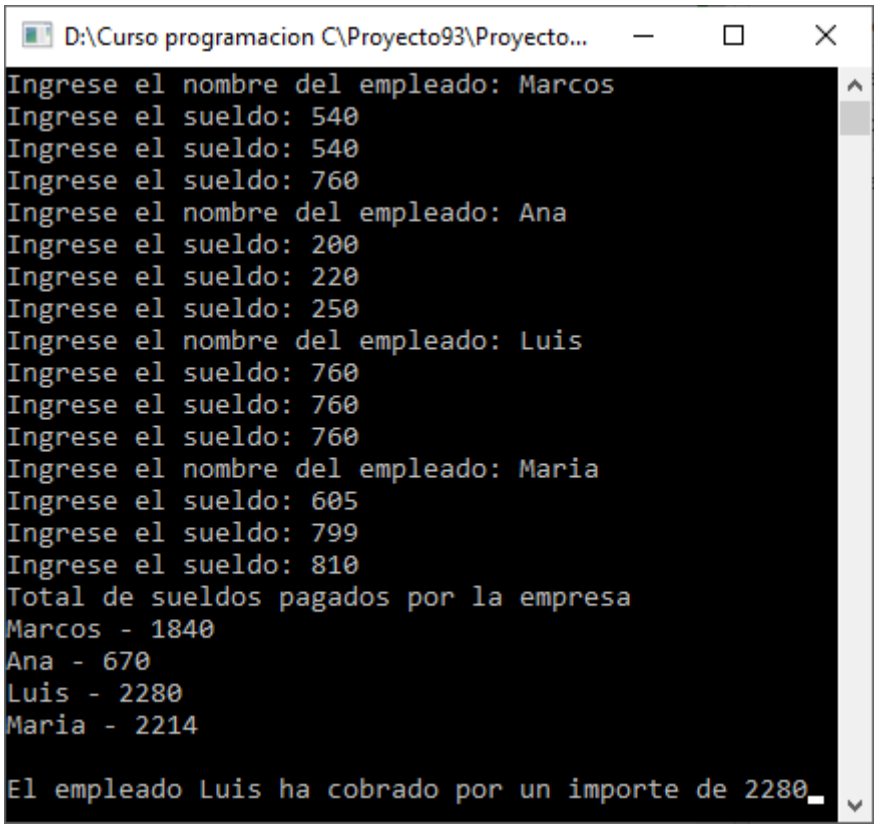
empleados	sueldos			sueldostot
Marcos	540	540	760	
Ana	200	220	250	
Luis	760	760	760	
María	605	799	810	

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto93
{
    2 referencias
    class Empresa
    {
        private string[] empleados;
        private int[,] sueldos;
        private int[] sueldostot;

        1 referencia
        public void cargar()
        {
            empleados = new string[4];
            sueldos = new int[4,3];
            for (int f = 0; f < 4; f++)
```

```
        {
            Console.Write("Ingrese el nombre del empleado: ");
            empleados[f]=Console.ReadLine();
            for (int c = 0; c < 3; c++)
            {
                Console.Write("Ingrese el sueldo: ");
                sueldos[f,c] = int.Parse(Console.ReadLine());
            }
        }
    }

    1 referencia
    public void sueldoTri()
    {
        int suma;
        sueldostot = new int[4];
        for (int f = 0;f < 4; f++)
        {
            suma = 0;
            for (int c = 0;c < 3; c++)
            {
                suma = suma + sueldos[f, c];
            }
            sueldostot[f] = suma;
            suma = 0;
        }
    }

    1 referencia
    public void totalSueldos()
    {
        Console.WriteLine("Total de sueldos pagados por la empresa");
        for (int f = 0; f<4; f++)
        {
            Console.WriteLine(empleados[f] + " - " + sueldostot[f]);
        }
        Console.WriteLine();
    }

    1 referencia
    public void mayorSueldoTri()
    {
        int max = 0;
        int pos = 0;
        for (int f=0; f < 4; f++)
        {
            if (sueldostot[f] > max)
            {
                max= sueldostot[f];
                pos = f;
            }
        }
        Console.Write("El empleado " + empleados[pos]
            + " ha cobrado por un importe de " + sueldostot[pos]);
    }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Empresa e = new Empresa();
            e.cargar();
            e.sueldoTri();
            e.totalSueldos();
            e.mayorSueldoTri();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 93.- Matrices y vectores paralelos – 2

Problema propuesto

Se desea saber la temperatura media trimestral de cuatro países. Para ello se tiene como dato las temperaturas medias mensuales de dichos países.

Se debe ingresar el nombre del país y seguidamente las tres temperaturas medias mensuales.

Seleccionar una estructura de datos adecuadas para el almacenamiento de los datos en memoria.

a- Cargar por teclado los nombres de los países y las temperaturas medias mensuales.
b- Imprimir los nombres de los países y las temperaturas medias mensuales de las mismas.
c- Calcular la temperatura media trimestral de cada país.
d- Imprimir los nombres de los países y las temperaturas medias trimestrales.
e- Imprimir el nombre del país con la temperatura media trimestral mayor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Security.Permissions;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto94
{
    2 referencias
    class Temperatura
    {
        private string[] pais;
        private int[,] temp;
        private int[] tempMedia;

        1 referencia
        public void cargar()
        {
            pais = new string[4];
            temp = new int[4,3];
            for (int f = 0; f < 4; f++)
            {
                Console.Write("Ingrese el nombre del país: ");
                pais[f]=Console.ReadLine();
                for (int c = 0; c < 3; c++)
                {
                    Console.Write("Ingrese las temperaturas medias del mes: ");
                    temp[f,c]=int.Parse(Console.ReadLine());
                }
            }
            Console.WriteLine();
        }
```

```
1 referencia
public void imprimir()
{
    Console.WriteLine("Listado de paises y sus temperaturas");
    Console.WriteLine("-----------------------------------");
    for (int f = 0; f < 4; f++)
    {
        Console.Write(pais[f] + " Temperaturas: ");
        for (int c = 0;c < 3; c++)
        {
            Console.Write(temp[f, c] + " - ");
        }
        Console.WriteLine() ;
    }
}

1 referencia
public void temperaturaMedia()
{
    int suma;
    tempMedia =  new int[4];
    for (int f = 0;f < 4; f++)
    {
        suma = 0;
        for (int c = 0;c < 3; c++)
        {
            suma = suma + temp[f,c];
        }
        tempMedia[f] = suma / 3;
        suma = 0;
    }
}
```

```
1 referencia
public void imprimirMedia()
{
    Console.WriteLine("Listado pasies y su tempertura media trimestral");
    Console.WriteLine("-----------------------------------------------");
    for (int f = 0; f<4; f++)
    {
        Console.WriteLine(pais[f] + " - " + tempMedia[f]);
    }
    Console.WriteLine();
}
```

```
1 referencia
public void TempMediaMayor()
{
    int mayor = 0;
    int pos = 0;
    for (int f = 0; f < 4; f++)
    {
        if (tempMedia[f] > mayor)
        {
            mayor = tempMedia[f];
            pos = f;
        }
```

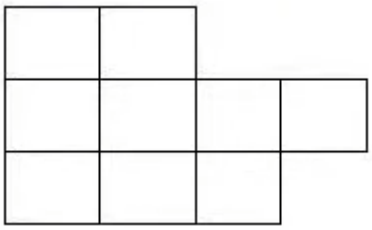
```
            }
            Console.WriteLine("El país " + pais[pos]
                + " tiene una temperatura media mayor: " + tempMedia[pos]);
        }

        0 referencias
        static void Main(string[] args)
        {
            Temperatura t = new Temperatura();
            t.cargar();
            t.imprimir();
            t.temperaturaMedia();
            t.imprimirMedia();
            t.TempMediaMayor();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

```
D:\Curso programacion C\Proyecto94\Proyecto94\...
Ingrese el nombre del país: Chile
Ingrese las temperaturas medias del mes: 22
Ingrese las temperaturas medias del mes: 23
Ingrese las temperaturas medias del mes: 24
Ingrese el nombre del país: Argentina
Ingrese las temperaturas medias del mes: 25
Ingrese las temperaturas medias del mes: 25
Ingrese las temperaturas medias del mes: 25
Ingrese el nombre del país: Uruguay
Ingrese las temperaturas medias del mes: 29
Ingrese las temperaturas medias del mes: 27
Ingrese las temperaturas medias del mes: 27
Ingrese el nombre del país: Brasil
Ingrese las temperaturas medias del mes: 30
Ingrese las temperaturas medias del mes: 31
Ingrese las temperaturas medias del mes: 32

Listado de paises y sus temperaturas
------------------------------------
Chile Temperaturas: 22 - 23 - 24 -
Argentina Temperaturas: 25 - 25 - 25 -
Uruguay Temperaturas: 29 - 27 - 27 -
Brasil Temperaturas: 30 - 31 - 32 -
Listado pasies y su tempertura media trimestral
-----------------------------------------------
Chile - 23
Argentina - 25
Uruguay - 27
Brasil - 31

El país Brasil tiene una temperatura media mayor: 31
```

Capítulo 94.- Matrices irregulares o dentadas – 1

C# nos permite crear matrices irregulares o dentadas. Se dice que un matriz es irregular si la cantidad de elementos de cada fila varía. Luego podemos imaginar una matriz irregular:



Como podemos ver la fila cero tiene reservado dos espacios, la fila uno reserva cuatro espacios y la última fila reserva espacio para tres componentes.

La sintaxis para declarar una matriz irregular es:

```
int [][] mat;
```

Primero creamos la cantidad de filas dejando vacío el espacio que indica la cantidad de columnas.

```
mat=new int[3][];
```

Luego debemos ir creando cada fila de matriz indicando la cantidad de elementos de la respectiva fila:

```
mat[0]=new int[2];
mat[1]=new int[4];
mat[2]=new int[3];
```

Luego la forma para acceder a sus componentes debe ser utilizando corchetes abiertos y cerrados para cada índice:

```
mat[0][0]=120;
```

Dará un error si queremos cargar el tercer componente de la fila cero (esto es debido a que no existe):

```
mat[0][2]=230;
```

Luego si queremos saber la cantidad de filas que tienen la matriz:

```
Console.Write(mat.Length);
```

Si queremos saber la cantidad de elementos de una determinada fila:

```
Console.Write("Cantidad de elementos de la fila 0:"+mat[0].Length);
Console.Write("Cantidad de elementos de la fila 1:"+mat[1].Length);
Console.Write("Cantidad de elementos de la fila 2:"+mat[2].Length);
```

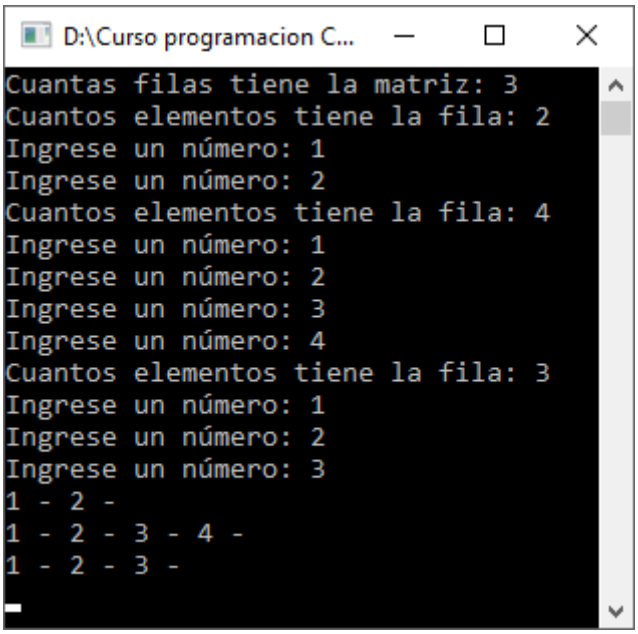
Problema

Confeccionaremos un programa que permita crear una matriz irregular y luego imprimir la matriz completa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto95
{
    2 referencias
    class Matriz
    {

        private int[][] mat;

        1 referencia
        public void cargar()
        {
            int filas, columnas;
            Console.Write("Cuantas filas tiene la matriz: ");
            filas=int.Parse(Console.ReadLine());
            mat = new int[filas][];
            for (int f = 0; f < mat.Length; f++)
            {
                Console.Write("Cuantos elementos tiene la fila: ");
                columnas = int.Parse(Console.ReadLine());
                mat[f] = new int[columnas];
                for (int c=0; c< mat[f].Length; c++)
                {
                    Console.Write("Ingrese un número: ");
                    mat[f][c] = int.Parse(Console.ReadLine());
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f = 0; f < mat.Length; f++)
            {
                for (int c = 0; c < mat[f].Length; c++)
                {
                    Console.Write(mat[f][c] + " - ");

                }
                Console.WriteLine();
            }
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            m.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 95.- Matrices irregulares o dentadas – 2

Problema propuesto

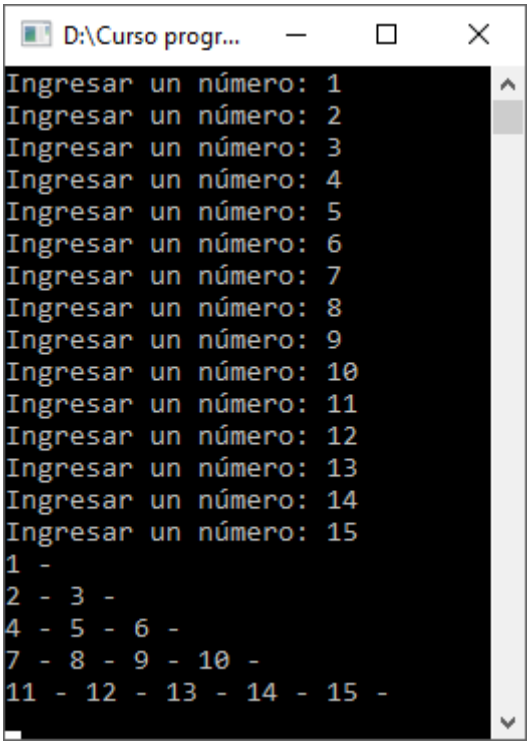
Confeccionar una clase para administrar una matriz irregular de 5 filas y 1 columna en la primera fila, 2 columnas en la segunda fila y así sucesivamente hasta 5 columnas en la última fila (crearla sin intervención del operador).

Realiza la carga por teclado e imprimir posteriormente.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto96
{
    2 referencias
    class Matriz
    {
        private int[][] mat;

        1 referencia
        public void cargar()
        {
            mat = new int[5][];
            for (int c=0; c<5; c++ )
            {
                mat[c] = new int[c+1];
            }
            for (int f=0; f < mat.Length; f++ )
            {
                for (int c=0; c < mat[f].Length; c++)
                {
                    Console.Write("Ingresar un número: ");
                    mat[f][c] = int.Parse(Console.ReadLine());
                }
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f = 0; f < mat.Length; f++)
            {
                for (int c = 0; c < mat[f].Length; c++)
                {
                    Console.Write(mat[f][c] + " - ");
                }
                Console.WriteLine();
            }
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Matriz m = new Matriz();
            m.cargar();
            m.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 96.- Matrices irregulares o dentadas – 3

Problema propuesto

Confeccionar una clase para administrar los días que han faltado los 3 empleados de un empresa.
Definir un vector de 3 elementos de tipo string para cargar los nombres y una matriz irregular para cargar los días que han faltado cada empleado (cargar el número de días que faltó).
Mostrar los empleados con la cantidad de inasistencias.
Que empleado faltó menos días.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto97
{
    2 referencias
    class Control
    {
        private string[] empleados;
        private int[][] dias;
        1 referencia
        public void cargar()
        {
            empleados = new string[3];
            dias = new int[3][];
            int numDias;
            for(int f=0; f < empleados.Length; f++)
            {
                Console.Write("Ingrese el nombre del operario: ");
                empleados[f] = Console.ReadLine();
                Console.Write("Cuantos días ha faltado al trabajo: ");
                numDias = int.Parse(Console.ReadLine());
                dias[f] = new int[numDias];
                for (int c=0; c < numDias; c++)
                {
                    Console.Write("Ingrese el día: ");
                    dias[f][c] = int.Parse(Console.ReadLine());
                }
            }
        }
        1 referencia
        public void imprimir()
        {
            Console.WriteLine("Relación de empleados y días de inasistencia");
            Console.WriteLine("----------------------------------------");
            for (int f=0; f < 3; f++)
            {
                Console.Write(empleados[f] + " ha faltado los días: ");
                for (int c=0;c < dias[f].Length; c++)
                {
                    Console.Write(dias[f][c] + " - ");
                }
                Console.Write(" un total de " + dias[f].Length + " días.");
                Console.WriteLine();
```

```
        }
    }
    1 referencia
    public void menosFaltas()
    {
        int pos = 0;
        int aux = dias[0].Length;
        Console.Write("El empleado que falto menos días es ");
        for (int k=1; k < empleados.Length; k++)
        {
            if (dias[k].Length < aux)
            {
                pos = k;
                aux = dias[k].Length;
            }
        }
        Console.WriteLine(empleados[pos] + " con " + dias[pos].Length + " falta");
    }

    0 referencias
    static void Main(string[] args)
    {
        Control control = new Control();
        control.cargar();
        control.imprimir();
        control.menosFaltas();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:

```
D:\Curso programacion C\Proyecto97\Proyecto97\bin\Debug\Proyecto97...
Ingrese el nombre del operario: Ana
Cuantos días ha faltado al trabajo: 1
Ingrese el día: 10
Ingrese el nombre del operario: Pedro
Cuantos días ha faltado al trabajo: 2
Ingrese el día: 1
Ingrese el día: 2
Ingrese el nombre del operario: Carlos
Cuantos días ha faltado al trabajo: 4
Ingrese el día: 12
Ingrese el día: 13
Ingrese el día: 24
Ingrese el día: 25
Relación de empleados y días de inasistencia
-------------------------------------------
Ana ha faltado los días: 10 -  un total de 1 días.
Pedro ha faltado los días: 1 - 2 -  un total de 2 días.
Carlos ha faltado los días: 12 - 13 - 24 - 25 -  un total de 4 días.
El empleado que falto menos días es Ana con 1 falta
```

Capítulo 97.- Constructor de la clase – 1

En C# podemos definir un método que se ejecuta inicialmente y en forma automática. Este método se llama constructor.

El constructor tiene las siguientes características:

- Tiene el mismo nombre de la clase.
- Es el primer método que se ejecuta.
- Se ejecuta de forma automática.
- No puede retornar datos.
- Se ejecuta una única vez.
- Un constructor tiene por objetivo inicializar atributos.

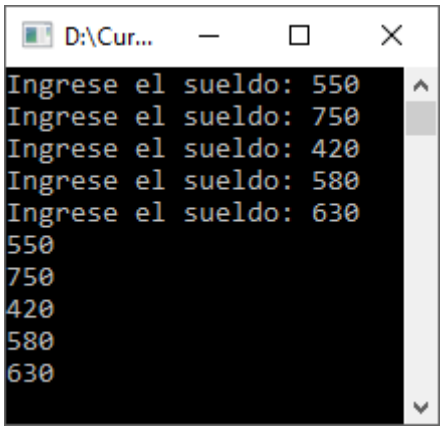
Problema

Se desea guardar los sueldos de 5 operarios en un vector. Realiza la creación y carga del vector en el constructor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto98
{
    3 referencias
    class Operarios
    {
        private int[] sueldos;

        1 referencia
        public Operarios()
        {
            sueldos = new int[5];
            for (int f=0; f < 5; f++)
            {
                Console.Write("Ingrese el sueldo: ");
                sueldos[f] = int.Parse(Console.ReadLine());
            }
        }

        1 referencia
        public void imprimir()
        {
            for (int f=0;f < sueldos.Length; f++)
            {
                Console.WriteLine(sueldos[f]);
            }
            Console.WriteLine();
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Operarios op = new Operarios();
            op.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 98.- Constructor de la clase – 2

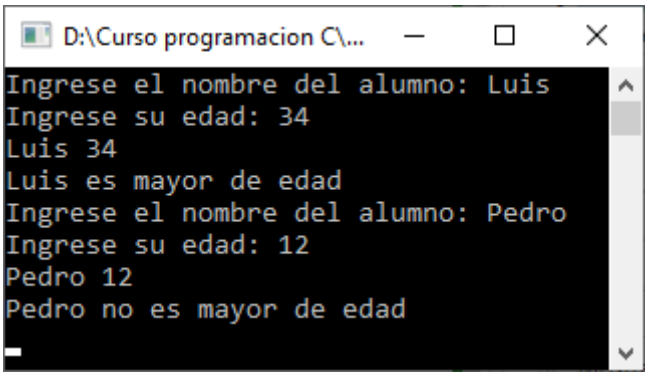
Problema

Plantear una clase llamada Alumno y definir como atributos su nombre y su edad. En el constructor realizar la carga de datos. Definir otros dos métodos para imprimir los datos ingresados y un mensaje si es mayor o no de edad (edad>=18).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto99
{
    5 referencias
    class Alumno
    {
        private string nombre;
        private int edad;

        2 referencias
        public Alumno()
        {
            Console.Write("Ingrese el nombre del alumno: ");
            nombre = Console.ReadLine();
            Console.Write("Ingrese su edad: ");
            edad = int.Parse(Console.ReadLine());
        }

        2 referencias
        public void imprimir()
        {
            Console.WriteLine(nombre + " " + edad);
        }

        2 referencias
        public void mayor()
        {
            if (edad >=18)
            {
                Console.WriteLine(nombre + " es mayor de edad");
            }
            else
            {
                Console.WriteLine(nombre + " no es mayor de edad");
            }
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Alumno alum1 = new Alumno();
            alum1.imprimir();
            alum1.mayor();
            Alumno alum2  = new Alumno();
            alum2.imprimir();
            alum2.mayor();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 99.- Constructor de la clase – 3

Problema propuesto

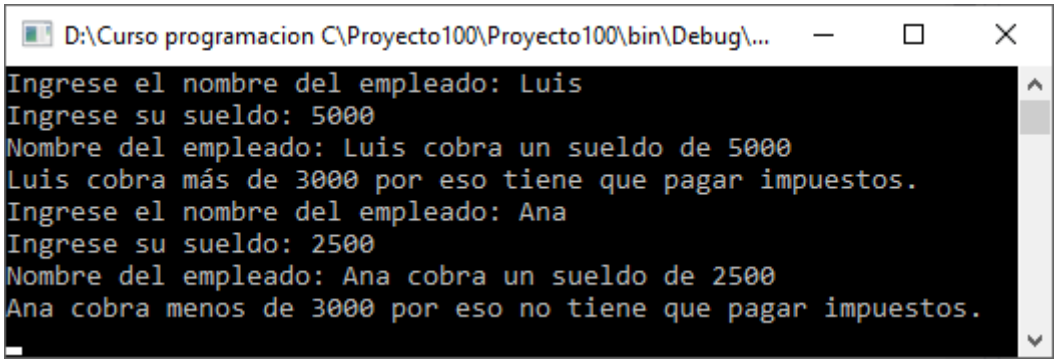
Confeccionar una clase que represente un empleado. Definir como atributos su nombre y sueldo.

En el constructor cargar los atributos y luego en otro método imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto100
{
    5 referencias
    class Empleado
    {
        private string nombre;
        private int sueldo;

        2 referencias
        public Empleado()
        {
            Console.Write("Ingrese el nombre del empleado: ");
            nombre = Console.ReadLine();
            Console.Write("Ingrese su sueldo: ");
            sueldo = int.Parse(Console.ReadLine());
        }

        2 referencias
        public void imprimir()
        {
            Console.WriteLine("Nombre del empleado: "+ nombre
                + " cobra un sueldo de " + sueldo);
        }

        2 referencias
        public void pagar()
        {
            if(sueldo > 3000)
            {
                Console.WriteLine(nombre
                    + " cobra más de 3000 por eso tiene que pagar impuestos.");
            }
            else
            {
                Console.WriteLine(nombre
                    + " cobra menos de 3000 por eso no tiene que pagar impuestos.");
            }
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Empleado empl1 = new Empleado();
            empl1.imprimir();
            empl1.pagar();
            Empleado empl2 = new Empleado();
            empl2.imprimir();
            empl2.pagar();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

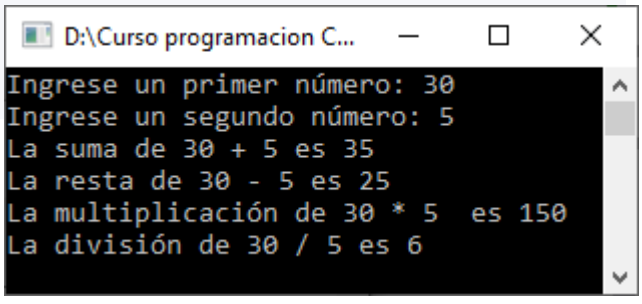


Capítulo 100.- Constructor de la clase – 4

Implementar la clase operaciones. Se deben cargar dos valores enteros en el constructor, calcular su suma resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto101
{
    3 referencias
    class Operaciones
    {
        private int num1, num2;

        1 referencia
        public Operaciones()
        {
            Console.Write("Ingrese un primer número: ");
            num1 = int.Parse(Console.ReadLine());
            Console.Write("Ingrese un segundo número: ");
            num2 = int.Parse(Console.ReadLine());
        }

        1 referencia
        public void suma()
        {
            int suma = num1 + num2;
            Console.WriteLine("La suma de " + num1 + " + " + num2 + " es " + suma);
        }

        1 referencia
        public void resta()
        {
            int resta = num1 - num2;
            Console.WriteLine("La resta de " + num1 + " - " + num2 +" es " + resta);
        }

        1 referencia
        public void multiplicacion()
        {
            int multiplica = num1 * num2;
            Console.WriteLine("La multiplicación de " + num1 + " * " + num2 +"  es " + multiplica);
        }

        1 referencia
        public void division()
        {
            int division = num1 / num2;
            Console.WriteLine("La división de " + num1 + " / " + num2 + " es " + division);
        }

        0 referencias
        static void Main(string[] args)
        {
            Operaciones op = new Operaciones();
            op.suma();
            op.resta();
            op.multiplicacion();
            op.division();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 101.- Colaboración de clases – 1

Normalmente un problema resuelto en la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.

Plantemos un problema separando las actividades en dos clases.

problema

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositado.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco.

Luego debemos definir los atributos y los métodos de cada clase:

```
Cliente
    atributos
        nombre
        monto
    métodos
        constructor
        Depositar
        Extraer
        RetornarMonto

Banco
    atributos
        3 Cliente (3 objetos de la clase Cliente)
    métodos
        constructor
        Operar
        DepositosTotales
```

Creamos un proyecto y dentro del proyecto creamos dos clases llamadas: Cliente y Banco.

Analicemos la implementación del problema.

Los atributos de una clase normalmente son privados para que no se tengan acceso directamente entre otras clases, los atributos son modificados por los métodos de la misma clase:

```
private string nombre;
private int monto;
```

El constructor recibe como parámetro el nombre del cliente y lo almacena en el atributo respectivo e inicializa el atributo monto (importe) en cero.

```
public Cliente(string nom)
{
    nombre = nom;
    monto = 0;
}
```

Los métodos Depositar y Extraer utilizar el atributo monto (importe) con el dinero que llega como parámetro (para simplificar el problema no hemos validado que cuando se extrae dinero el atributo monto quede en valor negativo):

```
public void Depositar(int m)
{
    monto = monto + m;
}

public void Extraer(int m)
{
    monto = monto - m;
}
```

El método RetornarMonto tiene por objetivo comunicar al Banco la cantidad de dinero que tiene el cliente (recordemos que como atributo monto es privado de la clase, debemos tener un método que lo retorne):

```
public int RetornarMonto()
{
    return monto;
}
```

Por último el método imprimir muestra el nombre y el monto de dinero del cliente:

```
public void Imprimir()
{
    Console.WriteLine(nombre+" tiene depositado la suma de "+monto);
}
```

Como podemos observar la clase Cliente no tiene función Main. Entonces donde definimos objetos de la clase Cliente?

La respuesta a esta pregunta es que en la clase Banco definimos tres objetos de la clase Cliente.

Veamos ahora la clase Banco que requiere la colaboración de la clase Cliente.

Primero definimos tres clases de tipo Cliente:

```
class Banco
{
   private Cliente cliente1, cliente2, cliente3;
```

En el constructor creamos los tres objetos (cada vez que creamos un objeto de la clase Cliente debemos pasar a su constructor el nombre del cliente, recordemos que su monto de depósito se inicializa con cero):

```
public Banco()
{
    cliente1=new Cliente("Juan");
    cliente2=new Cliente("Ana");
    cliente3=new Cliente("Pedro");
}
```

El método operar del banco (llamamos a los métodos Depositar y Extraer de los clientes):

```
public void Operar()
{
    cliente1.Depositar(100);
    cliente2.Depositar(150);
    cliente3.Depositar(200);
    cliente3.Extraer(150);
}
```

El método DepositosTotales obtiene el monto depositado de cada uno de los tres clientes, procede a mostrarlos y llama al método imprimir de cada cliente para poder mostrar el nombre y depósito:

```
public void DepositosTotales()
{
    int t = cliente1.RetornarMonto () +
            cliente2.RetornarMonto () +
            cliente3.RetornarMonto ();
    Console.WriteLine ("El total de dinero en el banco es:" + t);
    cliente1.Imprimir();
    cliente2.Imprimir();
    cliente3.Imprimir();
}
```

Por último en la Main definimos un objeto de tipo Banco (la clase Banco es la clase principal en nuestro problema):

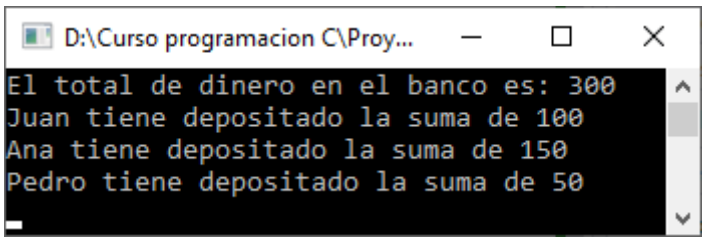
```
static void Main(string[] args)
{
    Banco banco1 = new Banco();
    banco1.Operar();
    banco1.DepositosTotales();
    Console.ReadKey();
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto102
{
    5 referencias
    class Cliente
    {
        private string nombre;
        private int importe;

        3 referencias
        public Cliente(string nom)
        {
            nombre = nom;
            importe = 0;
        }
```

```
        3 referencias
        public void Depositar(int im)
        {
            importe = importe + im;
        }

        1 referencia
        public void extraer(int im)
        {
            importe = importe - im;
        }

        3 referencias
        public int RetornarImporte()
        {
            return importe;
        }

        3 referencias
        public void imprimir()
        {
            Console.WriteLine(nombre + " tiene depositado la suma de " + importe);
        }
    }

    3 referencias
    class Banco
    {
        private Cliente cliente1, cliente2, cliente3;

        1 referencia
        public Banco()
        {
            cliente1 = new Cliente("Juan");
            cliente2 = new Cliente("Ana");
            cliente3 = new Cliente("Pedro");
        }

        1 referencia
        public void Operar()
        {
            cliente1.Depositar(100);
            cliente2.Depositar(150);
            cliente3.Depositar(200);
            cliente3.extraer(150);
        }
        1 referencia
        public void DepositoTotales()
        {
            int t = cliente1.RetornarImporte() +
                    cliente2.RetornarImporte() +
                    cliente3.RetornarImporte();
            Console.WriteLine("El total de dinero en el banco es: " + t);
            cliente1.imprimir();
            cliente2.imprimir();
            cliente3.imprimir();
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            Banco banco1 = new Banco();
            banco1.Operar();
            banco1.DepositoTotales();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 102.- Colaboración de clases – 2

Problema

Plantear un programa que permita jugar a los dados. Las reglas de juego son: Se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "ganó", sino "perdió".

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Dado y la clase JuegoDados.

Luego los atributos y los métodos de cada clase:

```
Dado
    atributos
        valor
    métodos
        constructor
        Tirar
        Imprimir
        RetornarValor

JuegoDeDados
    atributos
        3 Dado (3 objetos de la clase Dado)
    métodos
        constructor
        Jugar
```

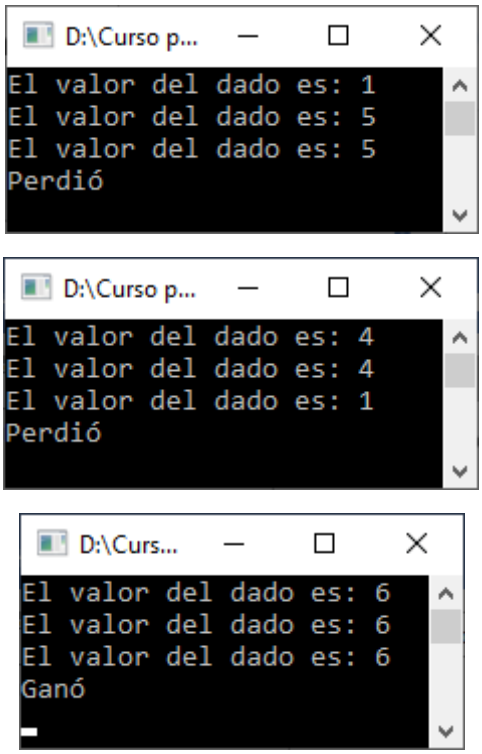
Creamos un proyecto y dentro del proyecto creamos dos clases llamadas:

Dado y JuegoDeDados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto103
{
    5 referencias
    class Dado
    {
        private int valor;
        private static Random aleatorio;

        3 referencias
        public Dado()
        {
            aleatorio = new Random();
        }

        3 referencias
        public void Tirar()
        {
            valor = aleatorio.Next(1,7);
        }
```

```
        3 referencias
        public void imprimir()
        {
            Console.WriteLine("El valor del dado es: " + valor);
        }

        4 referencias
        public int RetornarValor()
        {
            return valor;
        }
    }

    3 referencias
    class JuegoDeDados
    {
        private Dado dado1, dado2, dado3;

        1 referencia
        public JuegoDeDados()
        {
            dado1 = new Dado();
            dado2 = new Dado();
            dado3 = new Dado();
        }


        1 referencia
        public void jugar()
        {
            dado1.Tirar();
            dado2.Tirar();
            dado3.Tirar();
            dado1.imprimir();
            dado2.imprimir();
            dado3.imprimir();
            if (dado1.RetornarValor() == dado2.RetornarValor() &&
                dado1.RetornarValor()==dado3.RetornarValor())
            {
                Console.WriteLine("Ganó");
            }
            else
            {
                Console.WriteLine("Perdió");
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            JuegoDeDados juego = new JuegoDeDados();
            juego.jugar();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 103.- Colaboración de clases – 3

Problema propuesto

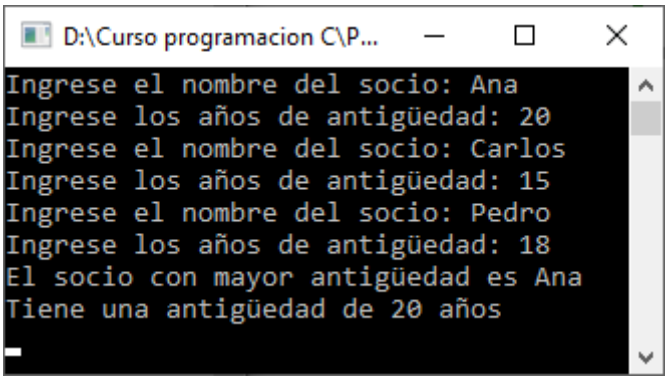
Plantear una clase Club y otra clase Socio.

La clase Socio debe tener los siguientes atributos privados: nombre y la antigüedad en el club (en años). En el constructor pedir la carga del nombre y su antigüedad. La clase Club debe tener como atributos 3 objetos de la clase Socio. Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto104
{
    5 referencias
    class Socio
    {
        private string nombre;
        private int antiguedad;
        3 referencias
        public Socio()
        {
            Console.Write("Ingrese el nombre del socio: ");
            nombre = Console.ReadLine();
            Console.Write("Ingrese los años de antigüedad: ");
            antiguedad = int.Parse(Console.ReadLine());
        }

        9 referencias
        public int anti()
        {
            return antiguedad;
        }

        3 referencias
        public string nom()
        {
            return nombre;
        }
    }

    3 referencias
    class Club
    {
        Socio socio1, socio2, socio3;
```

```
        1 referencia
        public Club()
        {
            socio1 = new Socio();
            socio2 = new Socio();
            socio3 = new Socio();

        }

        1 referencia
        public void mayorAnt()
        {
            if (socio1.anti() > socio2.anti() && socio1.anti()>socio3.anti())
            {
                Console.WriteLine("El socio con mayor antigüedad es " + socio1.nom());
                Console.WriteLine("Tiene una antigüedad de " + socio1.anti() + " años");
            }
            else
            {
                if(socio2.anti()>socio3.anti())
                {
                    Console.WriteLine("El socio con mayor antigüedad es " + socio2.nom());
                    Console.WriteLine("Tiene una antigüedad de " + socio2.anti() + " años");
                }
                else
                {
                    Console.WriteLine("El socio con mayor antigüedad es " + socio3.nom());
                    Console.WriteLine("Tiene una antigüedad de " + socio1.anti() + " años");
                }
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            Club club1 = new Club();
            club1.mayorAnt();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:



Capítulo 104.- Concepto de propiedad – 1

La mayoría de los lenguajes de programación orientado a objetos acceden a sus atributos a través de métodos. Esto lo vimos en el capítulo anterior cuando accedíamos al atributo monto (importe) de un cliente:

```
public void Depositar(int m)
{
    monto = monto + m;
}

public int RetornarMonto()
{
    return monto;
}
```

Vimos que luego llamamos a dichos métodos con la sintaxis:

```
cliente3.Depositar(200);
int m = cliente3.RetornarMonto();
```

En C# normalmente este tipo de problema se lo resuelve implementando una propiedad. Veamos el mismo problema resolviéndolo utilizando propiedades.

Problema

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Propiedades1
{
  class Cliente
  {
    private string nombre;
    private int monto;

    public string Nombre
    {
      set
      {
        nombre = value;
      }
      get
      {
        return nombre;
      }
    }
```

```
    public int Monto
    {
        set
        {
            monto = value;
        }
        get
        {
            return monto;
        }
    }

    public void Imprimir()
    {
        Console.WriteLine(Nombre + " tiene depositado la suma de " + Monto);
    }
}

class Banco
{
    private Cliente cliente1, cliente2, cliente3;

    public Banco()
    {
        cliente1 = new Cliente();
        cliente1.Nombre = "Juan";
        cliente1.Monto = 0;
        cliente2 = new Cliente();
        cliente2.Nombre = "Ana";
        cliente2.Monto = 0;
        cliente3 = new Cliente();
        cliente3.Nombre = "Pedro";
        cliente3.Monto = 0;
    }

    public void Operar()
    {
        cliente1.Monto = cliente1.Monto + 100;
        cliente2.Monto = cliente2.Monto + 150;
        cliente3.Monto = cliente3.Monto + 200;
    }

    public void DepositosTotales()
    {
        int t = cliente1.Monto + cliente2.Monto + cliente3.Monto;
        Console.WriteLine("El total de dinero en el banco es:" + t);
        cliente1.Imprimir();
        cliente2.Imprimir();
```

```
            cliente3.Imprimir();
        }

        static void Main(string[] args)
        {
            Banco banco1 = new Banco();
            banco1.Operar();
            banco1.DepositosTotales();
            Console.ReadKey();
        }
    }
}
```

La propiedad Nombre mediante el modificador set inicializa el atributo nombre con el valor que llega del objeto:

cliente1.Nombre = "Juan";

Como vemos donde definimos el objeto cliente1 accedemos a la propiedad mediante el operador punto y le asignamos un valor (en este caso un string porque la propiedad es de tipo string).
Si queremos consultar el atributo nombre lo podemos hacer mediante la propiedad Nombre. Es común definir el nombre que le damos a la propiedad con el mismo nombre que tiene el atributo pero con el primer carácter en mayúsculas.

```
//atributo en minúsculas
    private int monto;
    //nombre de la propiedad con el mismo nombre pero en mayúsculas.
    public int Monto
    {
        set
        {
            monto = value;
        }
        get
        {
            return monto;
        }
    }
```

Podemos observar que la sintaxis para acceder a las propiedades donde definimos objetos es mucho más intuitiva y sencilla, por ejemplo para saber cuento dinero hay en el banco la sintaxis con propiedades es:

```
int t = cliente1.Monto + cliente2.Monto + cliente3.Monto;
```

Y como lo vimos anteriormente por medio de un método que retorna el monto tenemos la siguiente sintaxis:

```
int t = cliente1.RetornarMonto () +
        cliente2.RetornarMonto () +
        cliente3.RetornarMonto ();
```

Lo primero que nos viene a la mente es porque no definir los atributos con el modificador public:

```
public int monto;
```

Para luego poder consultarlos y/o modificarlos con la sintaxis:

```
int t = cliente1.monto + cliente2.monto + cliente3.monto;
```

Ahora veamos que cuando consultamos o inicializamos una propiedad en realidad lo que está sucediendo es la ejecución de un método (set o get) donde podemos disponer código donde validar el valor asignado. Por ejemplo si disponemos la restricción que el Monto siempre debe ser positivo para que se almacene, luego debemos codificar la propiedad con la siguiente sintaxis:

```
public int Monto
    {
        set
        {
                if (value >= 0)
                {
                        monto = value;
                 }
                 else
                 {
                        Console.WriteLine("No se puede tener un monto negativo.");
                }
        }
        get
        {
                return monto;
         }
    }
```

Es decir si el valor que le asignamos a la propiedad Monto es negativo luego no se inicializa el atributo monto con dicho valor.

Si ejecutamos este código luego debe mostrar un mensaje indicando que "No puede tener monto negativo":

```
cliente1.Monto = -100;
```

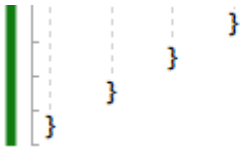
Capítulo 105.- Concepto de propiedad – 2

Problema

Plantear un programa que permita jugar a los dados. Las reglas de juego son: se tiene tres dados si los tres salen con el mismo valor mostrar un mensaje que "gano", sino "perdió".

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto106
{
    namespace Proyecto105
    {
        5 referencias
        class Dado
        {
            private int valor;
            private static Random aleatorio;

            5 referencias
            public int Valor
            {
                private set
                {
                    valor = value;
                }
                get
                {
                    return valor;
                }
            }

            3 referencias
            public Dado()
            {
                aleatorio = new Random();
            }

            3 referencias
            public void Tirar()
            {
                Valor = aleatorio.Next(1, 7);
            }
```

```
        3 referencias
        public void imprimir()
        {
            Console.WriteLine("El valor del dado es: " + valor);
        }

        0 referencias
        public int RetornarValor()
        {
            return valor;
        }
    }

    3 referencias
    class JuegoDeDados
    {
        private Dado dado1, dado2, dado3;

        1 referencia
        public JuegoDeDados()
        {
            dado1 = new Dado();
            dado2 = new Dado();
            dado3 = new Dado();
        }

        1 referencia
        public void jugar()
        {
            dado1.Tirar();
            dado2.Tirar();
            dado3.Tirar();
            dado1.imprimir();
            dado2.imprimir();
            dado3.imprimir();
            if (dado1.Valor == dado2.Valor &&
                dado1.Valor == dado3.Valor)
            {
                Console.WriteLine("Ganó");
            }
            else
            {
                Console.WriteLine("Perdió");
            }
        }

        0 referencias
        static void Main(string[] args)
        {
            JuegoDeDados juego = new JuegoDeDados();
            juego.jugar();
            Console.ReadKey();
```



Capítulo 106.- Concepto de propiedad – 3

Problema propuesto

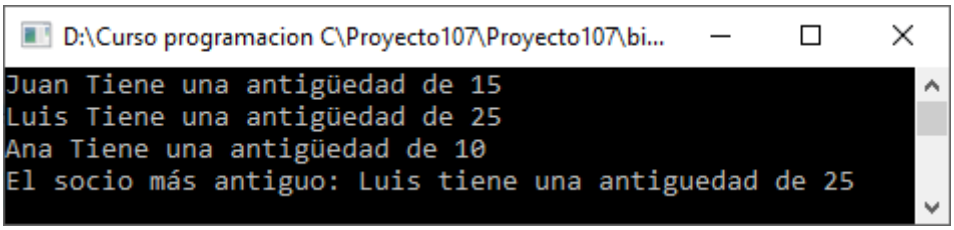
Plantear una clase Club y otra clase Socio.

La clase Socio debe tener los siguientes atributos privados: nombre y la antigüedad en el club (en años). Definir dos propiedades para poder acceder al nombre y a la antigüedad del socio (no permitir cargar un valor negativo en la antigüedad). La clase Club debe tener como atributo 3 objetos de la clase Socio. Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto107
{
    5 referencias
    class Socio
    {
        private string nombre;
        private int antiguedad;

        9 referencias
        public string Nombre
        {
            set
            {
                nombre = value;
            }
            get
            {
                return nombre;
            }
        }
        15 referencias
        public int Antiguedad
        {
            set
            {
                if (value >= 0)
                {
                    antiguedad = value;
                }
                else
                {
                    Console.WriteLine("La antigüedad no puede ser negativa");
                }
            }
```

```
            get
            {
                return antiguedad;
            }
        }
    }
3 referencias
class Club
{
    Socio socio1, socio2, socio3;

    1 referencia
    public Club()
    {
        socio1 = new Socio();
        socio2 = new Socio();
        socio3 = new Socio();

    }

    1 referencia
    public void imprimir()
    {
        Console.WriteLine(socio1.Nombre + " Tiene una antigüedad de " + socio1.Antiguedad);
        Console.WriteLine(socio2.Nombre + " Tiene una antigüedad de " + socio2.Antiguedad);
        Console.WriteLine(socio3.Nombre + " Tiene una antigüedad de " + socio3.Antiguedad);
    }

    1 referencia
    public void MasAnt()
    {
        if (socio1.Antiguedad > socio2.Antiguedad && socio1.Antiguedad>socio3.Antiguedad)
        {
            Console.WriteLine("El socio más antiguo: " + socio1.Nombre
                + " tiene una antiguedad de " + socio1.Antiguedad);
        }
        else
        {
            if (socio2.Antiguedad> socio3.Antiguedad)
            {
                Console.WriteLine("El socio más antiguo: " + socio2.Nombre
               + " tiene una antiguedad de " + socio2.Antiguedad);
            }
            else
            {
                Console.WriteLine("El socio más antiguo: " + socio3.Nombre
               + " tiene una antiguedad de " + socio3.Antiguedad);
            }
        }
    }

    0 referencias
    static void Main(string[] args)
    {
        Club club1 = new Club();
        club1.socio1.Nombre = "Juan";
        club1.socio1.Antiguedad = 15;
        club1.socio2.Nombre = "Luis";
        club1.socio2.Antiguedad = 25;
        club1.socio3.Nombre = "Ana";
        club1.socio3.Antiguedad = 10;
        club1.imprimir();
        club1.MasAnt();
        Console.ReadKey();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 107.- Herencia – 1

Vimos en el capítulo anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relación entre clases que es la Herencia.

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todos los atributos, propiedades y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir atributos, propiedades y métodos propios.

clase padre

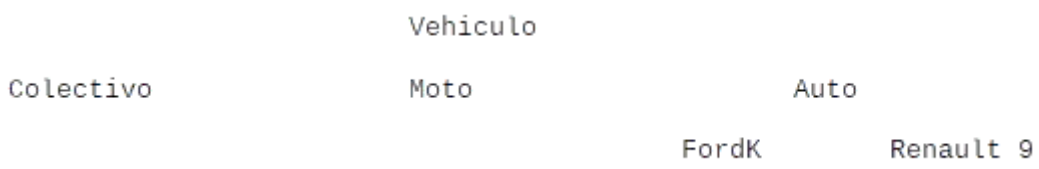
Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos, propiedades y métodos de la clase padre.

subclase

Clase descendiente de otra. Hereda automáticamente los atributos, propiedades y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

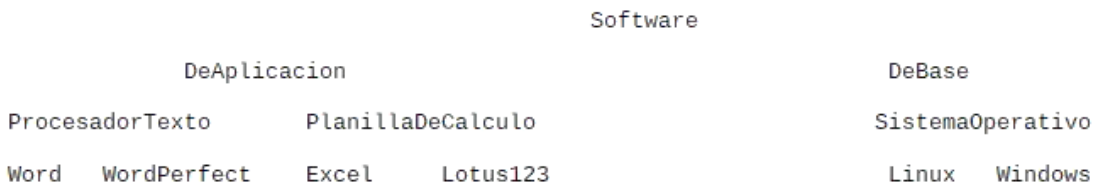
Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos la clase vehículo. Qué clase podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (la subclase añaden nuevos atributos, propiedades y métodos).

2) Imaginemos la clase Software. Qué clases podrían derivar de ella?



El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que en cuando una clase contiene un objeto de otra clase como atributo.

Cuando la relación entre dos clases es de tipo "…tiene un…" o "…es parte de…", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existen una relación de tipo "…tiene un…", no debe implementarse herencia sino declarar en la ClaseA un atributo de la ClaseB.

Por ejemplo: ternemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "…tiene 4…" Ruedas, Volante "…es parte de…" Auto; pero la clase Auto no debe derivar de Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración.

Debemos declarar en la clase Auto 4 atributos de topo Rueda y 1 de tipo Volante.

Luego si vemos que dos clases responder a la pregunta ClaseA "..es un.." ClaseB es posible que haya una relación de herencia.

Por ejemplo:

```
Auto "es un" Vehiculo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operacion
```

Problema

Ahora planteamos el primer problema utilizando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1 y valor2 y resultado. Las propiedades a definir son Valor1, Valor2 y Resultado, el método Operar (que en caso de la clase "Suma" suma los dos valores y en caso de la clase "Resta" hace la diferencia entre Valor1 y Valor2.

Si analizamos ambas clases encontramos que muchas propiedades son idénticas. En estos casos es bueno definir una clase padre que agrupe dichas propiedades, atributos y responsabilidades comunes.

La relación de herencia que podemos disponer para este problema es:

```
                    Operacion

Suma                                    Resta
```

Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Opercion), luego las propiedades Valor1, Valor2 son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operación. Lo mismo las propiedades Valor1, Valor2 y Resultado se definirán en la clase padre Operación.

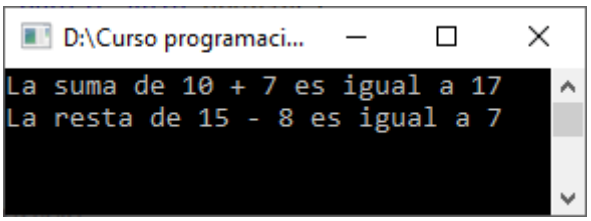
Crear un proyecto y luego crear cuatro clases llamadas: Operación, Suma, Resta y Prueba.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto108
{
```

```
2 referencias
class Operacion
{
    protected int valor1;
    6 referencias
    public int Valor1
    {
        set
        {
            valor1 = value;
        }
        get
        {
            return valor1;
        }
    }
    protected int valor2;
    6 referencias
    public int Valor2
    {
        set
        {
            valor2 = value;
        }
        get
        {
            return valor2;
        }
    }
    protected int resultado;

    4 referencias
    public int Resultado
    {
        set
        {
            resultado = value;
        }
        get
        {
            return resultado;
        }
    }
}
2 referencias
class Suma: Operacion
{
    1 referencia
    public void operar()
    {
        Resultado = Valor1 + Valor2;
    }
}
```

```
2 referencias
class Resta : Operacion
{
    1 referencia
    public void operar()
    {
        Resultado = Valor1 - Valor2;
    }
}

0 referencias
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        Suma suma1 = new Suma();
        suma1.Valor1 = 10;
        suma1.Valor2 = 7;
        suma1.operar();
        Console.WriteLine("La suma de " + suma1.Valor1 + " + "
            + suma1.Valor2 + " es igual a " + suma1.Resultado);
        Resta resta1 = new Resta();
        resta1.Valor1 = 15;
        resta1.Valor2 = 8;
        resta1.operar();
        Console.WriteLine("La resta de " + resta1.Valor1 + " - "
            + resta1.Valor2 + " es igual a " + resta1.Resultado);
        Console.ReadLine();
    }
}
}
```

Si ejecutamos este será el resultado:



Capítulo 108.- Herencia – 2

Problema

Confeccionar una clase Persona que tenga como atributos el nombre y la edad (definir las propiedades para poder acceder a dichos atributos). Definir como responsabilidad un método para imprimir.

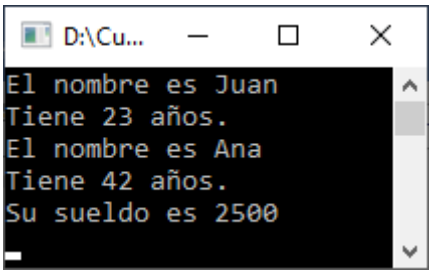
Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo (y su propiedad) y el método para imprimir su sueldo.

Definir un objeto de la clase Persona y llamar a sus métodos y propiedades. También crear un objeto de la clase Empleado y llamar a sus métodos y propiedades.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto109
{
    3 referencias
    class Persona
    {
        protected string nombre;

        3 referencias
        public string Nombre
        {
            set
            {
                nombre = value;
            }
            get
            {
                return nombre;
            }
        }

        protected int edad;

        3 referencias
        public int Edad
        {
            set
            {
                edad = value;
            }
            get
            {
                return edad;
            }
        }
```

```
        2 referencias
        public void imprimir()
        {
            Console.WriteLine("El nombre es " + Nombre);
            Console.WriteLine("Tiene " + Edad + " años.");
        }

    }
    2 referencias
    class Empleado: Persona
    {
        private int sueldo;
        2 referencias
        public int Sueldo
        {
            set
            {
                sueldo = value;
            }
            get
            {
                return sueldo;
            }
        }

        1 referencia
        new public void imprimir()
        {
            base.imprimir();
            Console.WriteLine("Su sueldo es " +  Sueldo);
        }


        0 referencias
        static void Main(string[] args)
        {
            Persona persona1 = new Persona();
            persona1.Nombre = "Juan";
            persona1.Edad = 23;
            persona1.imprimir();

            Empleado empleado1 = new Empleado();
            empleado1.Nombre = "Ana";
            empleado1.Edad = 42;
            empleado1.Sueldo = 2500;
            empleado1.imprimir();
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:


D:\Cu...
El nombre es Juan
Tiene 23 años.
El nombre es Ana
Tiene 42 años.
Su sueldo es 2500

Capítulo 109.- Orden de ejecución de los constructores con herencias

Cuando tenemos constructores en las clases y subclases el orden de ejecución de los mismos es:

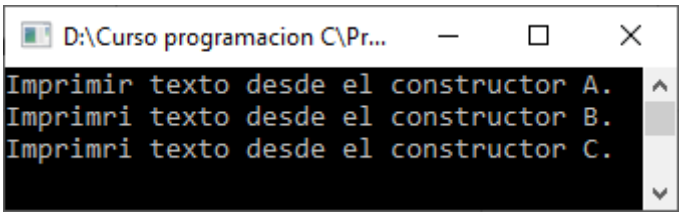
Primero se ejecuta el constructor de la clase Padre.
Segundo se ejecuta el constructor de la subclase.

Problema

Plantear tres clases A, B, C que B hereda de A y C hereda de B. definir un constructor a cada clase que muestre un mensaje. Luego definir un objeto de la clase C.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto110
{
    2 referencias
    class A
    {
        0 referencias
        public A()
        {
            Console.WriteLine("Imprimir texto desde el constructor A.");
        }
    }

    2 referencias
    class B: A
    {
        0 referencias
        public B()
        {
            Console.WriteLine("Imprimri texto desde el constructor B.");
        }
    }
    3 referencias
    class C: B
    {
        1 referencia
        public C()
        {
            Console.WriteLine("Imprimri texto desde el constructor C.");
        }
        0 referencias
        static void Main(string[] args)
        {
            C c1 = new C();
            Console.ReadLine();
        }
    }
}
```

Si ejecutamos este será el resultado:



Problema

Plantear tres clases A, B, C que B herede de A y C de B. Definir un constructor a cada clase que reciba como parámetro un entero. Luego definir un objeto de la clase C.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto111
{
    3 referencias
    class A
    {
        protected int numA;

        1 referencia
        public A(int a)
        {
            Console.WriteLine(a);
        }
    }

    3 referencias
    class B: A
    {
        protected int numB;

        1 referencia
        public B(int b): base(b/2)
        {
            Console.WriteLine(b);
        }

    }
    3 referencias
    class C: B
    {
        protected int numC;
        1 referencia
        public C(int c): base(c/2)
        {
            Console.WriteLine(c);
        }
```

```
        0 referencias
        static void Main(string[] args)
        {
            C c1 = new C(20);
            Console.ReadKey();
        }
    }
}
```

Si ejecutamos este será el resultado:

Capítulo 110.- Clase parcial (partial class)

Hasta ahora hemos visto que una clase se la implementa en forma completa dentro de un archivo. El lenguaje C# permite la implementación de una clase en dos o más archivos. Para esto hay que agregarle el modificador partial cuando declaramos la clase.

Este concepto es ampliamente utilizado por el entorno de Visual Studio.Net en la generación de interfaces visuales.

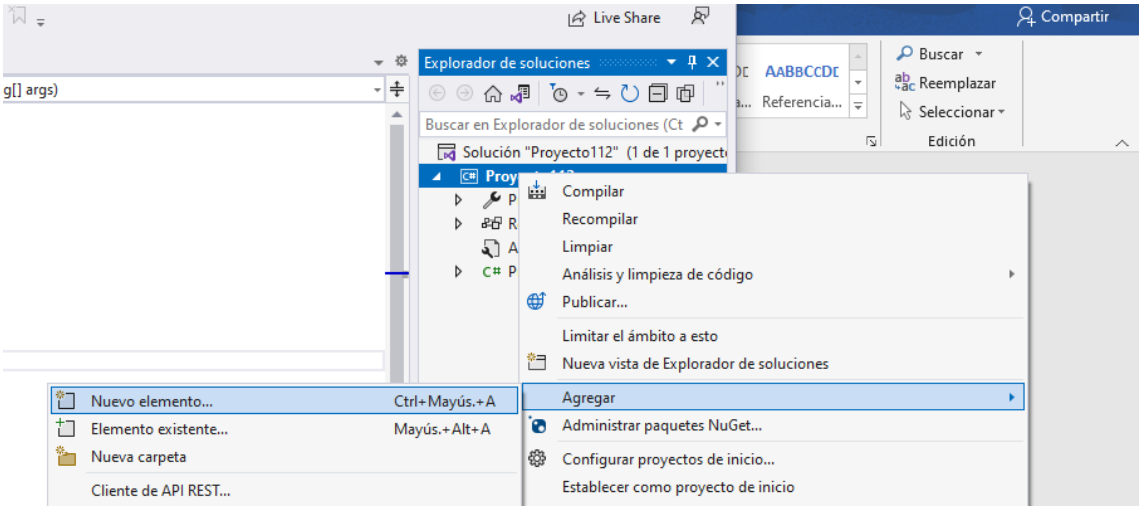
Como veremos en capítulos futuros es necesario presentar "partial class" para su entendimiento.

Una clase parcial no es más ni menos que crear una clase completa y luego agrupar métodos y propiedades den dos o más archivos.

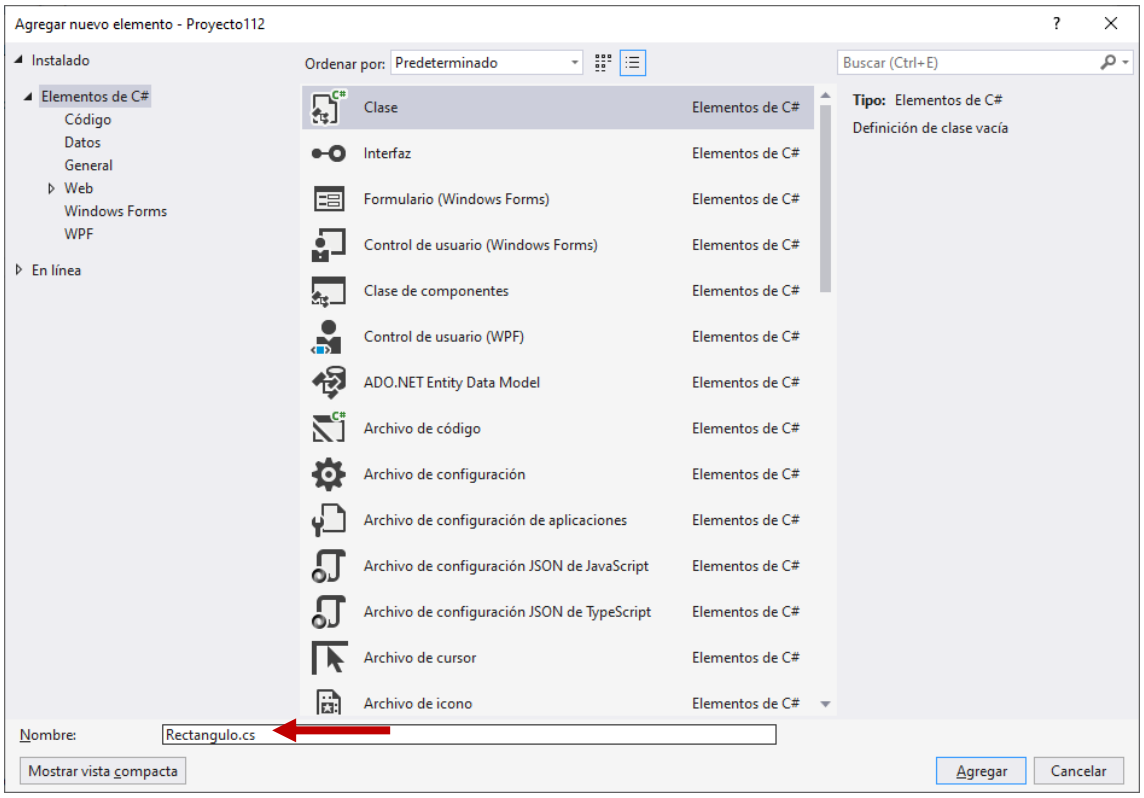
Problema

Plantear un clase Rectángulo, definir dos propiedades: Lado1 y Lado2. Definir dos métodos RetornarSuperficie y RetornarPerimetro. Dividir la clase en dos archivos utilizando el concepto de "partial class".

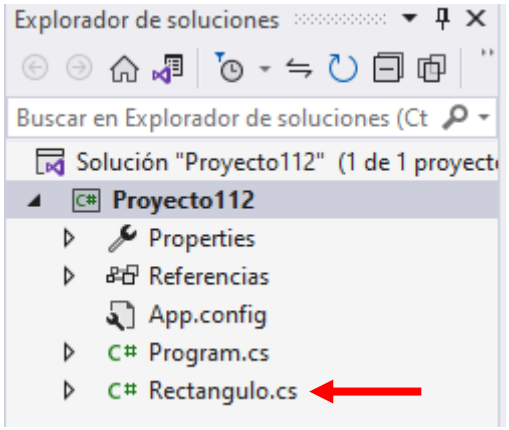
Como vamos a trabajar con varios archivos:



En el apartado de Explorador de soluciones botón derecho sobre el nombre del proyecto del menú seleccionaremos agregar y de este nuevo elemento.



Seleccionamos Clase y como nombre Rectangulo.cs, seguido del botón agregar.

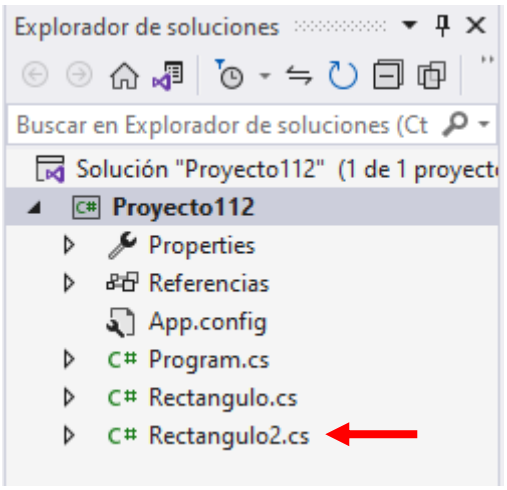



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto112
{
    0 referencias
    partial class Rectangulo
    {

    }
}
```

Le agregamos el parámetro partial.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto112
{
    0 referencias
    partial class Rectangulo
    {
        private int lado1;

        0 referencias
        public int Lado1
        {
            set
            {
                lado1 = value;
            }
            get
            {
                return lado1;
            }
        }

        private int lado2;

        0 referencias
        public int Lado2
        {
            set
            {
                lado2 = value;
            }
            get
            {
                return lado2;
            }
        }
    }
}
```

Ya hemos realizado una parte de la clase, repitiendo los pasos anteriores vamos a agregar un nuevo archivo llamada Rectangulo2.




```
using System;
using System.Collections.Generi
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto112
{
    1 referencia
    partial class Rectangulo
    {
    }
}
```

Ponemos el mismo nombre que pusimos en la clase anterior con el parámetro partial.

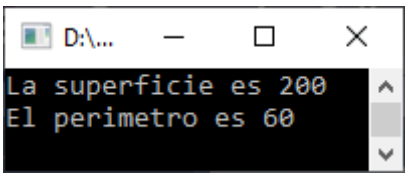
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto112
{
    1 referencia
    partial class Rectangulo
    {
        0 referencias
        public int RetornarSuperficie()
        {
            int sup = Lado1 * Lado2;
            return sup;
        }

        0 referencias
        public int RetornarPerimetro()
        {
            int per = (Lado1*2) + (Lado2*2);
            return per;
```

```
        }
    }
}
```

Ahora nos vamos al archivo donde está el método main.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proyecto112
{
    0 referencias
    internal class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            Rectangulo rect1 = new Rectangulo();
            rect1.Lado1 = 10;
            rect1.Lado2 = 20;
            Console.WriteLine("La superficie es " + rect1.RetornarSuperficie());
            Console.WriteLine("El perimetro es " + rect1.RetornarPerimetro());
            Console.ReadKey();
        }
    }
}
```

Cuando ejecutemos este será el resultado:

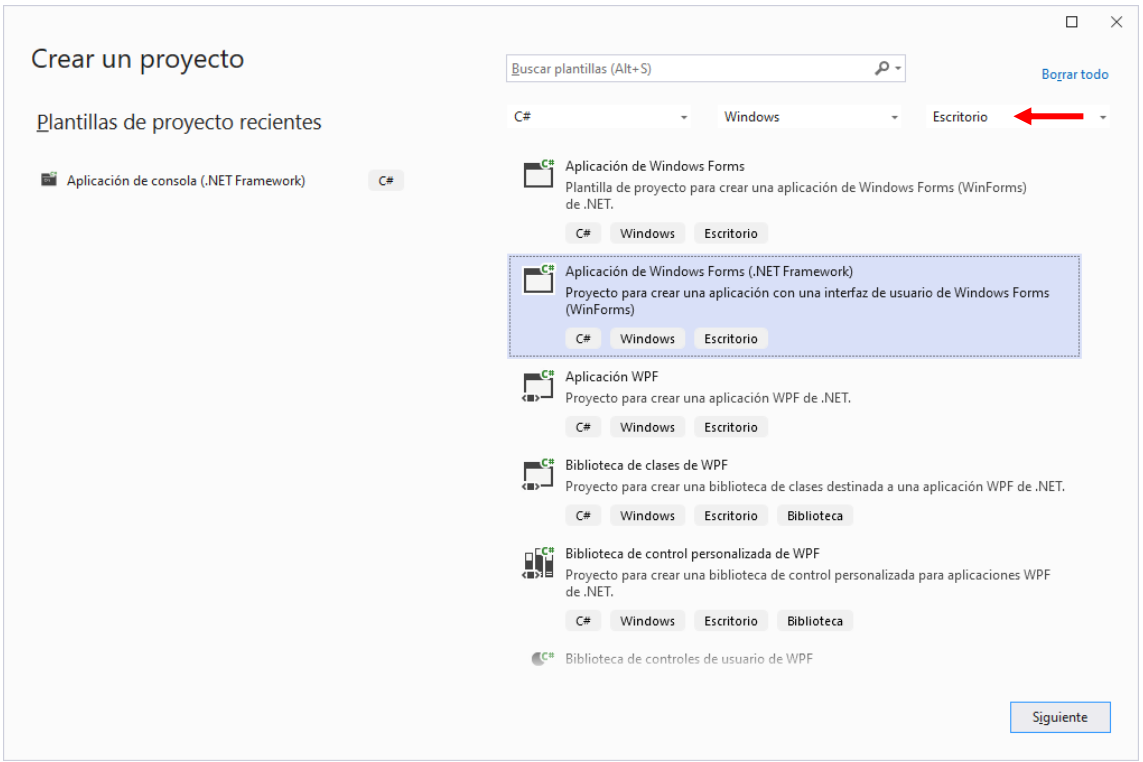


Capítulo 111.- interfaces visuales (Windows Forms)

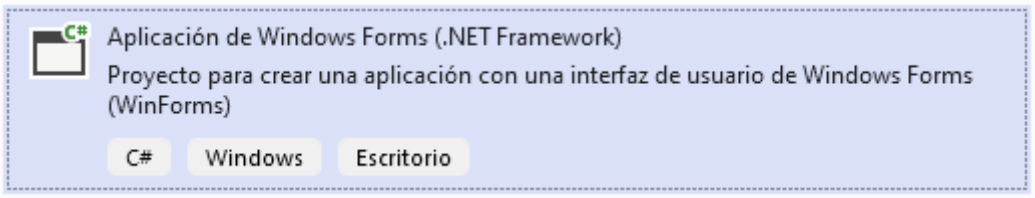
En C# existen varias librerías de clase para implementar interfaces visuales. Utilizaremos la librería Windows Forms.

Para crear una aplicación que utilice esta librería debemos crear un proyecto. Los pasos son los siguientes:

1. Desde el menú de opciones de Visual Studio.Net seleccionamos la opción Archivo -> Nuevo -> Proyecto…
2. Seleccionamos la plantilla "Aplicación de Windows Forms (.Net Framework)".

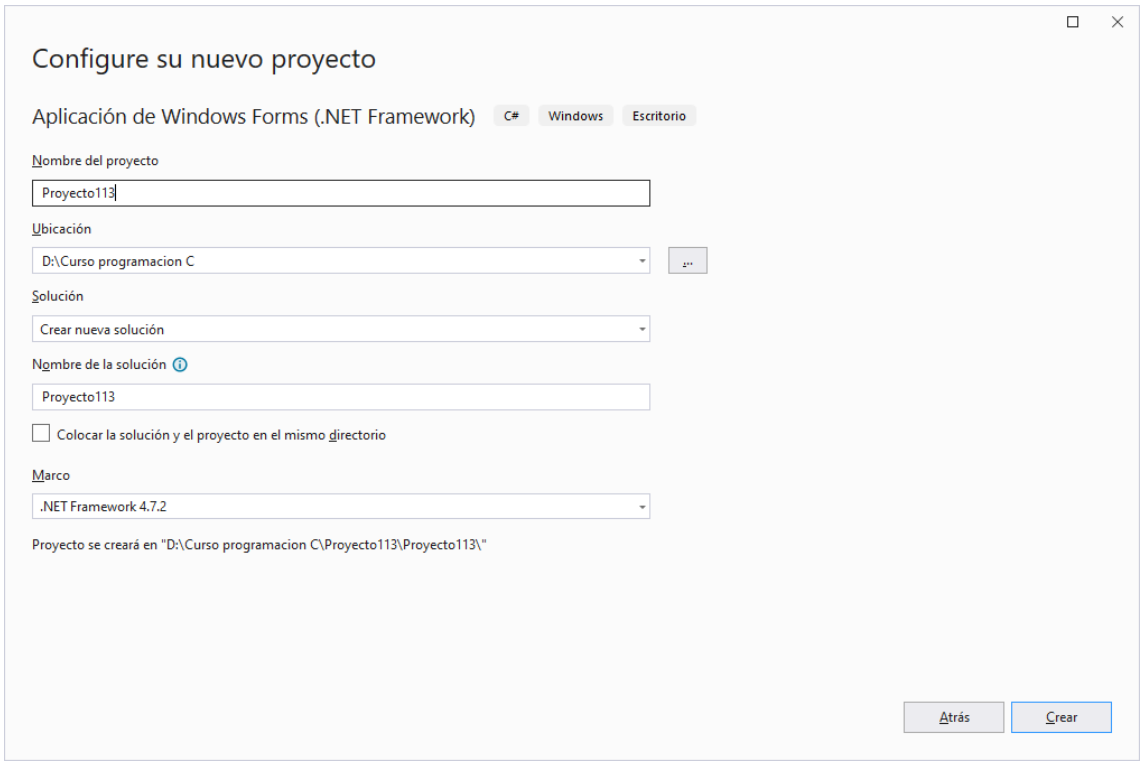


En la parte superior derecha seleccionamos Escritorio.

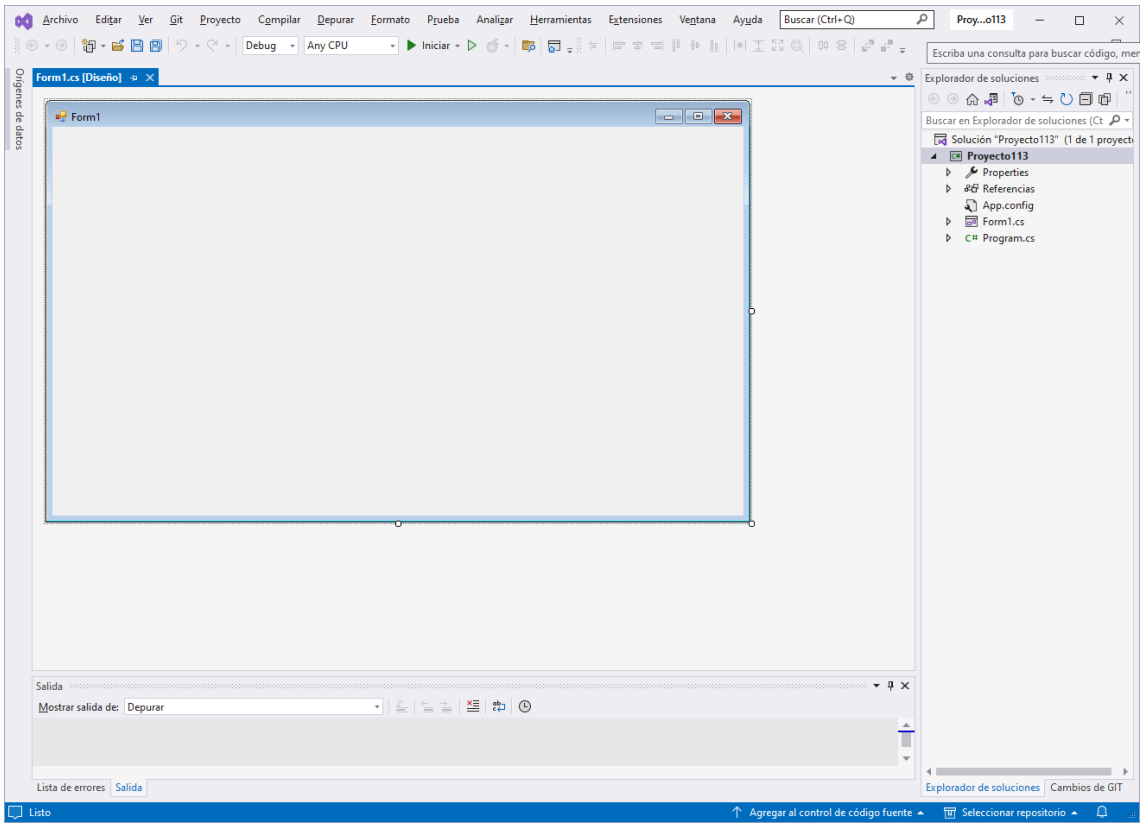


A continuación seleccionaremos "Aplicaciones de Windows Forms (.NET Framewort).

A continuación de damos al botón siguiente.

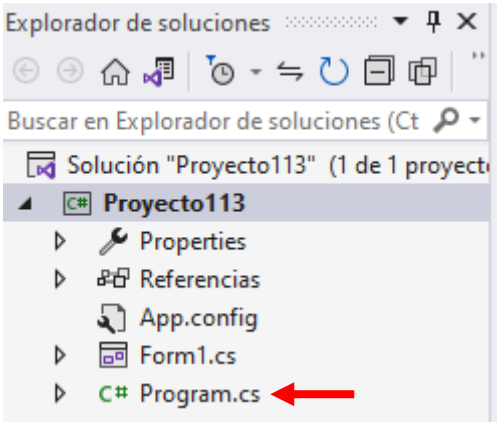


Asignamos un nombre a nuestro proyecto, seguido del botón Crear.



Ya podemos trabajar con interfaces visuales.

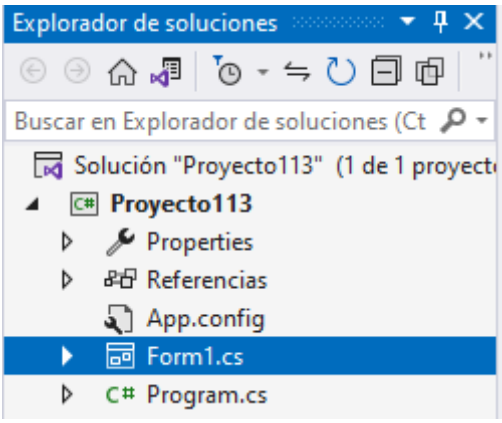
Ya nos ha generado automáticamente una ventana.



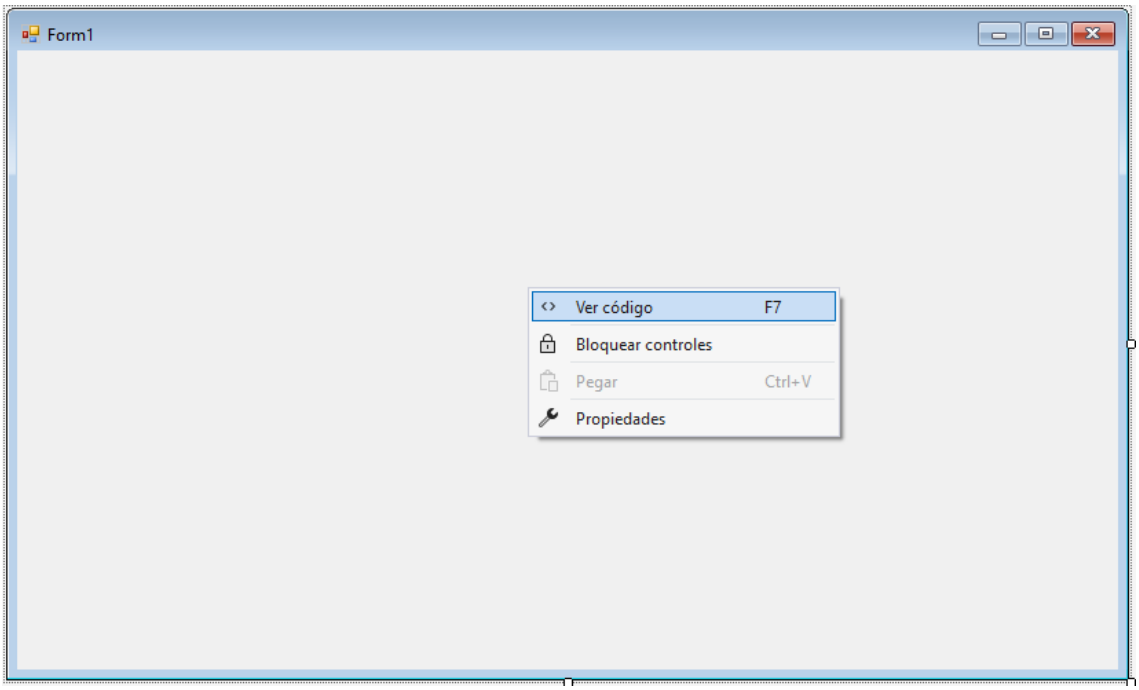
Desde el explorador de soluciones podemos acceder al código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto113
{
    0 referencias
    internal static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        0 referencias
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Podemos observar que en el método main ya le ha agregado unas líneas de código.

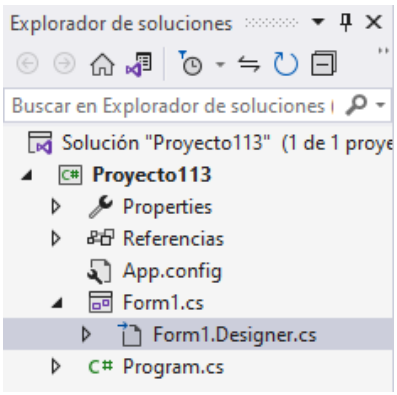


Para regresar al entorno gráfico seleccionaremos con doble clic Form1.cs.



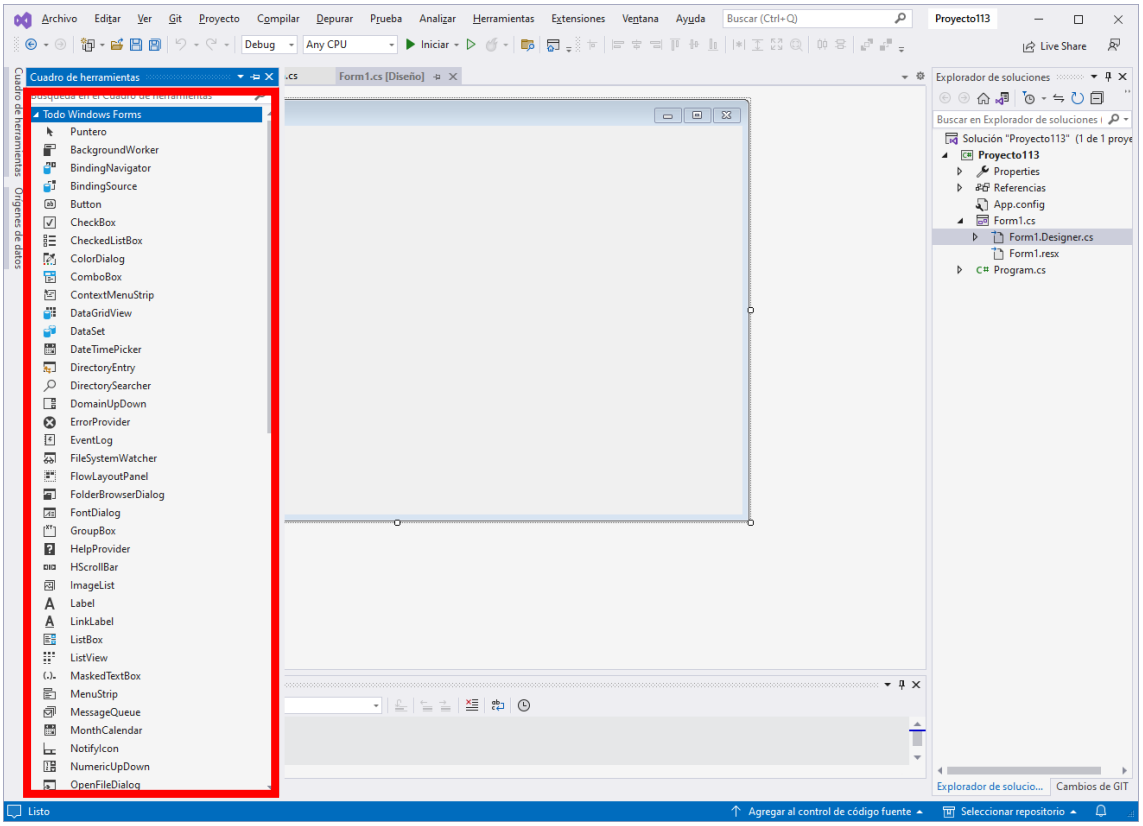
Si seleccionamos con el botón derecho el formulario tendremos la opción de Ver código.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto113
{
    3 referencias
    public partial class Form1 : Form
    {
        1 referencia
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```



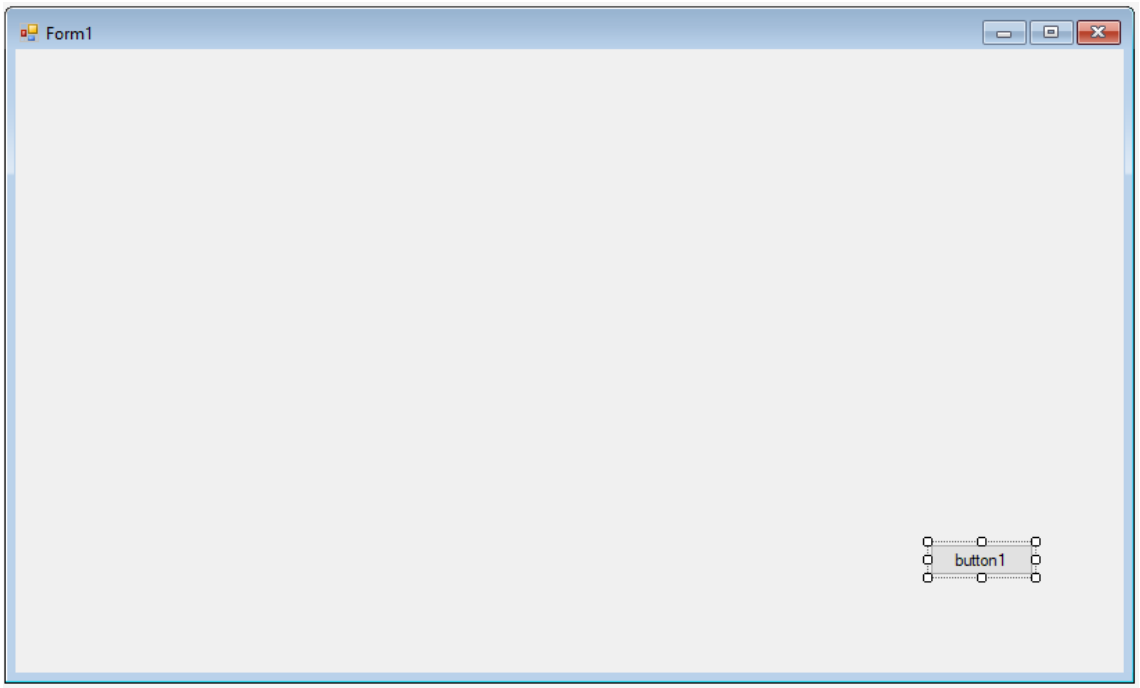
También se ha generado el siguiente archivo "Form1.Designer.cs".

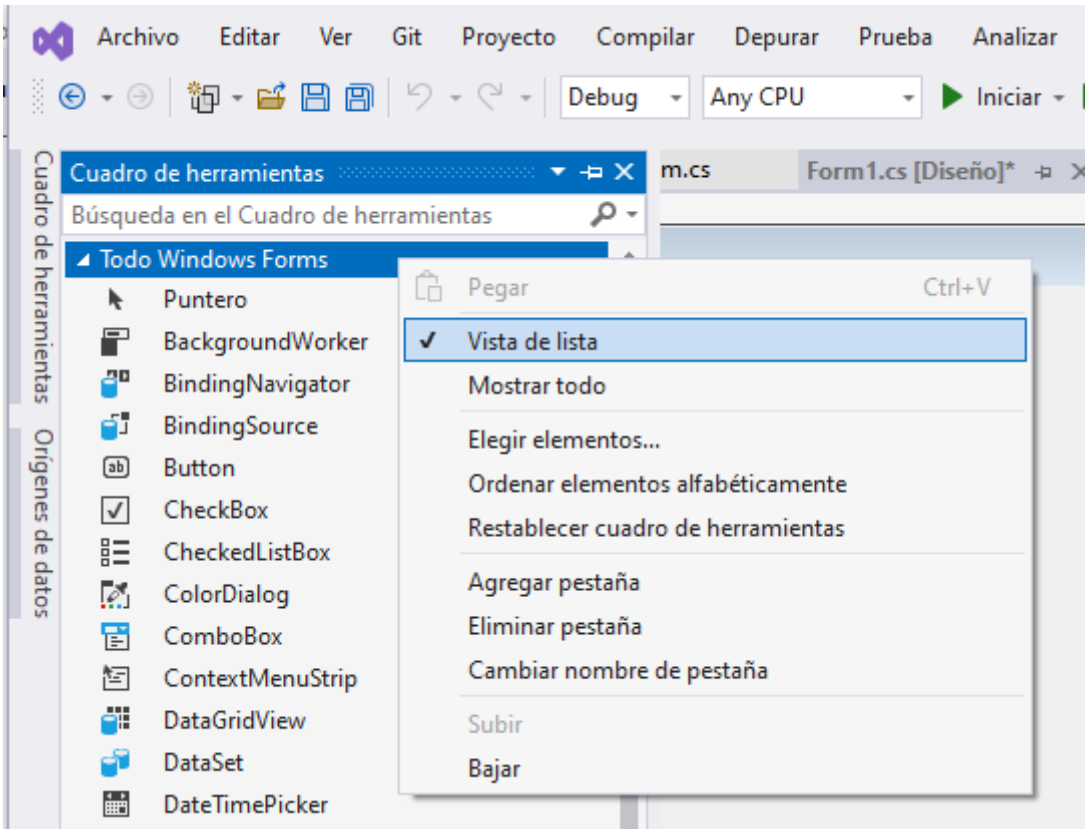
```
namespace Proyecto113
{
    3 referencias
    partial class Form1
    {
        /// <summary>
        /// Variable del diseñador necesaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se estén usando.
        /// </summary>
        /// <param name="disposing">true si los recursos administrados se deben desechar; false en caso contrario.</param>
        0 referencias
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Código generado por el Diseñador de Windows Forms
    }
}
```



En la parte izquierda tenemos todos los componentes visuales que podemos agregar en nuestro formulario, si esta no esta visible del menú ver seleccionaremos cuadro de herramientas.

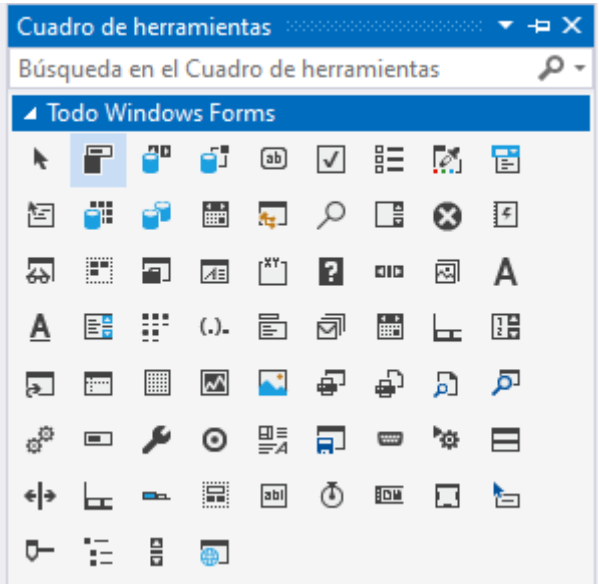
Para agregar uno solamente lo tenemos que seleccionar y arrastrarlo al formulario.





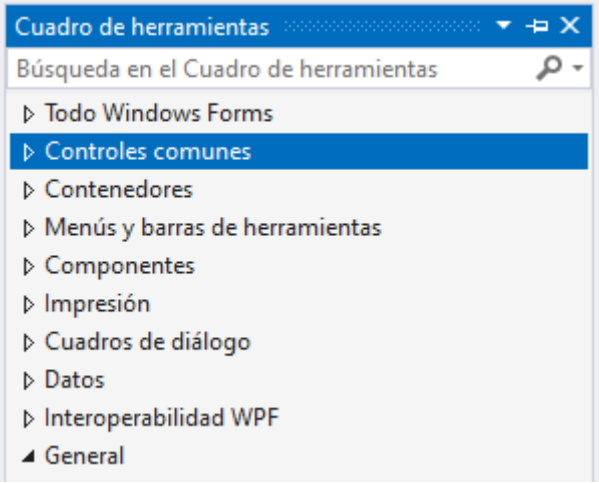
Si seleccionamos con el botón derecho donde pone Todo Windows Form y desmarcamos Vista de lista.

Los veremos de la siguiente forma:

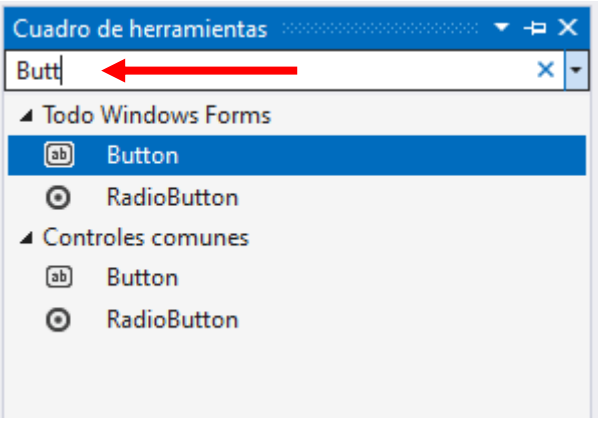


En principio trabajaremos en vista de lista para asociar la imagen con el texto.

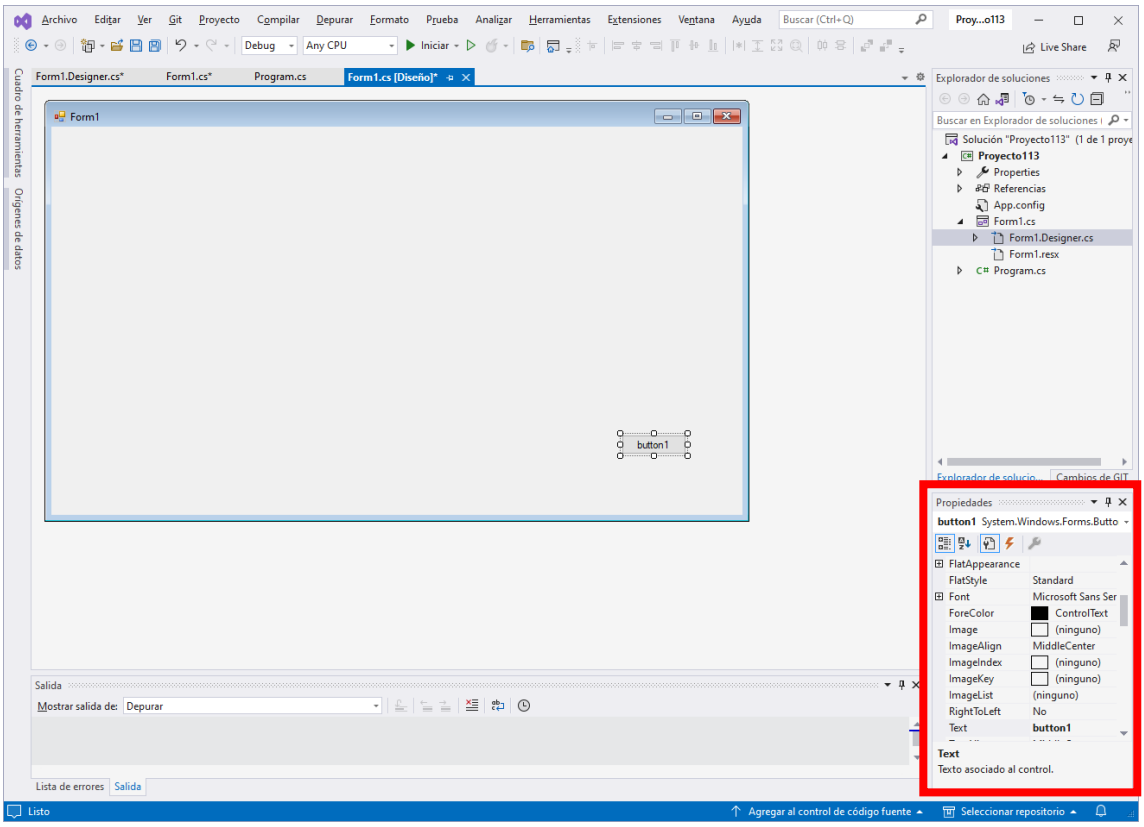
Estos controles se encuentran agrupados para su mejor localización.



También podemos realizar una búsqueda:

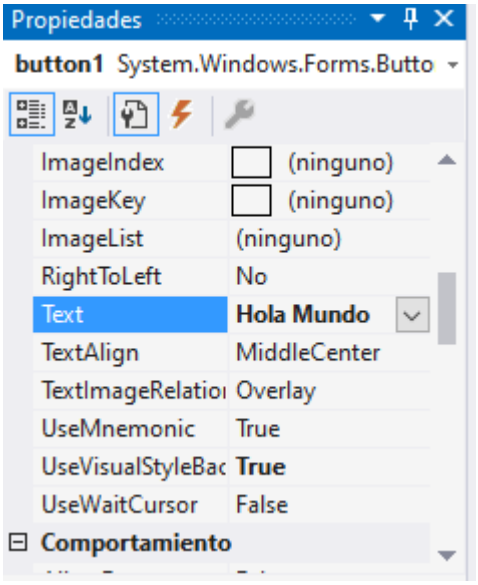


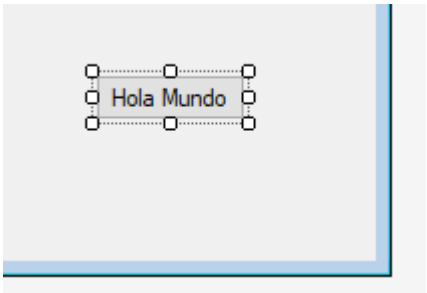
En la parte inferior derecha tenemos la ventana de propiedades.



Si no la tienes visible la podrás activar desde el menú ver y de este Ventana de propiedades o pulsar la tecla de función F4.

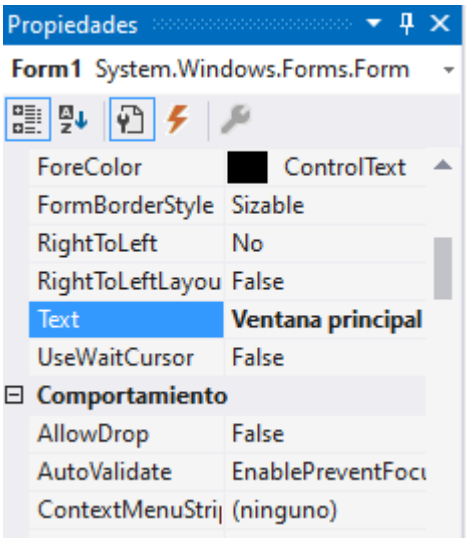
Teniendo seleccionado el objeto lo podemos cambiar las propiedades, ahora que tenemos seleccionado el botón vamos a cambiarle el texto que tiene.

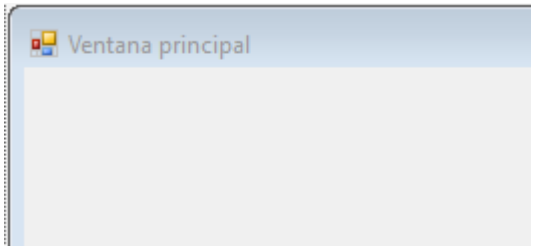




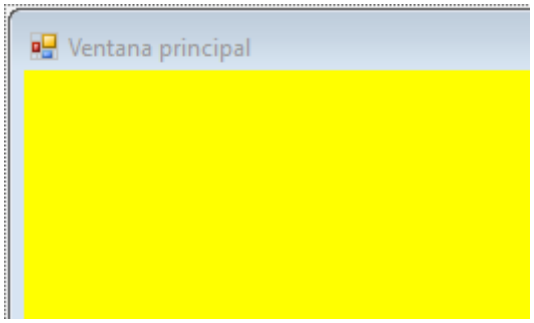
Podemos cambiar la fuente, el estilo, tamaño, color, etc.

Vamos a seleccionar el formulario para cambiar el título de la ventana.

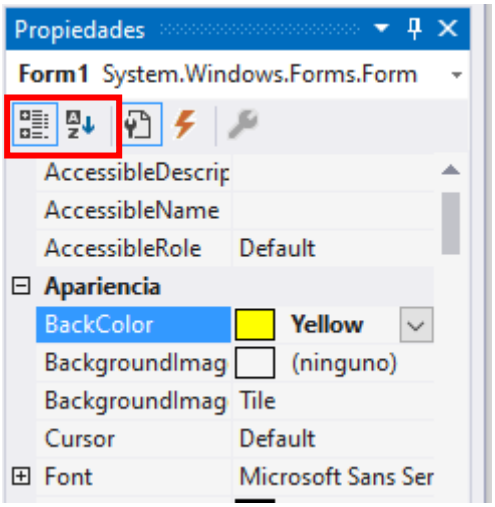




Si seleccionamos la propiedad BackColor podremos cambiar el color de fondo.



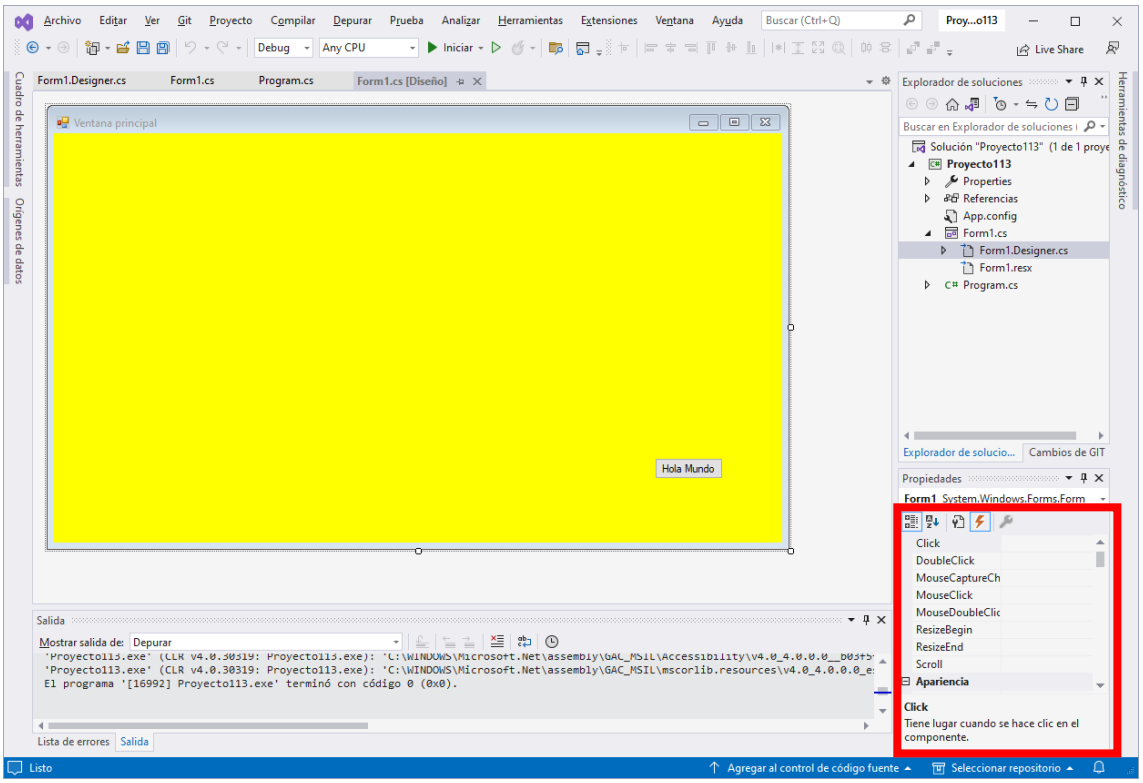
Las propiedades las podemos ver alfabéticamente u ordenadas por tipos.

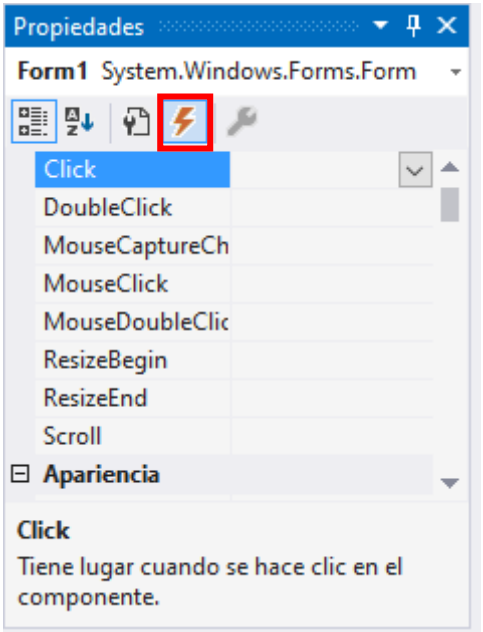


Si le damos a ejecutar muestra el resultado en un interfaz gráfico en lugar de la consola.

Capítulo 112.- Captura de eventos de controles visuales (Windows Forms)

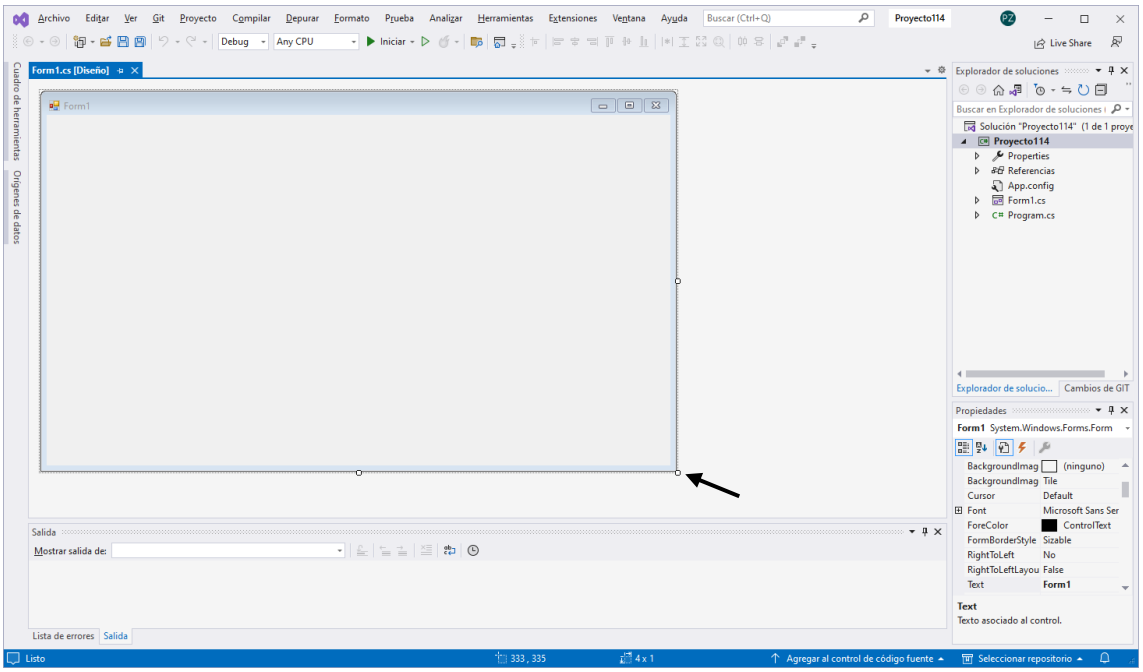
La ventana de eventos coincide con la ventana de propiedades. Para activar la lista de eventos disponible para un objeto debemos presionar:



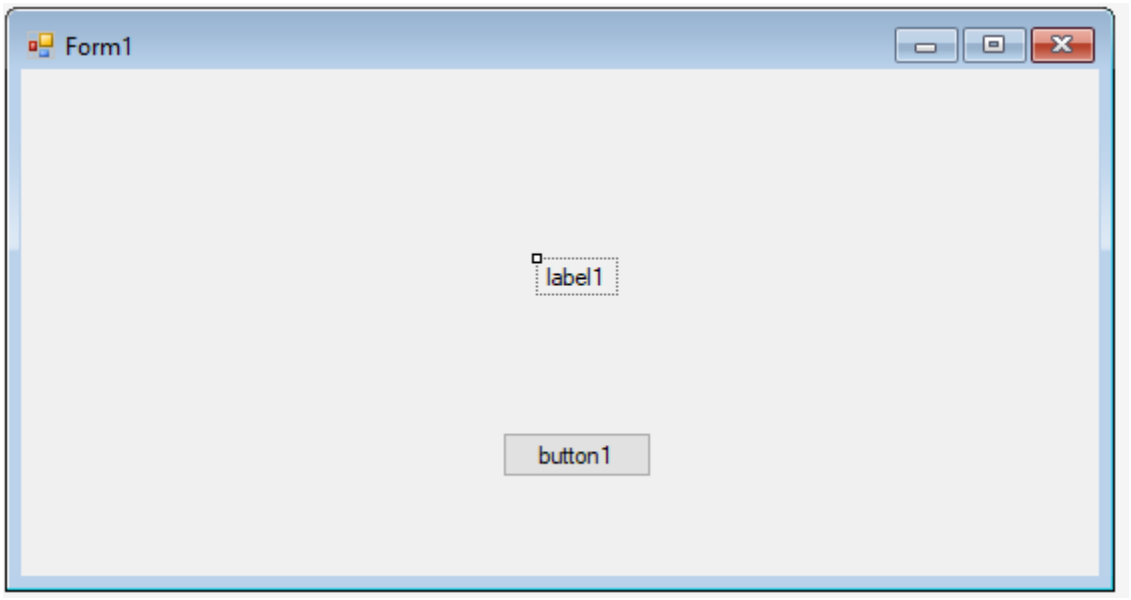


Podemos observar la lista de eventos que pueden reaccionar el objeto seleccionado en ese momento.

Vamos a realizar un nuevo proyecto, en el capítulo anterior se comentan los pasos:



Desde la esquina inferior derecha podemos modificar el tamaño de la ventana.



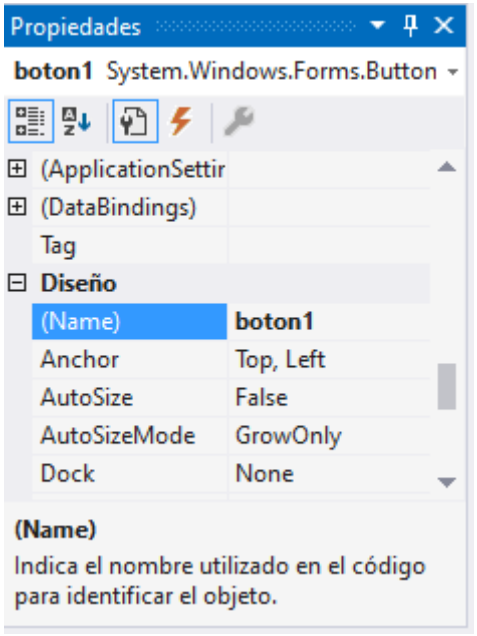
Agregamos un botón y una etiqueta, lo que queremos es que cuando presionemos el botón en la etiqueta parezca un texto.

En el archivo Form1.Designer.cs veremos el código de los objetos que hemos insertado.

```
namespace Proyecto114
{
    3 referencias
    partial class Form1
    {
        /// <summary>
        /// Variable del diseñador necesaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se estén usando.
        /// </summary>
        /// <param name="disposing">true si los recursos administrados se deben desechar; false en caso contrario.</param>
        0 referencias
        protected override void Dispose(bool disposing)
        {
```

```
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    Código generado por el Diseñador de Windows Forms

    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.Label label1;
}
```

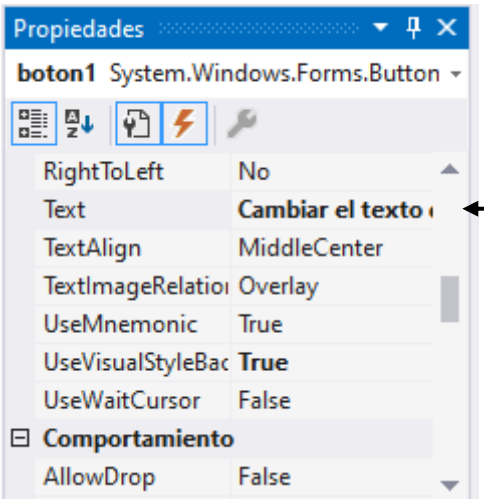
Vamos a renombrar el nombre del botón por boton1.



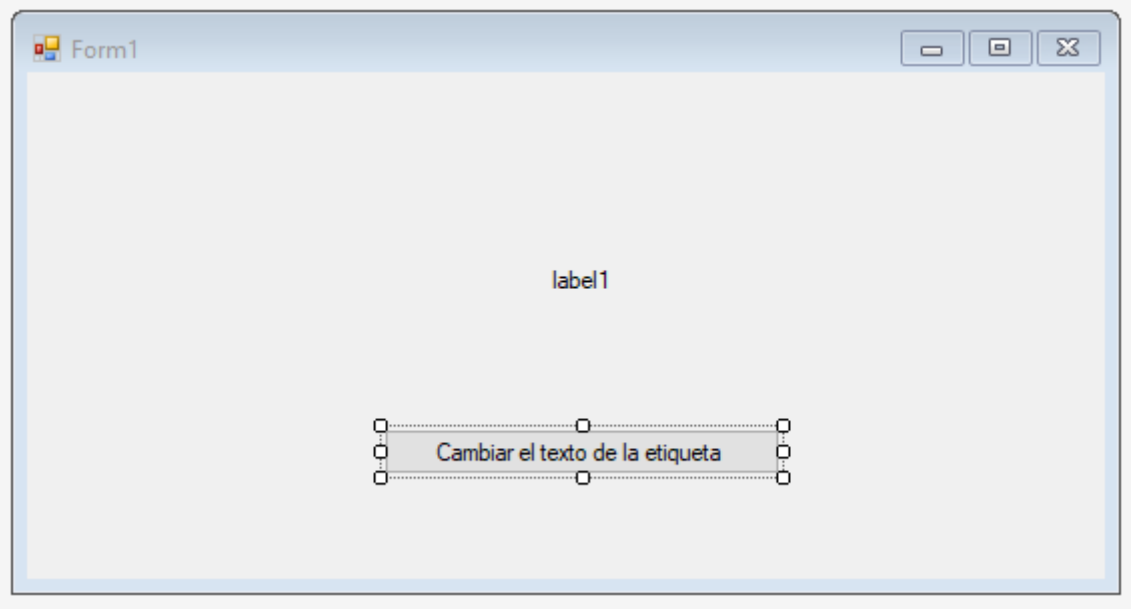

```
    Código generado por el Diseñador de Windows Forms

    private System.Windows.Forms.Button boton1;
    private System.Windows.Forms.Label label1;
}
```

En el código podrás observar como hemos renombrado el nombre del objeto.

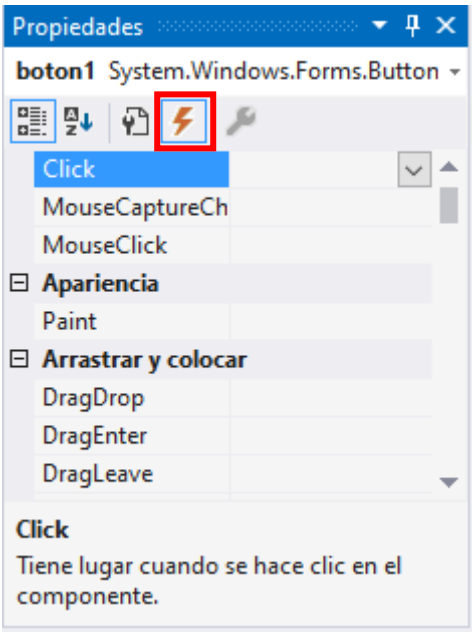
Ahora vamos a cambiar el texto del botón.



Este será el resultado en el botón:



Teniendo seleccionado el botón seleccionaremos eventos.



Haremos doble Click en el evento Click.

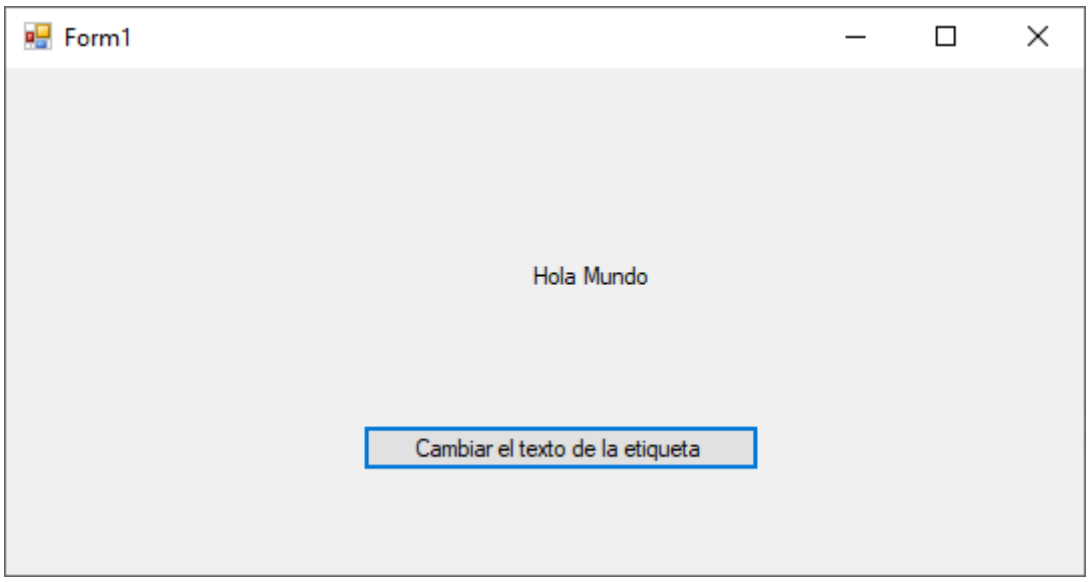
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace Proyecto114
{
    3 referencias
    public partial class Form1 : Form
    {
        1 referencia
        public Form1()
        {
            InitializeComponent();
        }

        1 referencia
        private void boton1_Click(object sender, EventArgs e)
        {

        }
    }
}
```

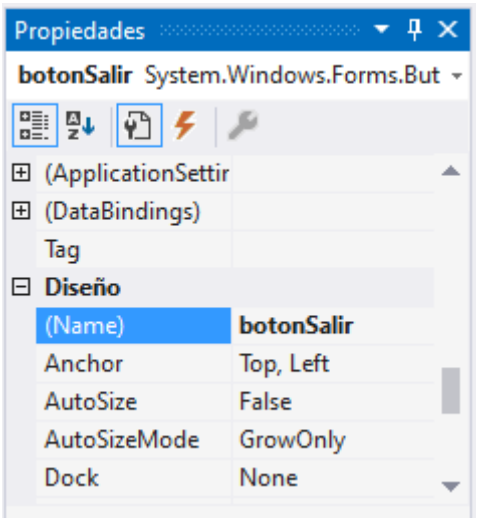
En esta línea de código tendrá que es las instrucciones que queremos que se realicen cuando hagamos clic sobre el botón.

```
        1 referencia
        private void boton1_Click(object sender, EventArgs e)
        {
            label1.Text = "Hola Mundo";
        }
    }
}
```

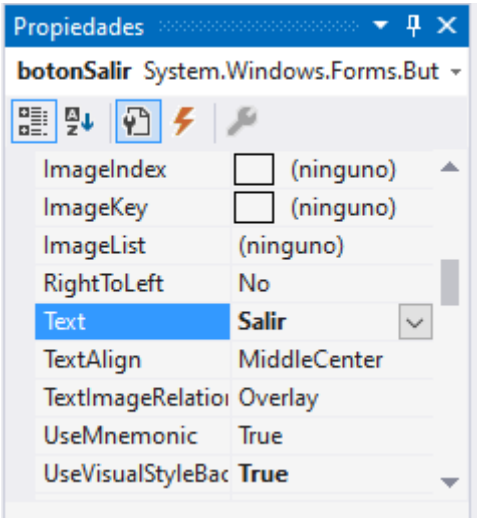
Ahora vamos a ejecutar el programa y haremos Click en el botón.



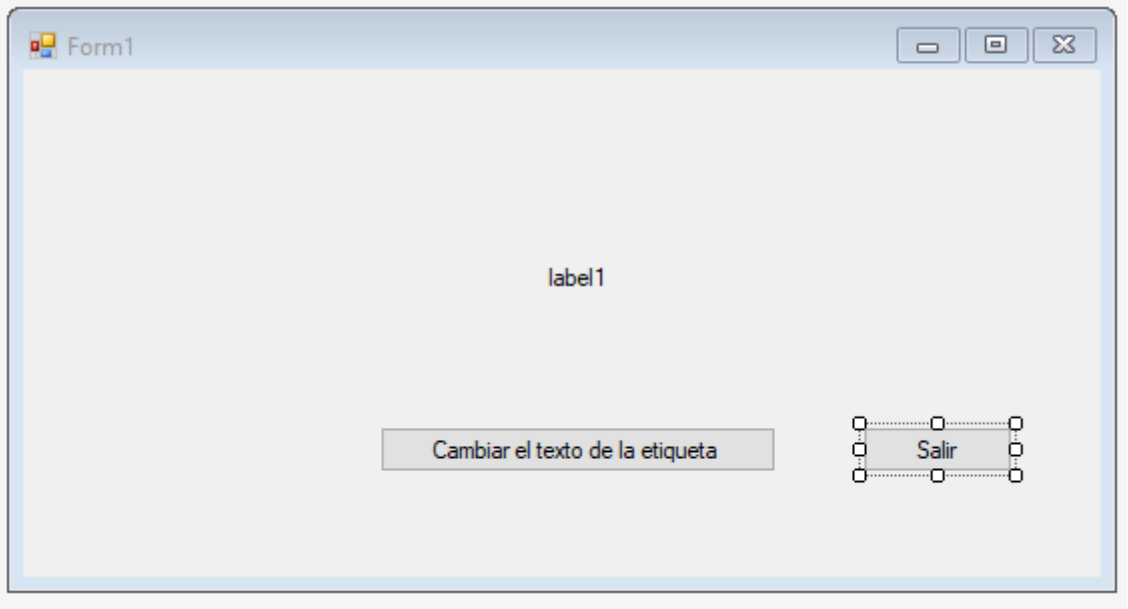
Vamos a agregar un botón para salir.



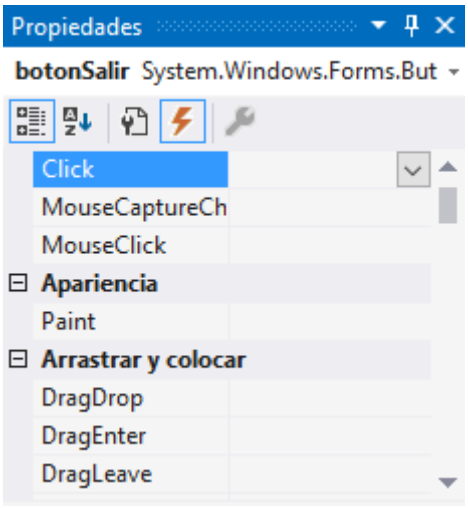
Como nombre del botón le ponemos botonSalir.



Como texto del botón Salir.



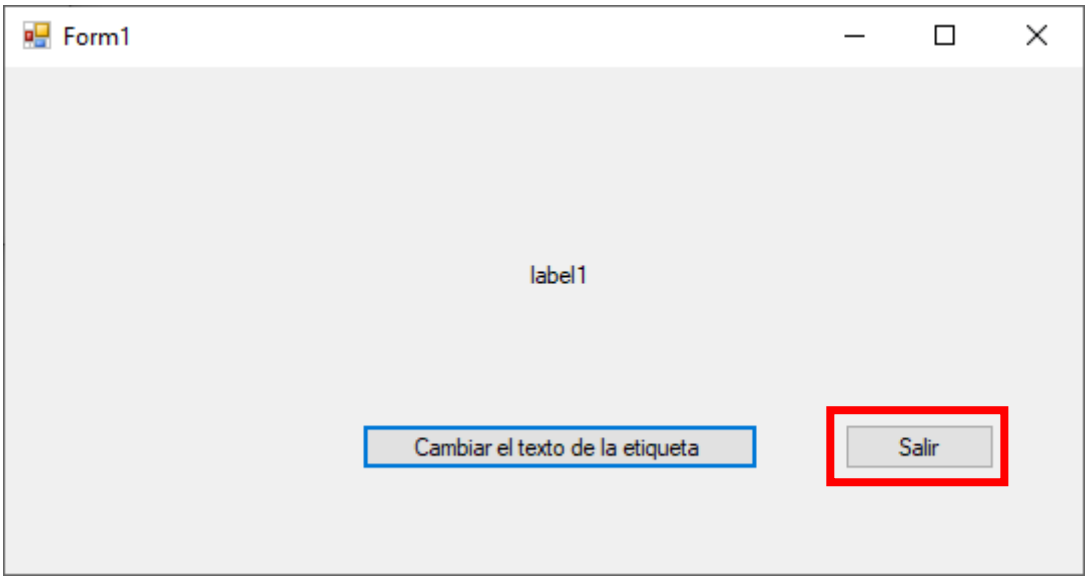
Seleccionamos el botón de eventos.



Doble Click en el evento Click.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto114
{
    3 referencias
    public partial class Form1 : Form
    {
        1 referencia
        public Form1()
        {
            InitializeComponent();
        }

        1 referencia
        private void boton1_Click(object sender, EventArgs e)
        {
            label1.Text = "Hola Mundo";
        }

        1 referencia
        private void botonSalir_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

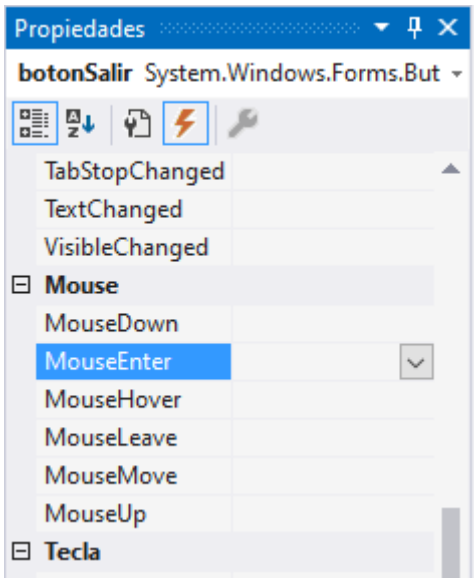
Ejecutamos de nuevo el programa y hacemos click en el botón Salir.



Si se cierra la ventana significa que lo has realizado correctamente.

Podemos observar que para el botón hay muchos más eventos.

Ahora lo que queremos hacer es que cuando el cursor del ratón esté encima del botón sin hacer clic que cambie el texto del botón.

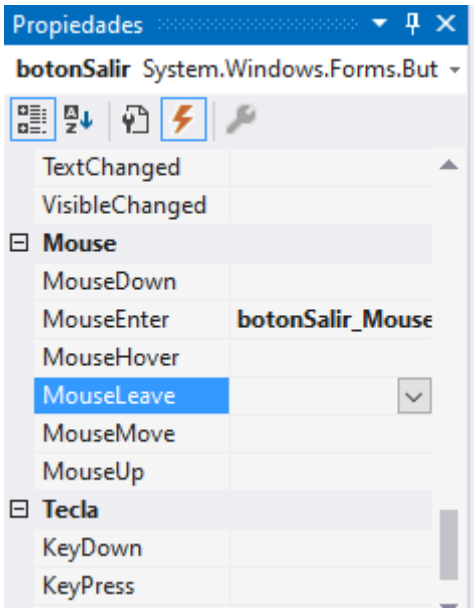


Este evento se llama MouseEnter.

Hacemos doble click sobre este evento.

```
1 referencia
private void botonSalir_MouseEnter(object sender, EventArgs e)
{
    botonSalir.Text = "¿Quiere salir?";
}
```

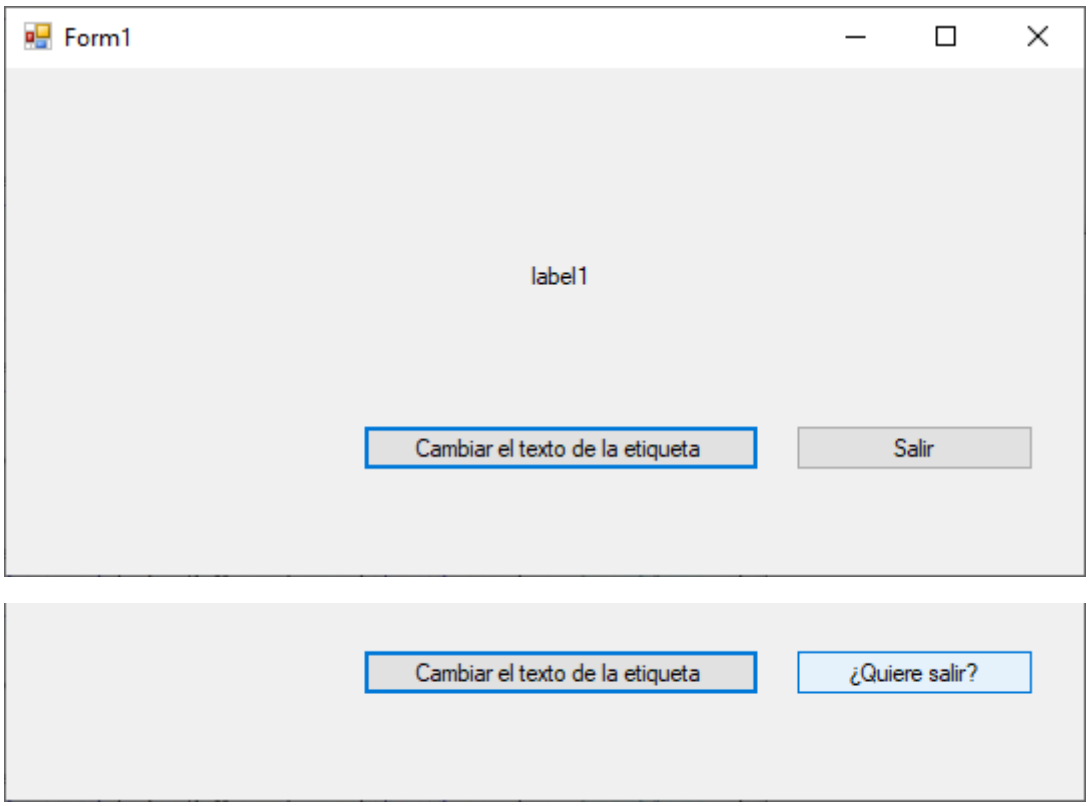
Hay otro evento cuando el mouse sale del botón.



Haremos que vuelva el texto 'Salir'.

```
1 referencia
private void botonSalir_MouseLeave(object sender, EventArgs e)
{
    botonSalir.Text = "Salir";
}
```

Vamos a ejecutar y podremos observar que el botón salir tienen tres eventos, uno cuando hacemos click sobre el para cerrar el programa, otro evento cuando el curso se sitúa encima el botón y un tercer evento cuando el cursor del ratón sale de la zona del botón.

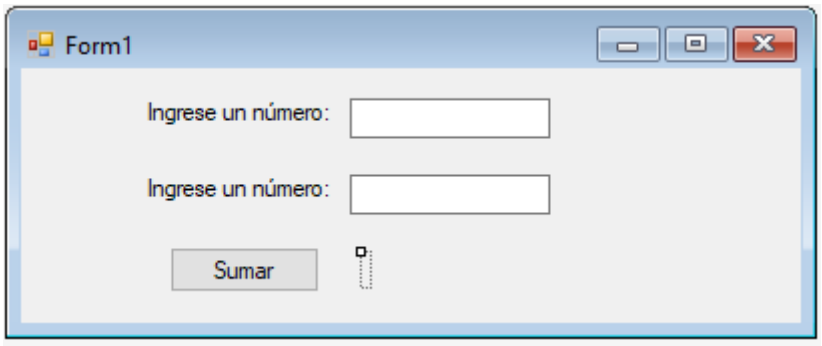


Capítulo 113.- Controles visuales, Button, Label, TextBox (Windows Forms)

Problemas propuestos

1.- Confeccionar un programa que permita ingresar dos valores enteros por teclado y al presionar un botón mostrar en un Label la suma de dichos valores.

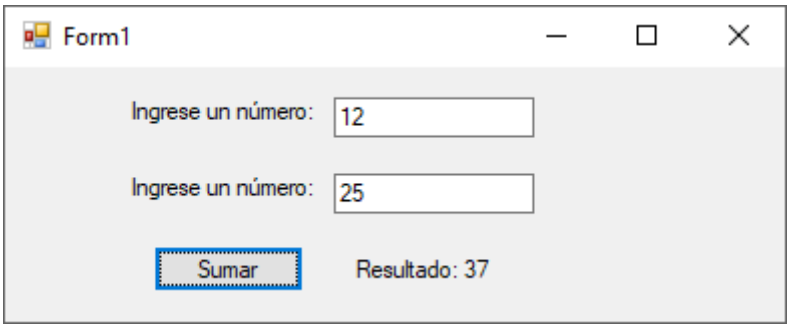
Realizamos el siguiente diseño:



El código de evento Click del botón:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    string s1 = textBox1.Text;
    string s2 = textBox2.Text;
    int valor1 = int.Parse(s1);
    int valor2 = int.Parse(s2);
    int suma = valor1 + valor2;
    string s3 = suma.ToString();
    label3.Text = "Resultado: " + s3;
}
```

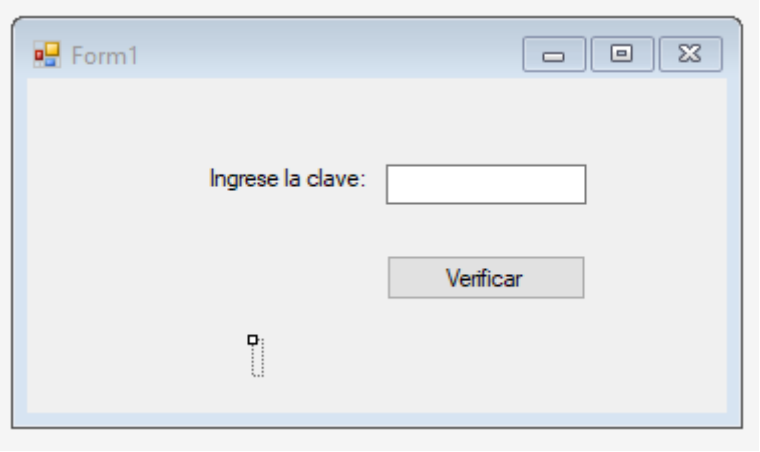
Este será el resultado:



2.- Solicitar que se ingrese una clave. Si se ingresa la cadena "abc123" mostrar un mensaje de clave correcta en caso contrario mostrar clave incorrecta.

Utilizar un control de tipo TextBox para el ingreso de la clave y un Label para mostrar el resultado al presionar un botón.

Inicializar la propiedad UseSystemPasswordChar con el valor true (esto hace que cuando el operador escriba caracteres dentro del TextBox se visualicen como asteriscos).

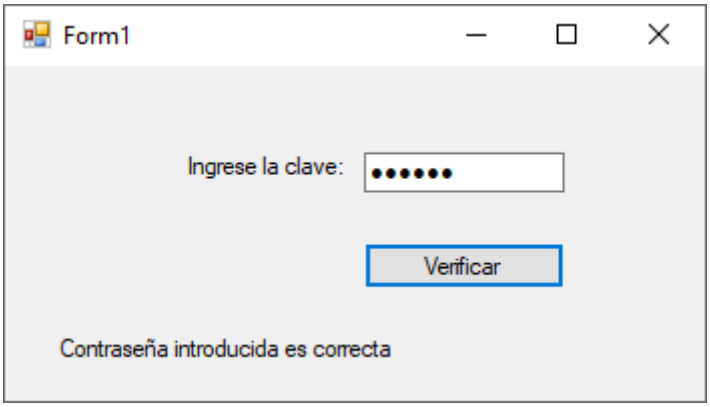


Permitir el ingreso de una clave hasta 10 caracteres modificando la propiedad MaxLength con el valor 10.

Este es el código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    string s1 = textBox1.Text;
    if (s1 == "abc123")
    {
        label2.Text = "Contraseña introducida es correcta";
    }
    else
    {
        label2.Text = "Contraseña introducida es incorrecta";
    }
}
```

Si ejecutamos este será el resultado introduciendo una contraseña correcta:



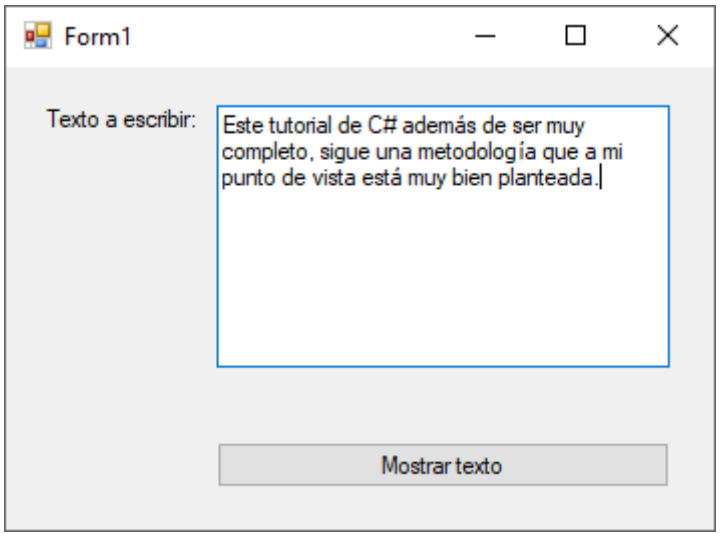
3.- Disponer un control de tipo TextBox e inicializar la propiedad Multiline con el valor true (esto permite ingresar múltiples líneas dentro de un TextBox).

Cuando se presiona el botón se muestra en cuadro de mensajes (MessageBox) el texto ingresado en el textBox2:

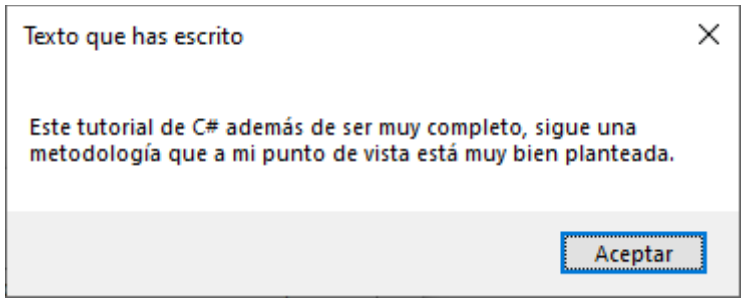
Este será el código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    String s1 = textBox1.Text;
    MessageBox.Show(s1, "Texto que has escrito");
}
```

Vamos a ejecutar:



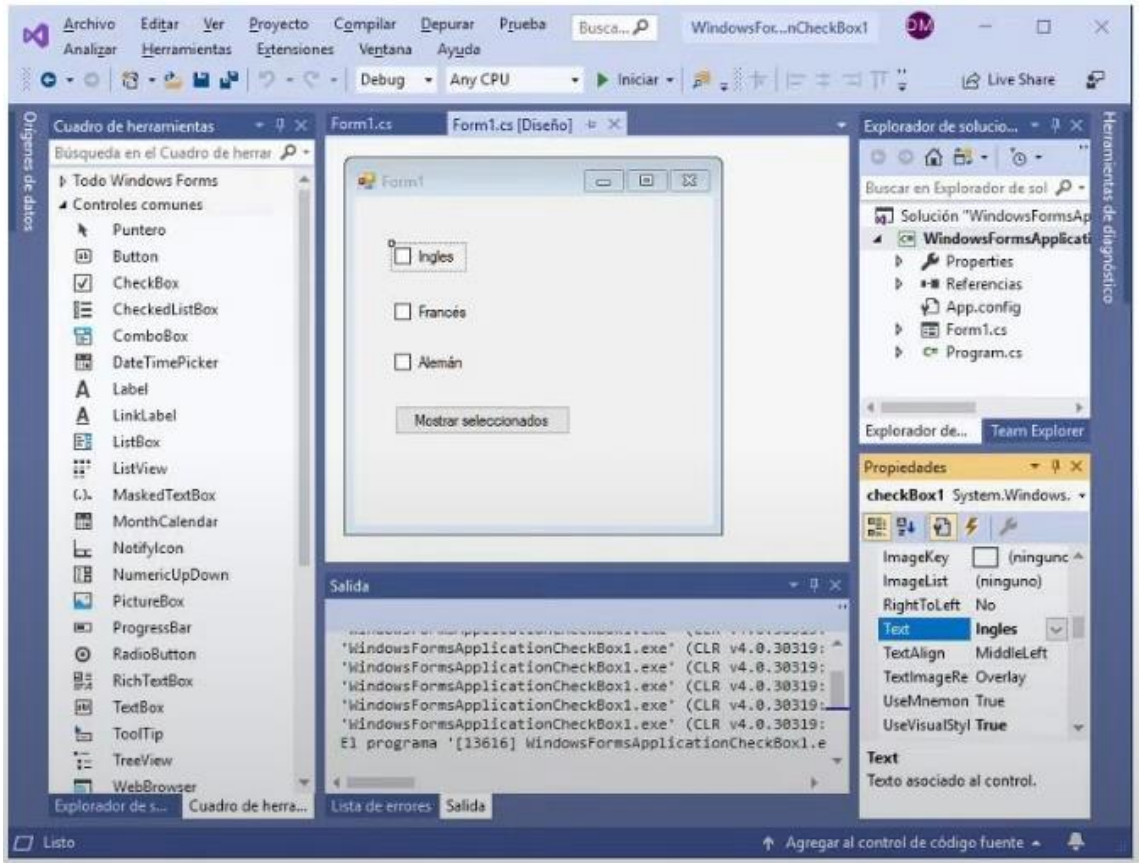
Seleccionamos el botón 'Mostrar texto'.



Capítulo 114.- Control visual CheckBox (Windows Forms)

Problema propuesto

1.- Confeccionar un programa que muestre 3 objetos de la clase CheckBox con etiquetas de tres idiomas. Cuando se presiona un botón mostrar en la barra de títulos del Form todos los CheckBox seleccionados hasta el momento.

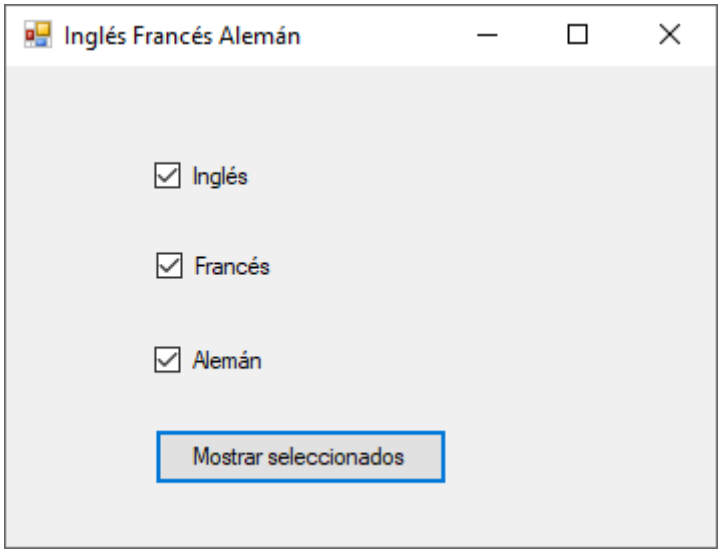


Este será el código:

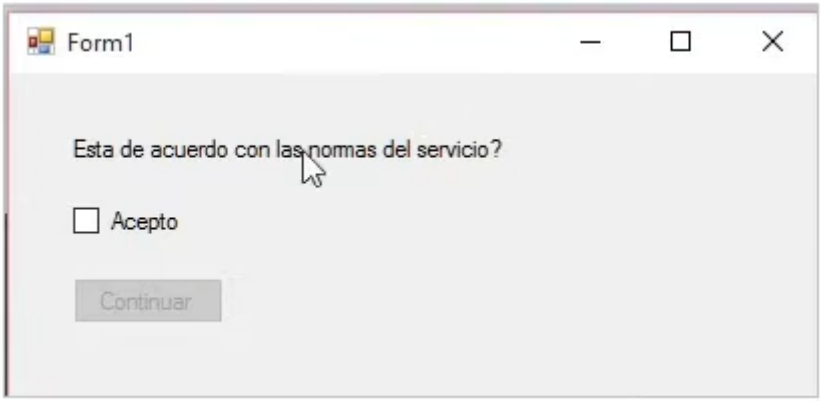
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    Text = "";
    if (checkBox1.Checked == true)
    {
        Text = Text + "Inglés ";
    }
    if (checkBox2.Checked == true)
    {
        Text = Text + "Francés ";
    }
    if (checkBox3.Checked == true)
    {
        Text = Text + "Alemán ";
    }
```

```
}
```

Cuando ejecutemos este será el resultado:



2.- Disponer un control Label que muestre el siguiente mensaje: "Estás de acuerdo con las nomas del servicio?", luego un CheckBox y finalmente un objeto de tipo Button desactivado (propiedades Enabled con false). Cuando se active el CheckBox debemos activar el botón (para esto debemos responder al evento).



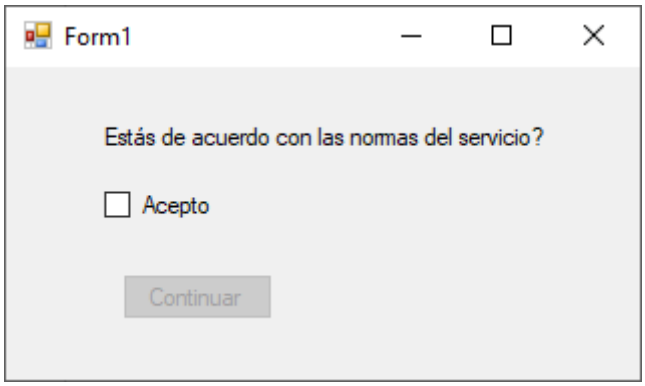
Debemos implementar el evento CheckedChange del objeto checkBox1 (preguntamos si el CheckBox se encuentra seleccionado o no, en caso de estar seleccionado activamos el botón asignando la propiedad Enabled el valor true).

Este es el código:

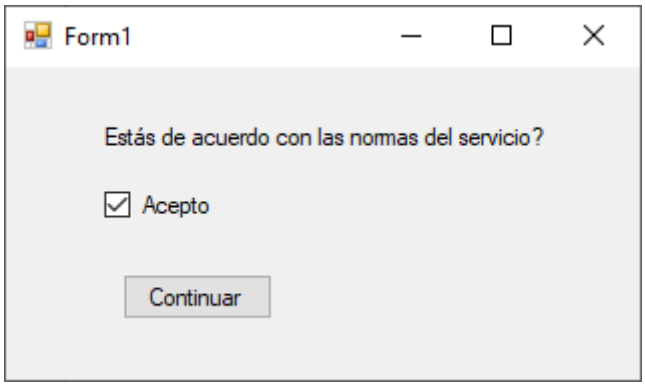
```
1 referencia
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked == true)
    {
        button1.Enabled = true;
    }
    else
    {
```

```
            button1.Enabled = false;
        }
    }
```

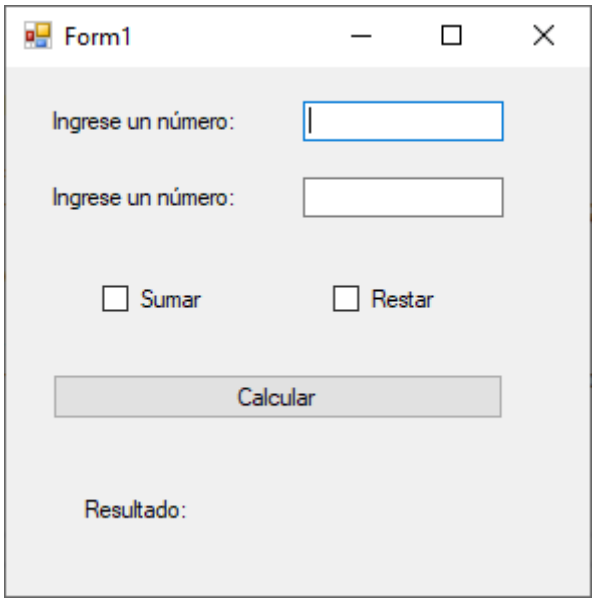
Si ejecutamos este será el resultado:



Activamos la casilla Acepto.



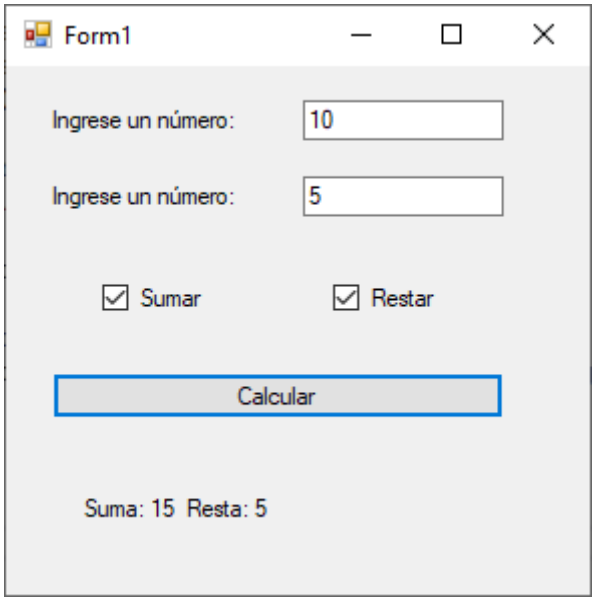
3.- Ingresar dos valores enteros por teclado en controles de tipo TextBox y mediante dos CheckBox indicar si queremos sumar y/o restarlos.



Este es el código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    label3.Text = "";
    int valor1 = int.Parse(textBox1.Text);
    int valor2 = int.Parse(textBox2.Text);
    if(checkBox1.Checked)
    {
        int sumar = valor1 + valor2;
        label3.Text = "Suma: " + sumar.ToString() + "  ";
    }
    if (checkBox2.Checked)
    {
        int restar = valor1 - valor2;
        label3.Text = label3.Text + "Resta: " + restar.ToString();
    }
}
```

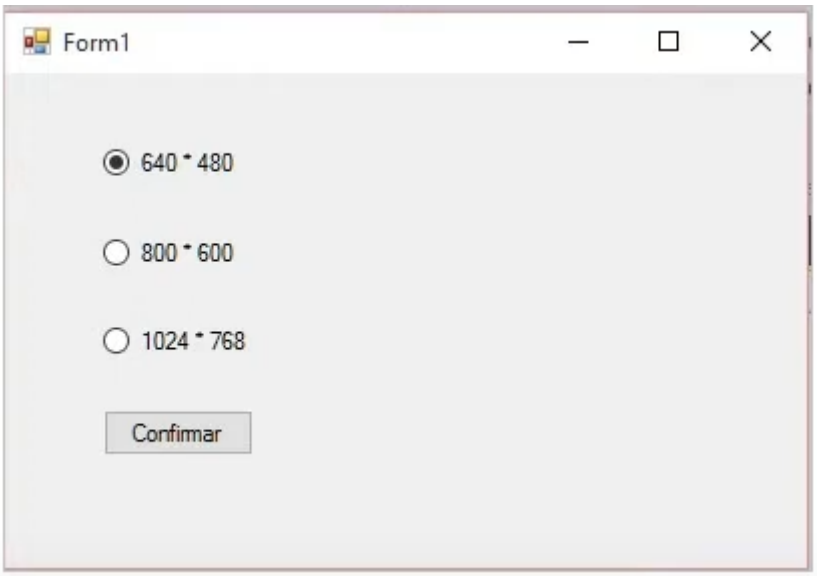
Vamos a ejecutar activando la suma y la resta:



Capítulo 115.- Controles visuales RadioButton, GroupBox y Panel (Windows Forms)

Problemas propuestos

1.- Confeccionar un programa que muestre 3 objetos de la clase RadioButton que permita configurar en ancho y alto del Form. Cuando se presione un botón actulizar el ancho y alto.

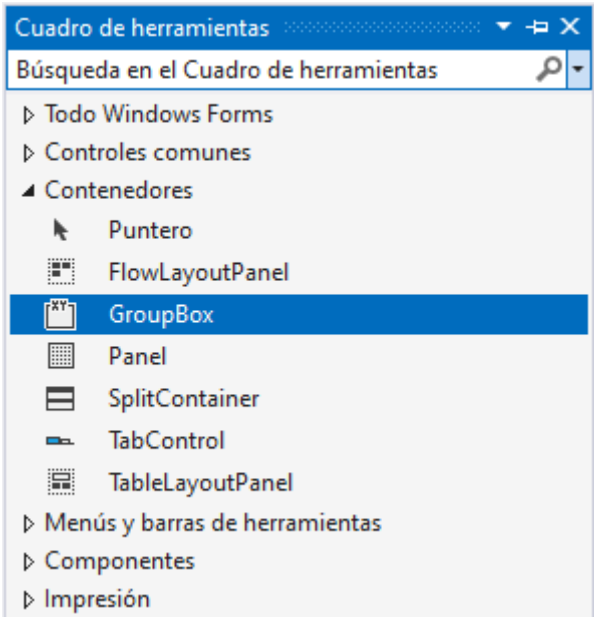


Este es el código:

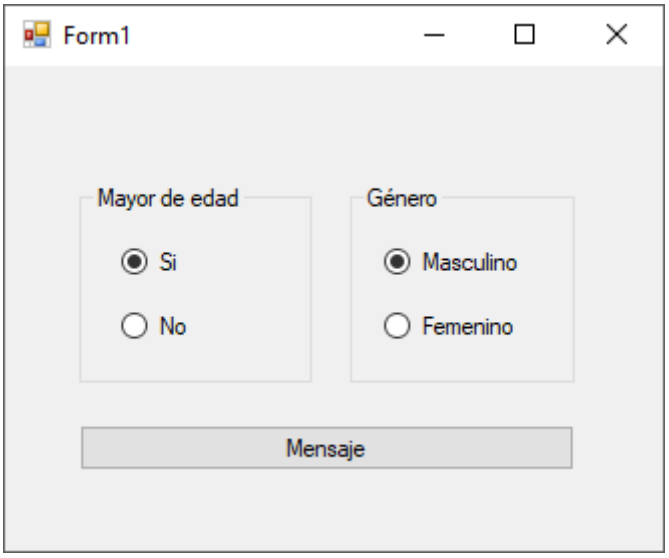
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    if (radioButton1.Checked)
    {
        Width = 640;
        Height = 480;
    }
    else
    {
        if (radioButton2.Checked)
        {
            Width = 800;
            Height = 600;
        }
        else
        {
            if (radioButton3.Checked)
            {
                Width = 1024;
                Height = 768;
            }
        }
    }
}
```

2.- Disponer 2 RadioButton para permitir seleccionar si una persona es mayor de edad o menor. Y por otro lado otros dos RadioButton que permita seleccionar si es de género masculino o femenino dicha persona.

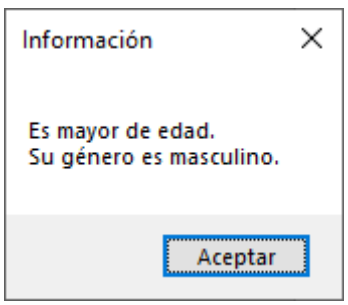
En el apartado de Contenedores encontramos GroupBox.



Vamos a realizar el siguiente diseño:



Hemos agregado un botón para que nos muestre lo que hemos seleccionado, si seleccionamos el botón:

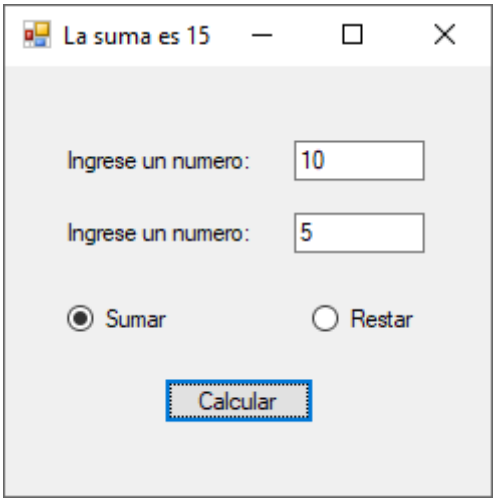


Este será el código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    String mensaje = "";
    if (radioButton1.Checked)
    {
        mensaje = "Es mayor de edad.";
    }
    else
    {
        if (radioButton2.Checked)
        {
            mensaje = "Es menor de edad.";
        }
    }
    if (radioButton3.Checked )
    {
        mensaje = mensaje + "\nSu género es masculino.";
    }
    else
    {
        if (radioButton4.Checked)
            {
                mensaje = mensaje + "\nSu género es femenino.";
            }
        }
    MessageBox.Show( mensaje, "Información");
}
```

3.- Permitir el ingreso de dos números en controles de tipo TextBox y mediante dos controles de tipo RadioButton permitir seleccionar si queremos sumarlos o restarlos. Al presionar un botón mostrar en el título del Form el resultado de la operación.

El diseño:



El código:

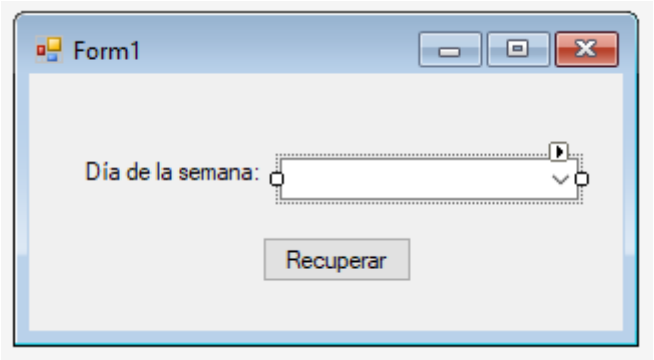
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    int valor1 = int.Parse(textBox1.Text);
    int valor2 = int.Parse(textBox2.Text);
    if (radioButton1.Checked )
    {
        int sumar = valor1 + valor2;
        Text = "La suma es "+ sumar.ToString();
    }
    else
    {
        if (radioButton2.Checked)
        {
            int restar = valor1 - valor2;
            Text = "La resta es " + restar.ToString();
        }
    }
}
```

Capítulo 116.- Control visual ComboBox

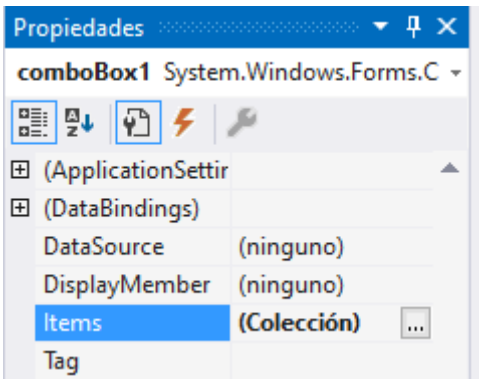
Problemas propuestos

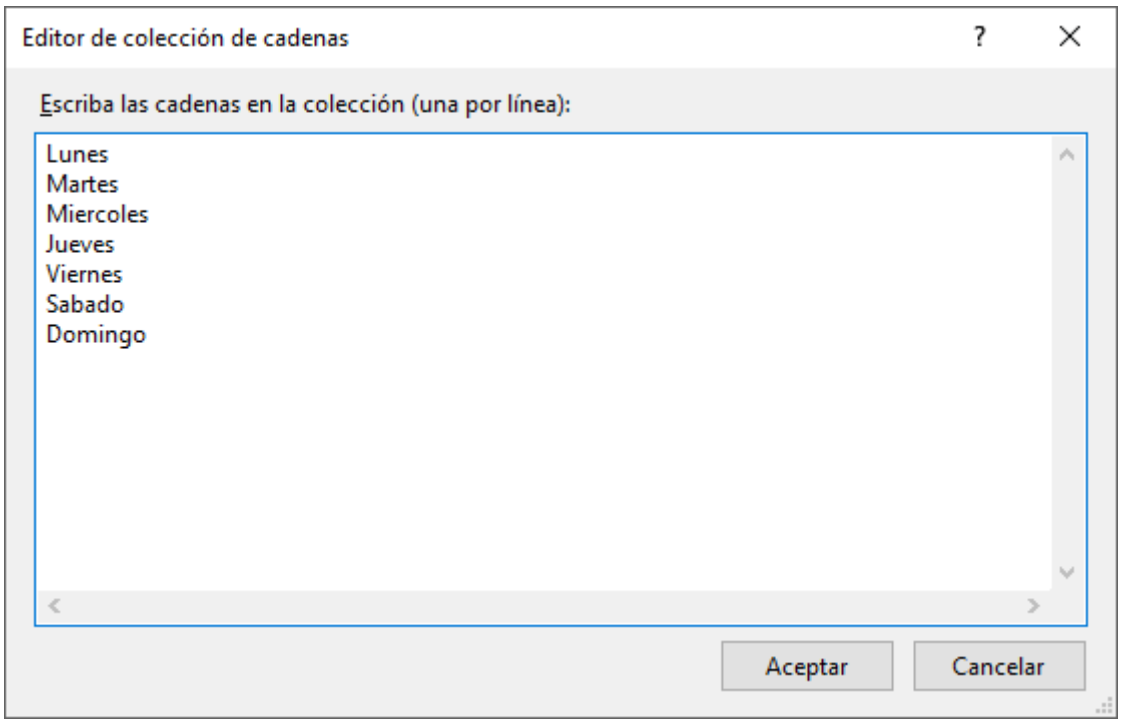
1.- Cargar un ComboBox los nombres de los días de la semana. Al seleccionar alguno mostrar en la barra de título del Form el string seleccionado.

Este será el diseño:



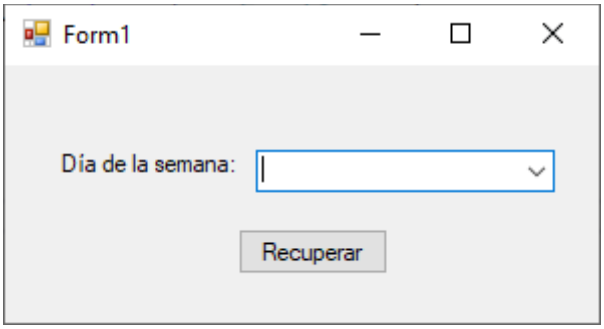
Para agregar los días de la semana seleccionaremos la propiedad Items.



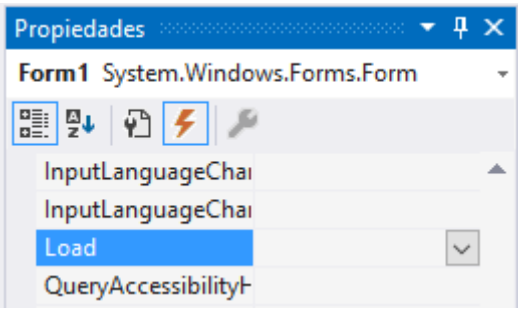


Este será el código cuando ejecutemos el botón después de haber seleccionado un día de la semana:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    Text = comboBox1.Text;
}
```



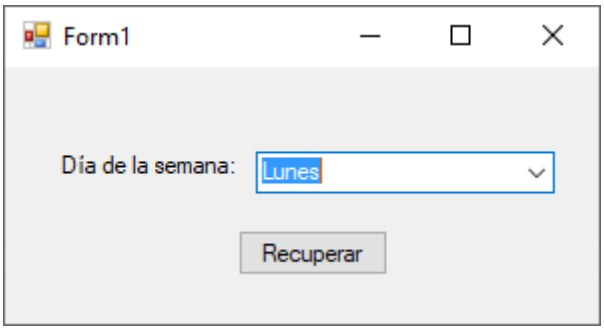
Si queremos que al cargarse el programa salga un día de la semana por defecto utilizaremos un evento del Form.



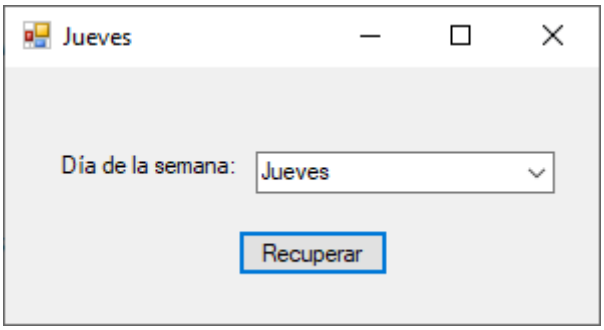
Agregaremos el siguiente código:

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    comboBox1.SelectedIndex = 0;
}
```

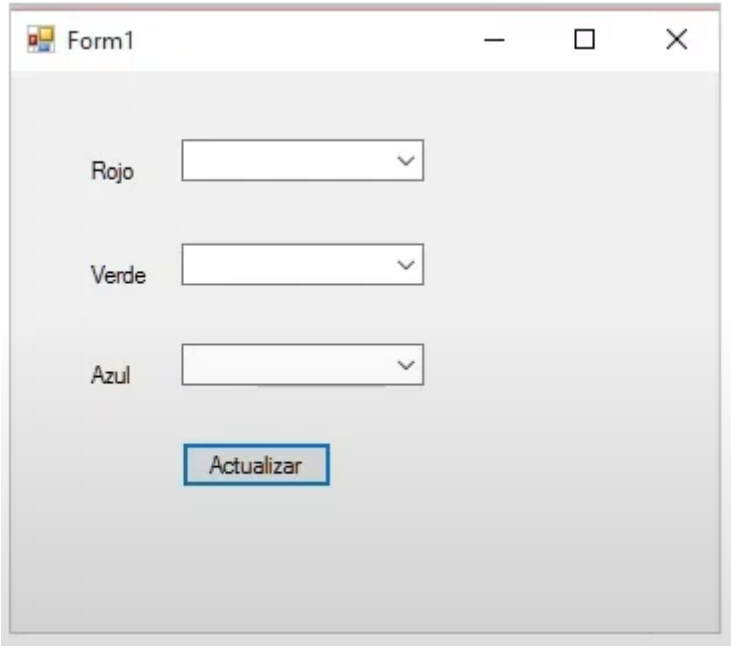
Queremos que se muestre por defecto el Lunes.



Igualmente lo podrás cambiar a otro día de la semana y seleccionar el botón Recuperar.



2.- Disponer tres controles de tipo ComboBox con valores entre 0 y 255 (cada uno representa la cantidad de rojo, verde y azul). luego al presionar un botón pinta el fondo del Form con el color que se genera combinando los valores de los ComboBox.

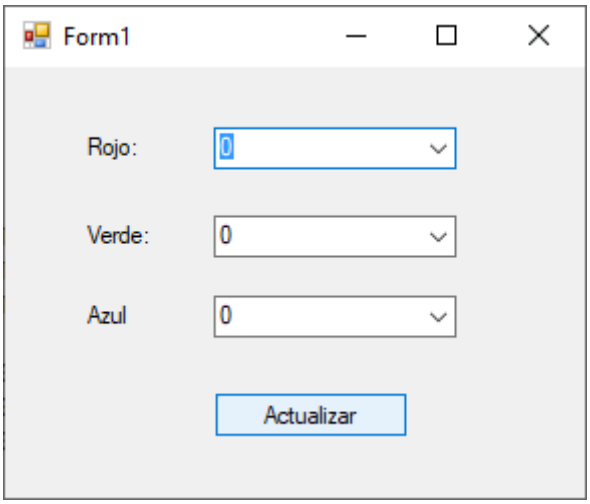


Desde el evento load del Form vamos a realizar es siguiente código para cargar los valores desde el 0 al 255.

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    for (int f=0; f<=255; f++)
    {
        comboBox1.Items.Add(f.ToString());
        comboBox2.Items.Add(f.ToString());
        comboBox3.Items.Add(f.ToString());
    }
    comboBox1.SelectedIndex = 0;
    comboBox2.SelectedIndex = 0;
    comboBox3.SelectedIndex = 0;
}
```

A continuación cuando ejecutemos veremos que ya podremos seleccionar valores desde 0 a 255.

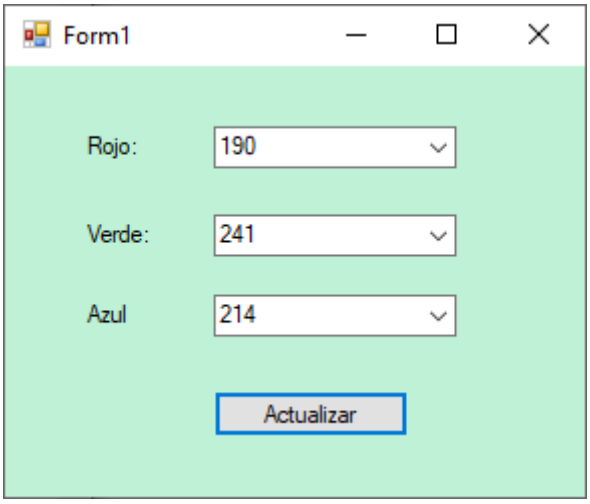
Queremos que muestre el valor por defecto de 0.



Ahora vamos a programar el evento del botón al hacer Click.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    int rojo = int.Parse(comboBox1.Text);
    int verde = int.Parse(comboBox2.Text);
    int azul = int.Parse(comboBox3.Text);
    BackColor = Color.FromArgb(rojo, verde, azul);
}
```

Ahora cuando ejecutemos ya podemos selección un valor del 0 al 255 para el color Rojo, Verde y Azul y le damos al botón Actualizar.



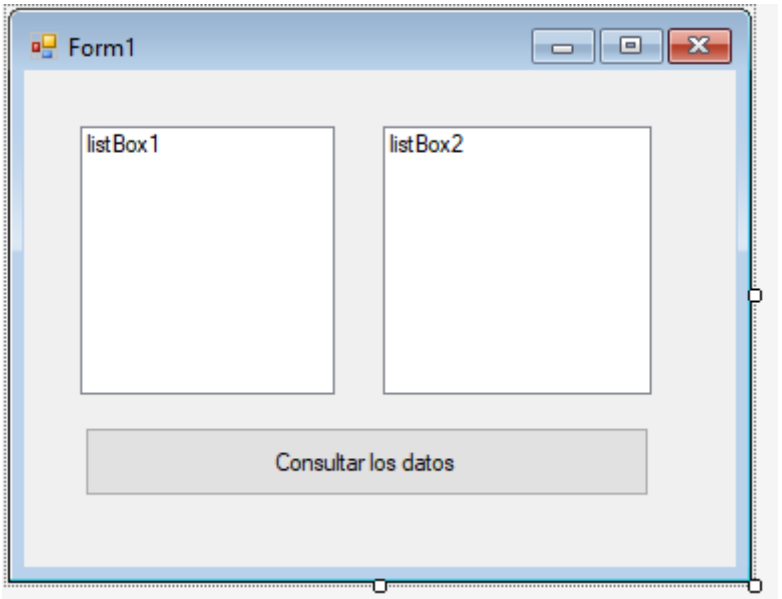
Capítulo 117.- Control visual ListBox (Windows Forms)

Problema propuestos

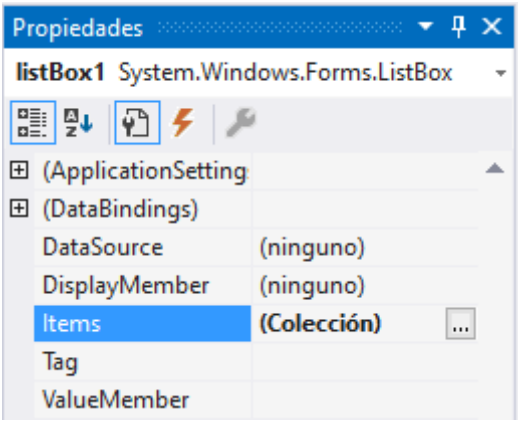
1.- Disponer en un control de tipo ListBox los días de la semana, permitir seleccionar varios días. En un segundo ListBox disponer los textos "mañana", "tarde" y "noche", permitir seleccionar solo uno de ellos.

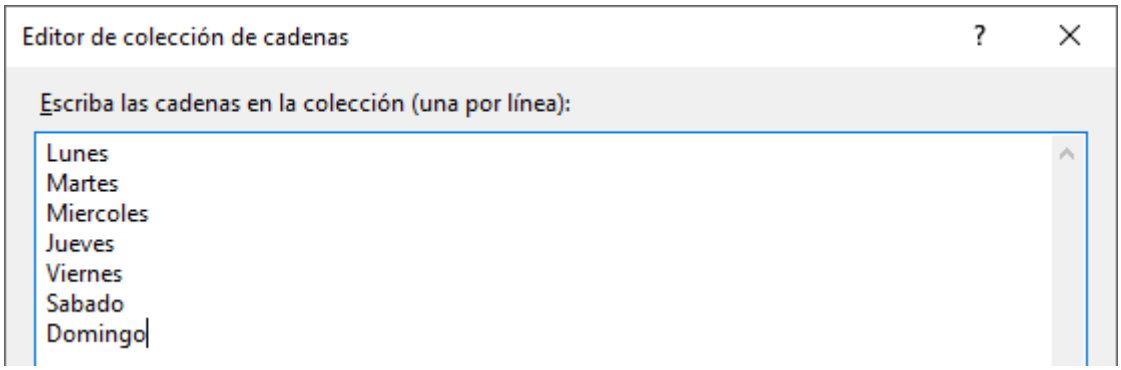
Recuperar los datos seleccionados al presionar un botón.

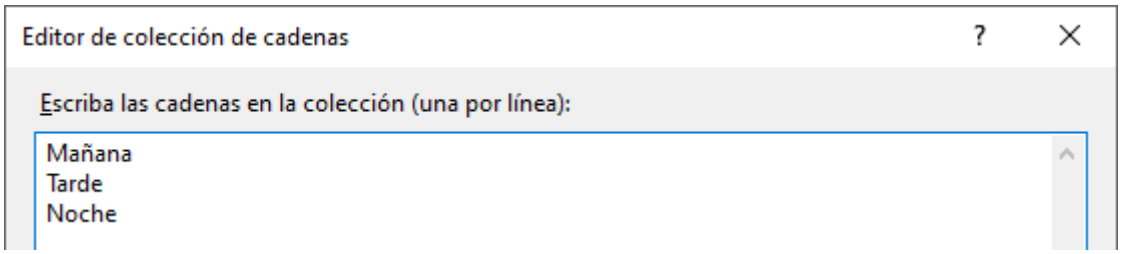
vamos a realizar el diseño:

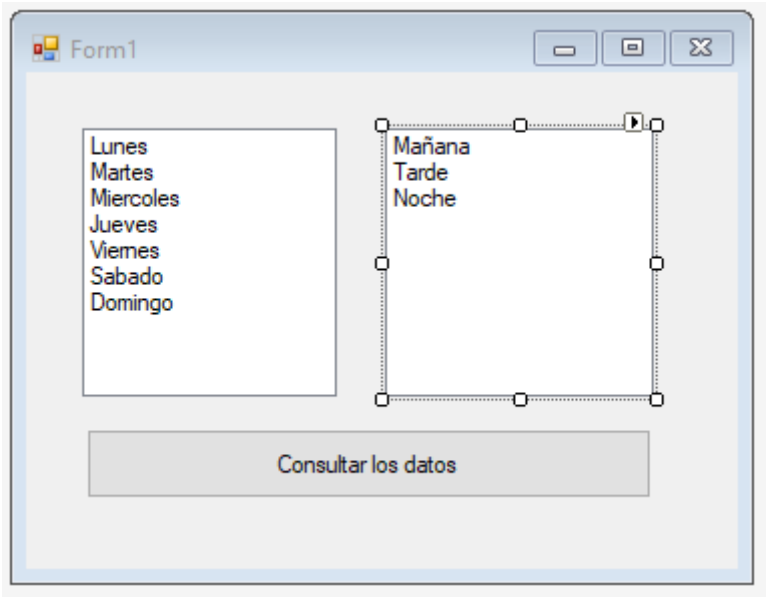


Seleccionaremos la propiedad Items para las opciones de cada lista.

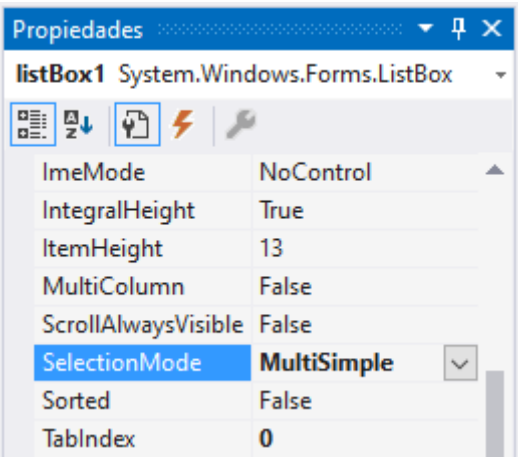




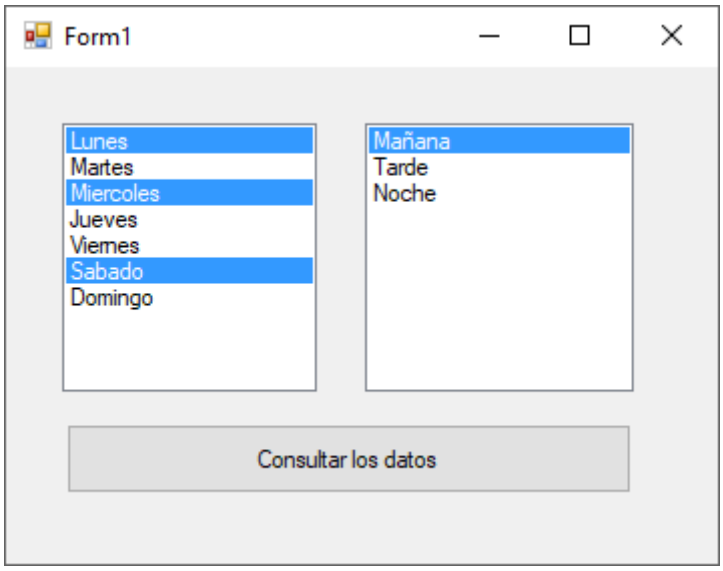




Por defecto está configurado para seleccionar un elemento para cambiarlos y poder seleccionar varios días de la semana.



La propiedad SelectionMode la cambiamos a MultiSimple.



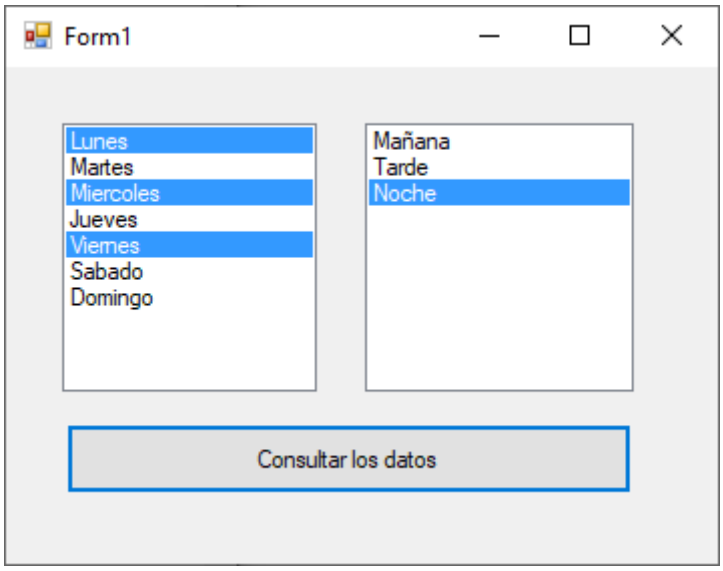
En el apartado de los días podemos seleccionar varios días, en cambio en los turno solo podemos seleccionar un turno.

Ahora cuando realicemos un click sobre el botón queremos que muestre la información que tenemos seleccionada.

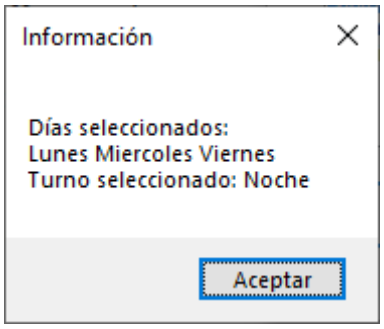
Este será el código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    string mensajeDias = "";
    ListBox.SelectedObjectCollection lista = listBox1.SelectedItems;
    mensajeDias = mensajeDias + "Días seleccionados:\n";
    for (int f=0; f<lista.Count; f++)
    {
        mensajeDias = mensajeDias + lista[f].ToString() + " ";
    }
    mensajeDias = mensajeDias + "\n";
    ListBox.SelectedObjectCollection lista2 = listBox2.SelectedItems;
    mensajeDias = mensajeDias + "Turno seleccionado: ";
    for (int f = 0; f < lista2.Count; f++)
    {
        mensajeDias = mensajeDias + lista2[f].ToString() + " ";
    }
    MessageBox.Show(mensajeDias, "Información");
}
```

Cuando ejecutemos este será el resultado:



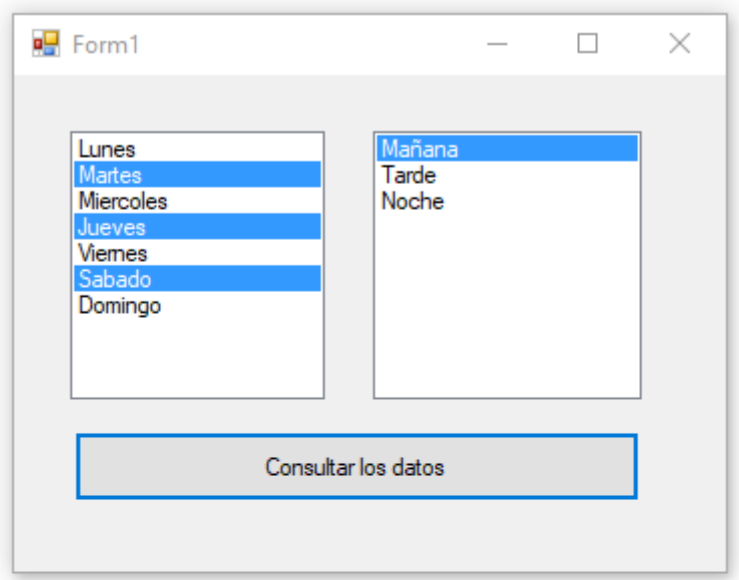
Seleccionaremos varios días de la semana y solo un turno, le damos al botón Consultar los datos:

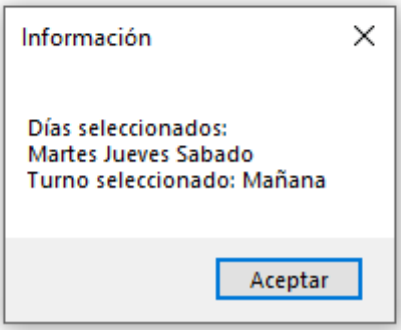


Otra forma de realizarlo es utilizando el foreach en lugar del for, este será el código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    string mensajeDias = "";
    ListBox.SelectedObjectCollection lista1 = listBox1.SelectedItems;
    mensajeDias = mensajeDias + "Días seleccionados:\n";
    foreach (var item in lista1)
    {
        mensajeDias = mensajeDias + item.ToString() + " ";
    }
    mensajeDias = mensajeDias + "\n";
    ListBox.SelectedObjectCollection lista2 = listBox2.SelectedItems;
    mensajeDias = mensajeDias + "Turno seleccionado: ";
    foreach(var item in lista2)
    {
        mensajeDias = mensajeDias + item.ToString() + " ";
    }
    MessageBox.Show(mensajeDias, "Información");
}
```

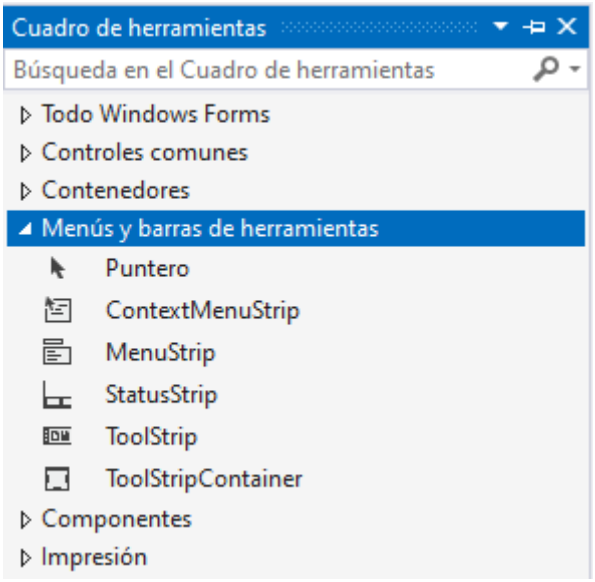
si ejecutamos este será el resultado:


Form1
Lunes
Martes
Miercoles
Jueves
Viernes
Sabado
Domingo
Mañana
Tarde
Noche
Consultar los datos


Información
Días seleccionados:
Martes Jueves Sabado
Turno seleccionado: Mañana
Aceptar

Capítulo 118.- Controles visuales MenuStrip y ToolStripMenuItem

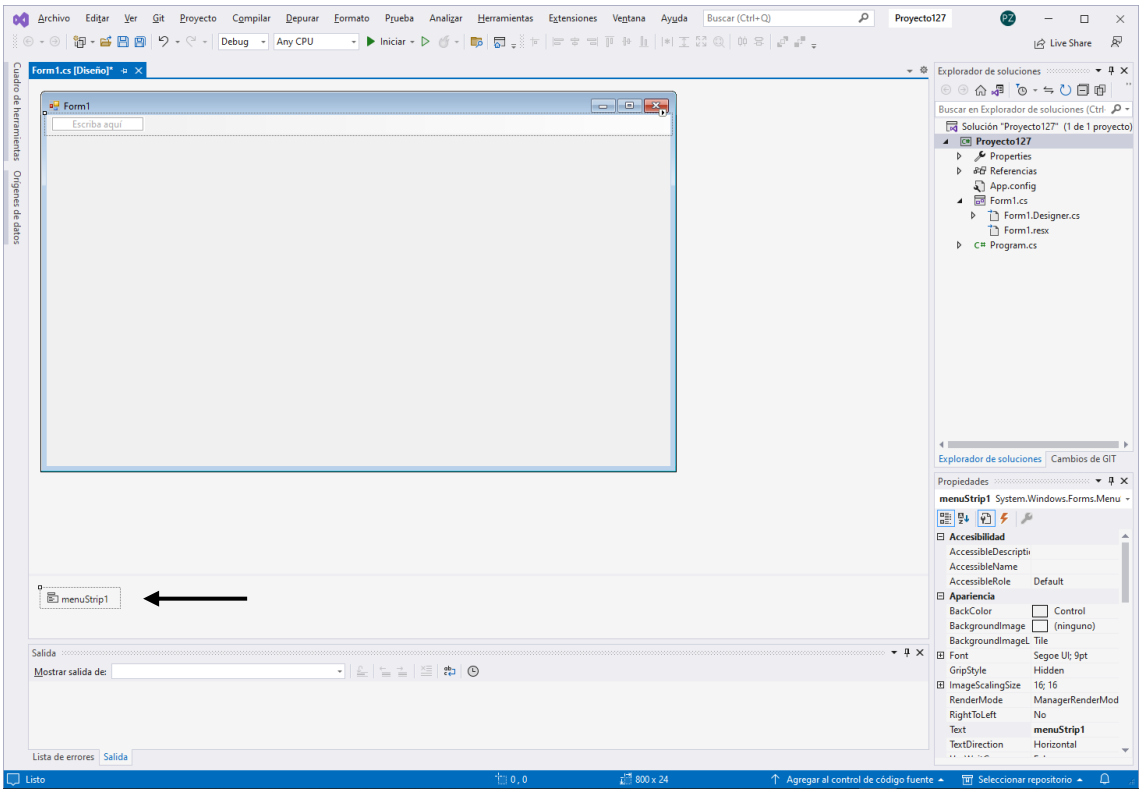
Partiendo de un nuevo proyecto.



Vamos a trabajar con los objetos que hay en Menús y barras de herramienta.

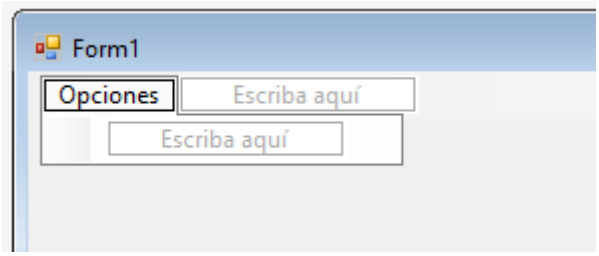
Para poder realizar un menú necesitamos un objeto de la clase MenuStrip.

Lo vamos a arrastrar a nuestro formulario.

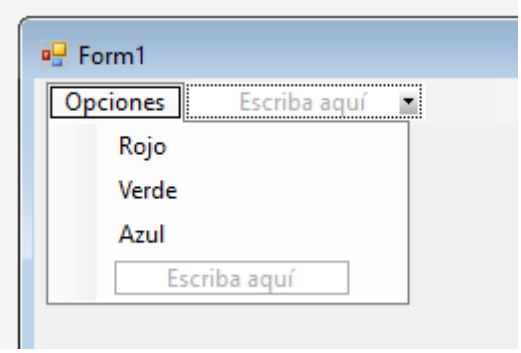


Observamos que en nuestro formulario no se observa ningún objeto pero en la parte inferior observamos un botón con dicho nombre.

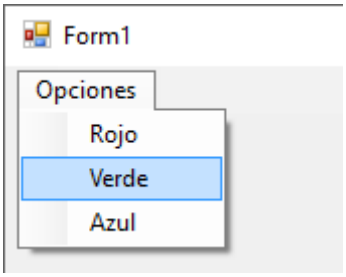
Una vez lo tengamos seleccionado podremos definir el menú.



A continuación vamos a escribir las opciones que tiene este menú.



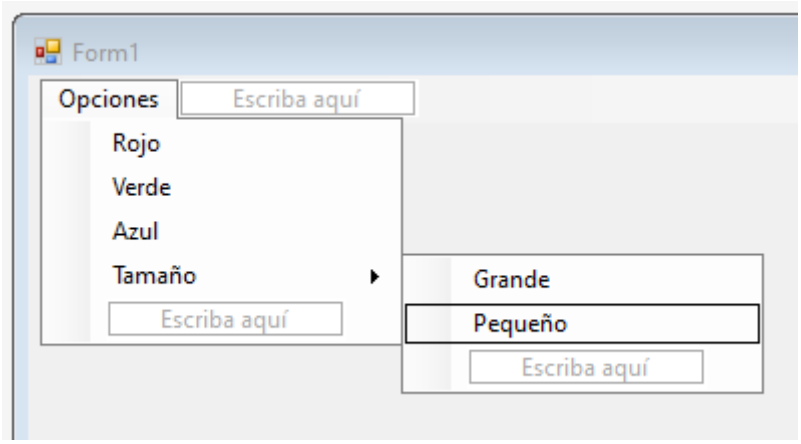
Si ahora lo ejecutamos el menú de opciones ya está visible.



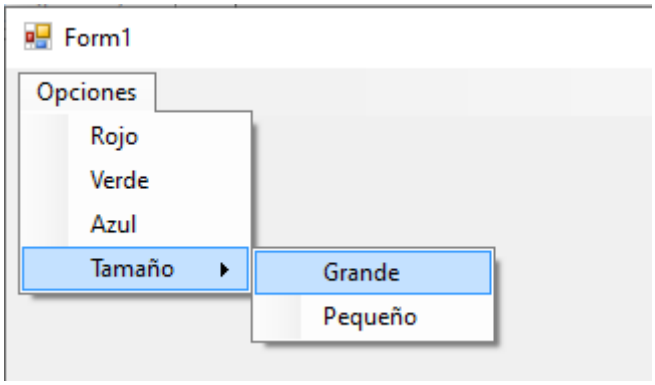

```
namespace Proyecto127
{
    3 referencias
    partial class Form1
    {
        /// <summary>
        /// Variable del diseñador necesaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Limpiar los recursos que se estén usando.
        /// </summary>
        /// <param name="disposing">true si los recursos administrados se deben desechar; false en caso contrario.</param>
        0 referencias
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Código generado por el Diseñador de Windows Forms

        private System.Windows.Forms.MenuStrip menuStrip1;
        private System.Windows.Forms.ToolStripMenuItem opcionesToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem rojoToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem verdeToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem azulToolStripMenuItem;
    }
}
```

Ha generado el código automáticamente.

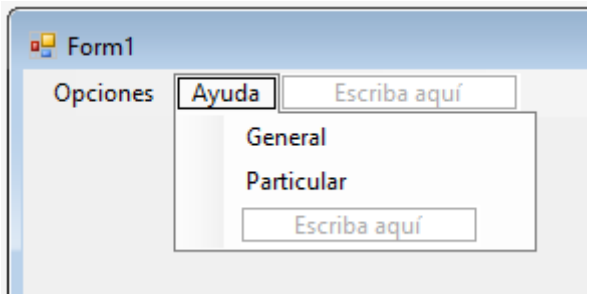
Continuando con el menú vamos podemos realizar submenús.



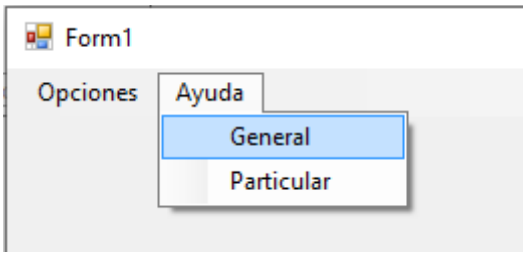
Si a continuación ejecutamos:



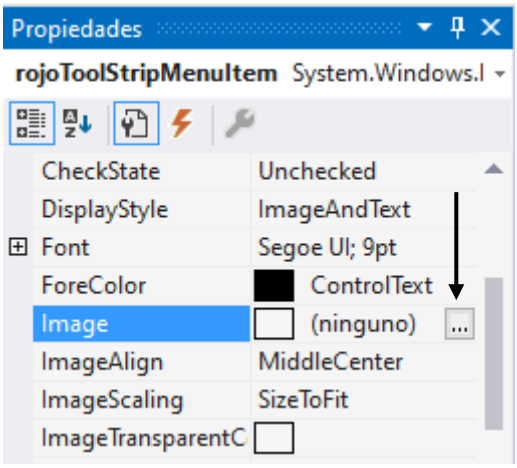
Vamos a seguir modificando el menú.



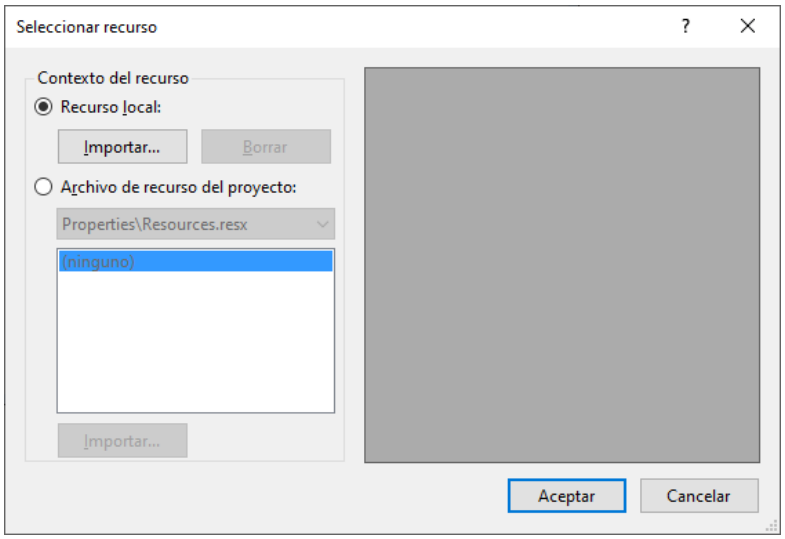
Vamos a ejecutar.



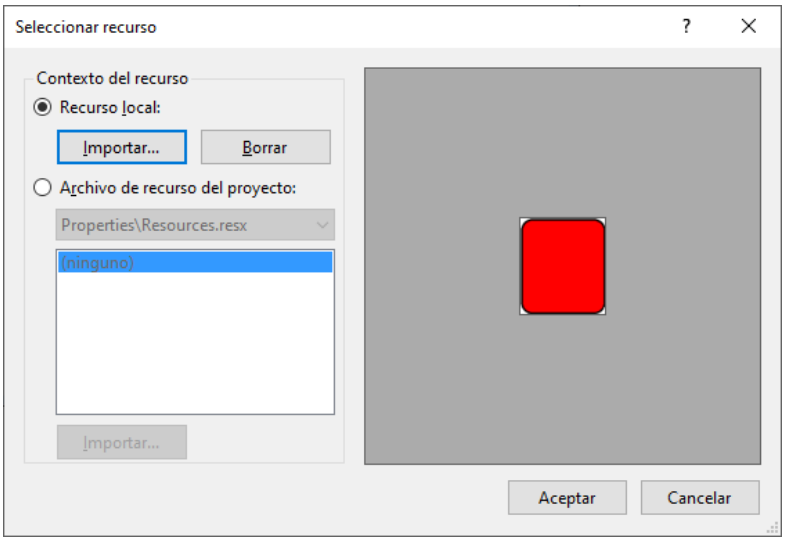
Si queremos insertar imágenes en las opciones del menú una vez seleccionado seleccionaremos la propiedad Image, estas imágenes las debemos tener en nuestro ordenador.



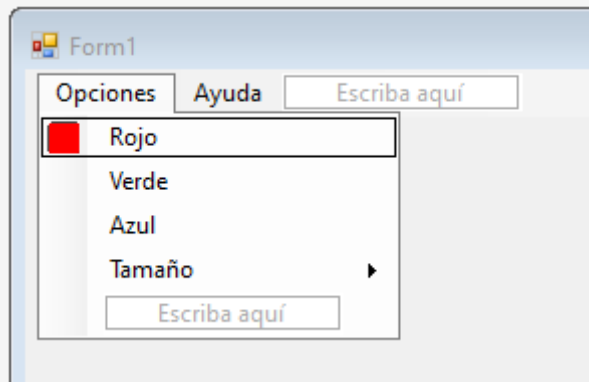
La seleccionamos y cargamos la imagen.



Seleccionamos recurso local seguido del botón Importar…

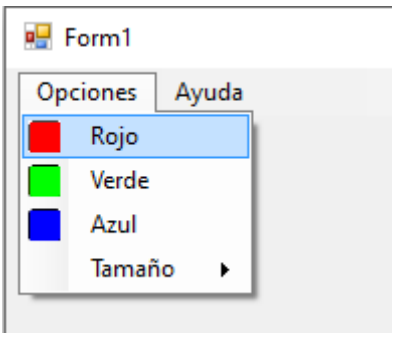


Una vez seleccionada le damos al botón Aceptar.

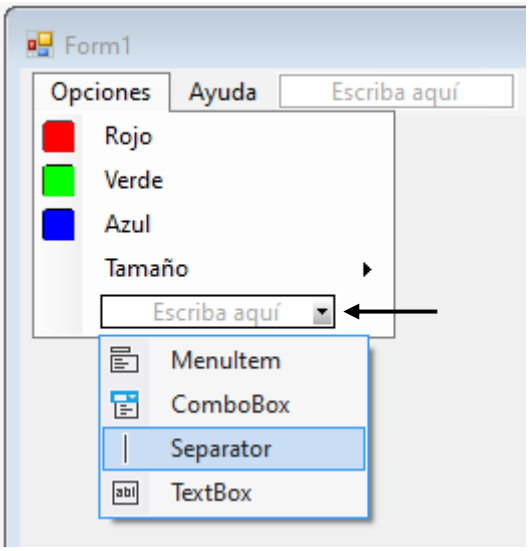


Lo repetimos con el color verde y azul.

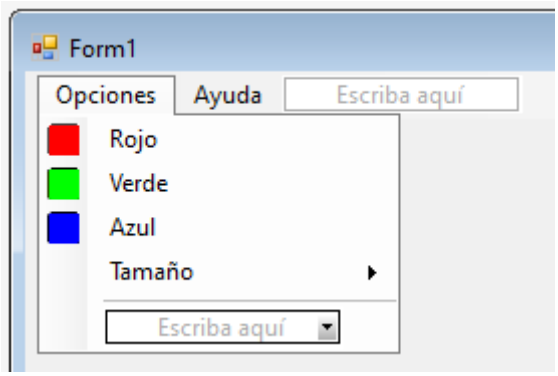
Si ejecutamos este será el resultado:



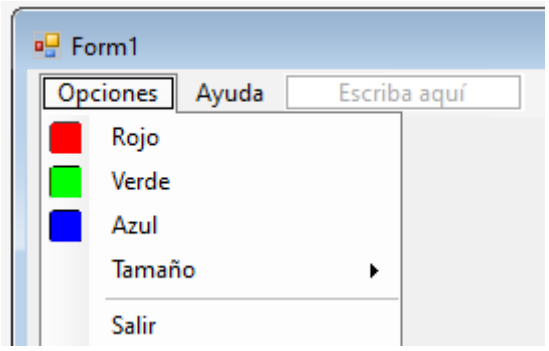
También podemos agregar separadores.



Seleccionamos el triángulo para marcar Separator.

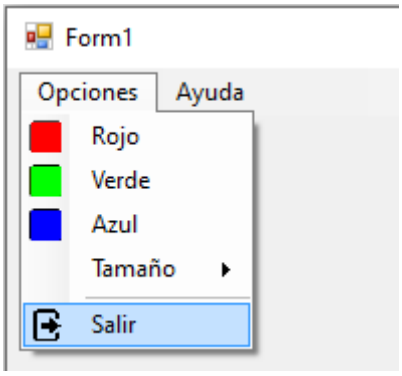


Ahora queremos agregar otra opción 'Salir' que la separará una línea.



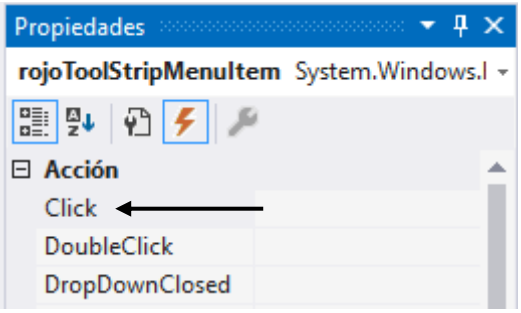
A la opción salir le vamos a agregar una imagen.

Cuando ejecutemos este será el resultado.



Ahora vamos a capturar eventos.

En el apartado de eventos:



Seleccionaremos el evento Click.

```
1 referencia
private void rojoToolStripMenuItem_Click(object sender, EventArgs e)
{
    BackColor = Color.Red;
}
```

Lo repetimos para el color verde y azul.

```
1 referencia
private void verdeToolStripMenuItem_Click(object sender, EventArgs e)
{
    BackColor = Color.Green;
}
```

```
1 referencia
private void azulToolStripMenuItem_Click(object sender, EventArgs e)
{
    BackColor= Color.Blue;
}
```

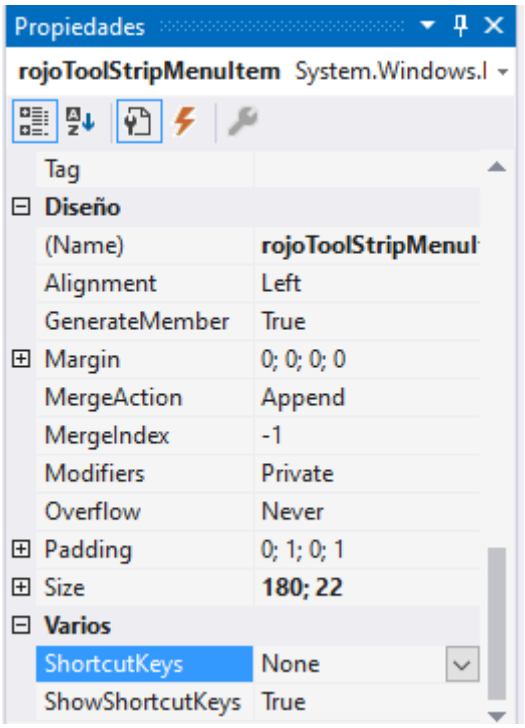
Si ejecutamos podremos cambiar el formulario a sus respectivos colores.

Ahora haremos la opción salir.

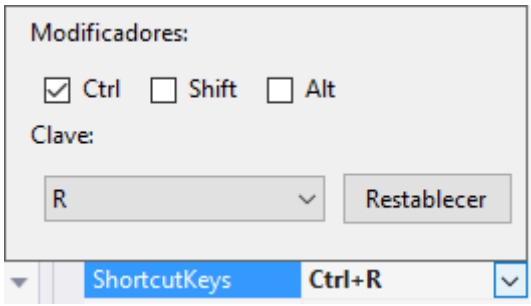
```
1 referencia
private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    Close();
}
```

Al menú también podemos agregar teclas rápidas sin la necesidad de tener que entrar al menú.

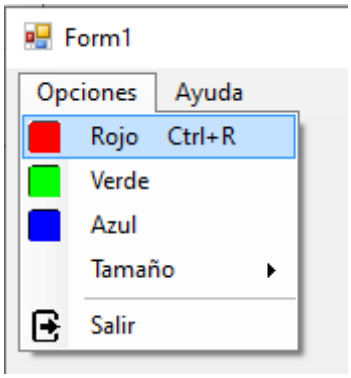
Seleccionaremos la opción Rojo del menú y en propiedades.



La propiedad ShortcutKeys por defecto pone None.

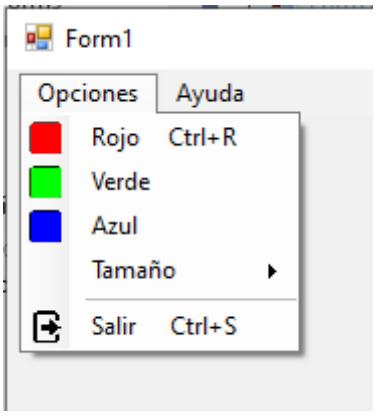


En este caso como la queremos ejecutar con Ctrol + R activamos la casilla Ctrol y seleccionamos la letra R.



Ahora sin necesidad de desplegar el menú para seleccionar la opción Rojo haremos Ctrol + R, obtenemos el mismo resultado.

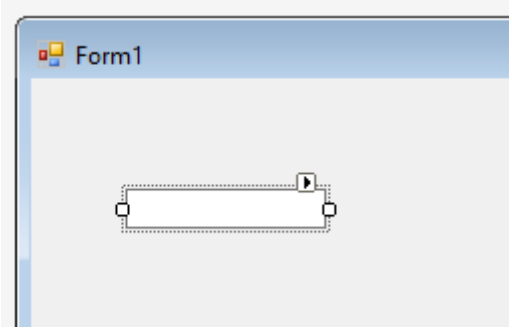
A la opción salir vamos a agregar las teclas rápidas Crol + S.



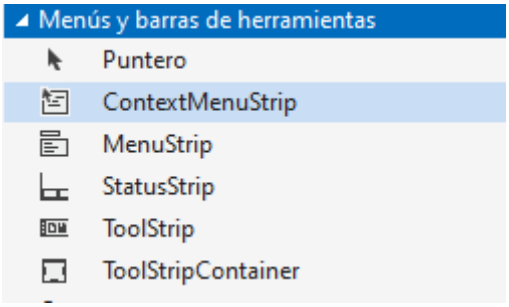
Si necesidad de desplegar el menú con las teclas Ctrol + S cerramos el programa.

Capítulo 119.- Controles visuales ContexMenuStrip y ToolStripMenuItem

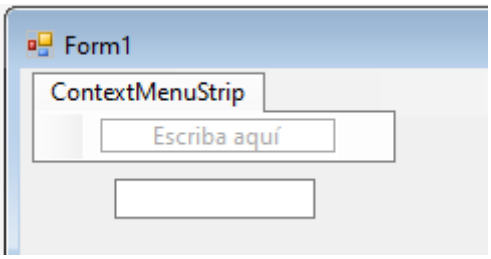
Son menús desplegables cuando seleccionamos con el botón derecho algún control.



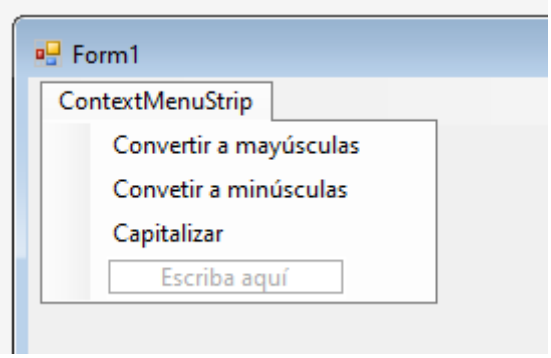
Agregamos un TextBox y con un menú contextual poder cambiar a mayúsculas, minúsculas y capitalizar.



Seleccionamos ContextMenuStrip.

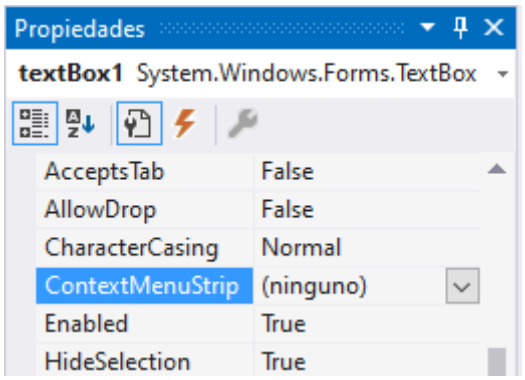


Escribimos:

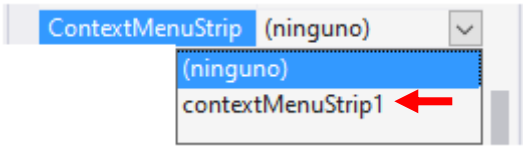


Este menú contextual lo vamos a asociar con el TextBox que agregamos al principio.

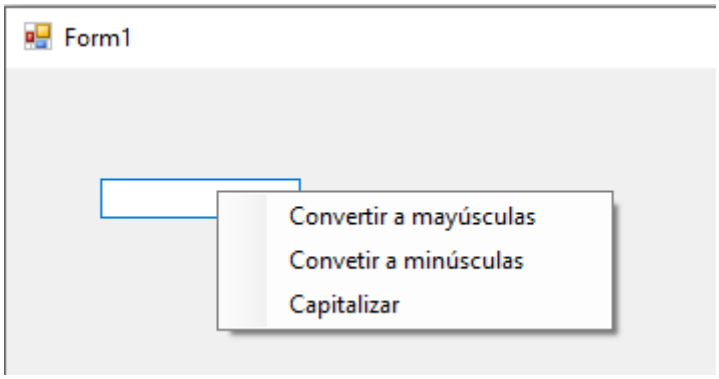
Para esto lo vamos a seleccionar y en propiedades seleccionaremos la opción ContextMenuStrip.



Seleccionamos el contextMenuStrip que hicimos con anterioridad.

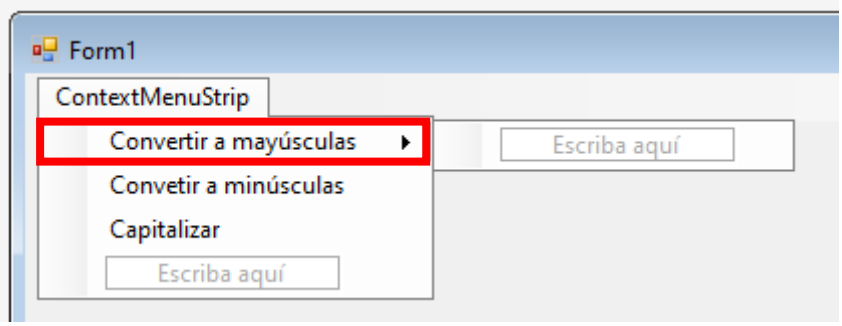


Si seleccionamos con el botón derecho del ratón el TextBox observaremos el siguiente menú:



Ahora solo nos queda la lógica para poder ejecutar estas opciones.

Le damos a la opción ContextMenuStrip1 que se encuentra en la parte inferior, para que aparezca de nuevo las opción del menú y poder seleccionarlas.



La seleccionamos y en el apartado evento seleccionaremos Click.

```
1 referencia
private void convertirAMayúsculasToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text.ToUpper();
}
```

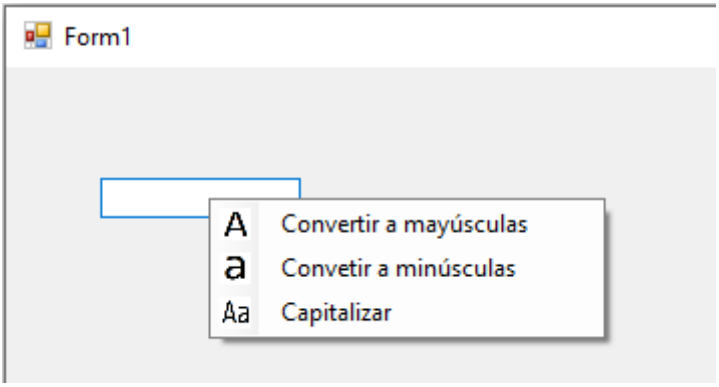
Ahora vamos a seleccionar la opción 'Convertir a minúsculas' y repetiremos seleccionando en eventos la opción Click.

```
1 referencia
private void convetirAMinúsculasToolStripMenuItem_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text.ToLower();
}
```

Ahora seleccionaremos el de Capitalizar.

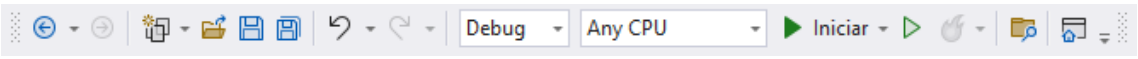
```
1 referencia
private void capitalizarToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Length > 0)
    {
        string cadena = textBox1.Text;
        textBox1.Text = cadena.Substring(0,1).ToUpper();
        textBox1.Text = textBox1.Text + cadena.Substring(1, cadena.Length - 1).ToLower();
    }
}
```

También se pueden agregar imágenes.

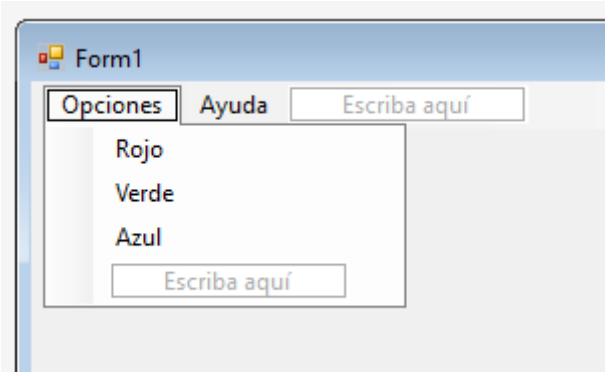


Este menú contextual se pueden utilizar para otro tipo de controles.

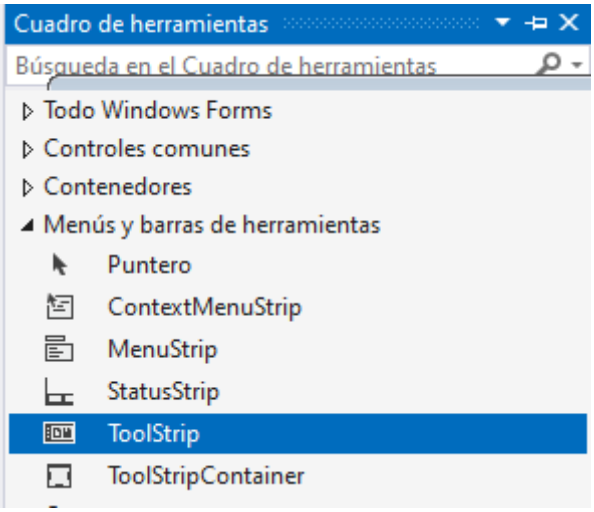
Capítulo 120.- Controles visuales ToolStrip, ToolStripButton, ToolStriptComoBox, ToolStripTextBox, etc.



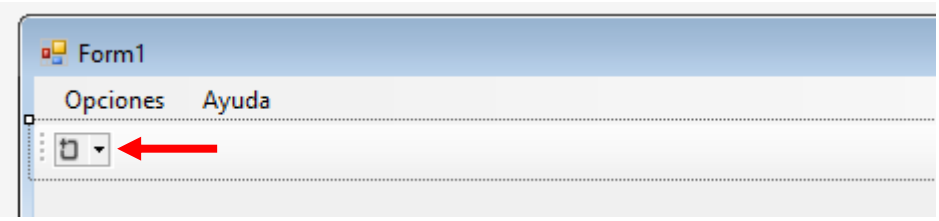
Para este capítulo primero vamos a crear un menú de opciones:



Ahora veremos cómo dispondremos un ToolStrip.

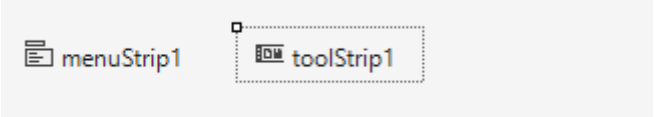


Que lo arrastraremos a la parte superior de formulario por debajo del menú que hemos creado.

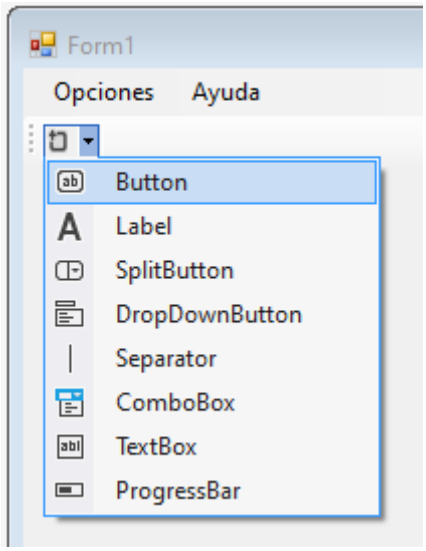


Se pueden colocar varios, uno debajo de otro.

Para seccionarlo si este no se muestra lo podremos hacer desde la parte de abajo.

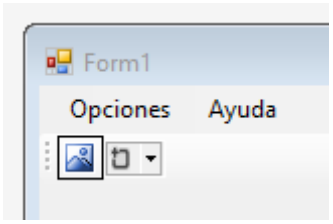


Si desplegamos el objeto marcado con fecha roja podremos observar:

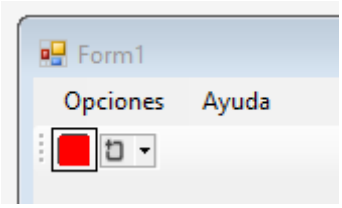


Todos los objetos que podremos agregar en la barra de herramientas.

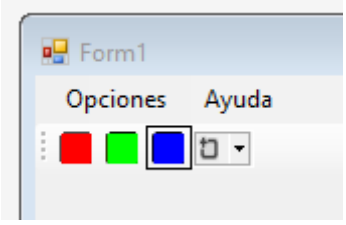
Vamos a agregar un botón.



Estando esta seleccionada en la parte de propiedades seleccionaremos Image para agregar una imagen.



Lo repetimos con un botón verde y otro azul.



Para que estos botones funcionen seleccionaremos el primer botón e iremos a eventos, seleccionaremos Click.

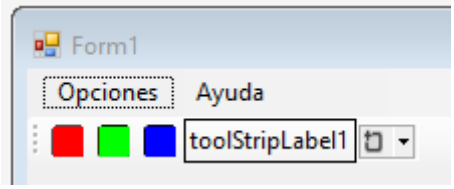
```
1 referencia
private void toolStripButton1_Click(object sender, EventArgs e)
{
    BackColor = Color.Red;
}
```

Que repetiremos con los botones verde y azul.

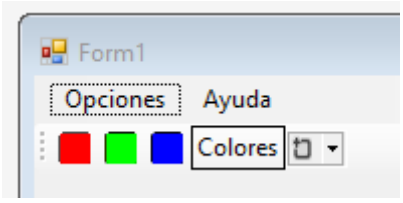
```
1 referencia
private void toolStripButton2_Click(object sender, EventArgs e)
{
    BackColor = Color.Green;
}

1 referencia
private void toolStripButton3_Click(object sender, EventArgs e)
{
    BackColor = Color.Blue;
}
```

Ahora cuando ejecutemos veremos que al presionar los botones el formulario cambia de color.

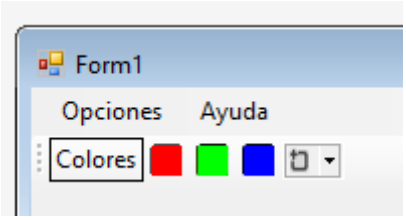
A continuación vamos a agregar un Label.



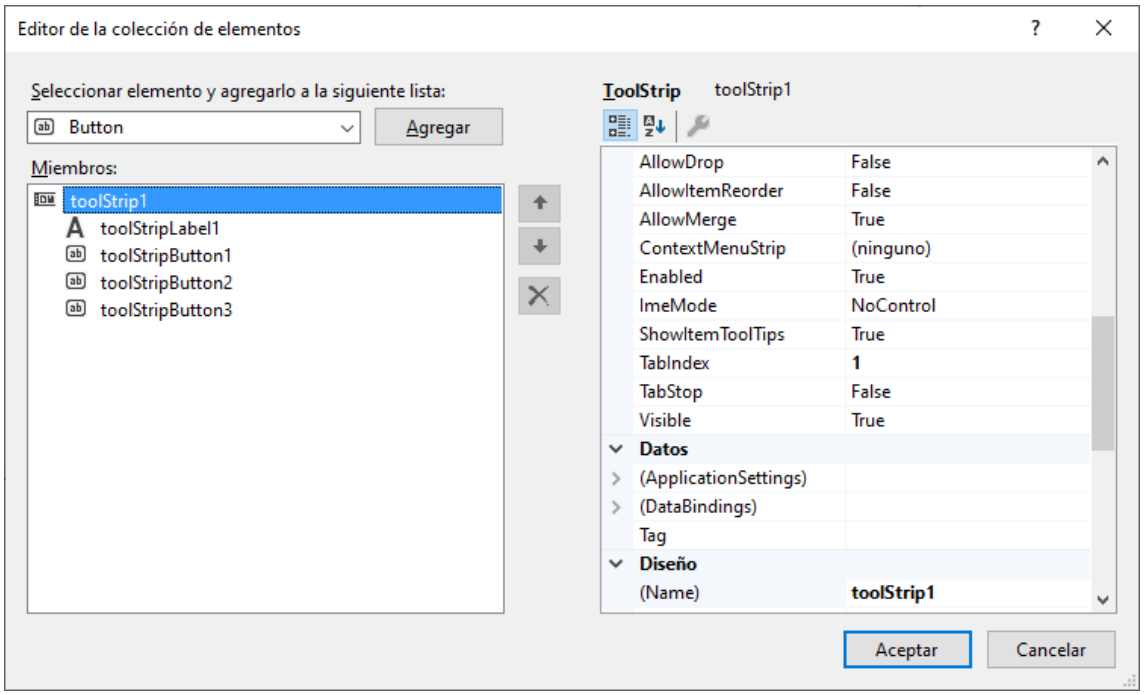
En propiedades Text le pondremos el nombre de Colores.



Si queremos desplazar la etiqueta Colores al principio de la barra de herramientas con seleccionarlo y manteniéndolo presionado lo arrastramos hacia la izquierda.

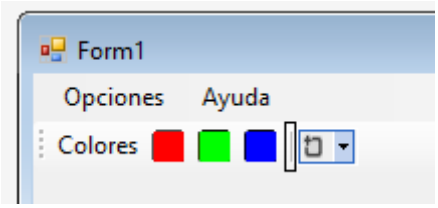


Otra forma de modificar la posición de los objetos es seleccionando la barra de herramientas y en propiedades Items.

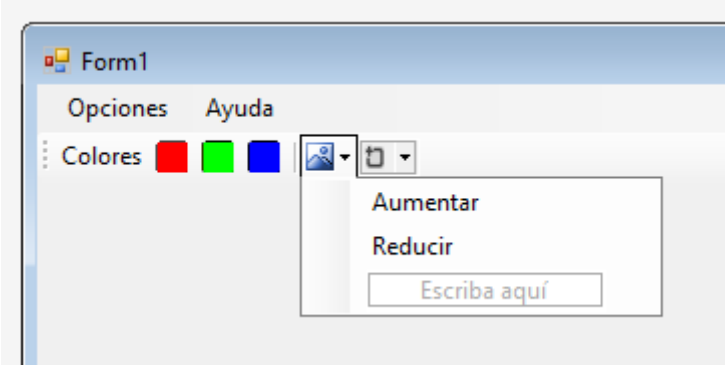


Podremos modificar el orden, además de poderlos eliminar.

A continuación vamos a agregar un separador.

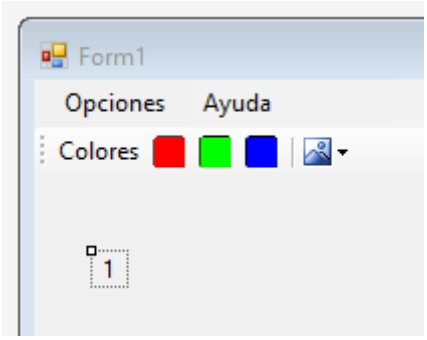


A continuación vamos a agregar un DropDownButton.



Agregamos las opciones Aumentar y reducir.

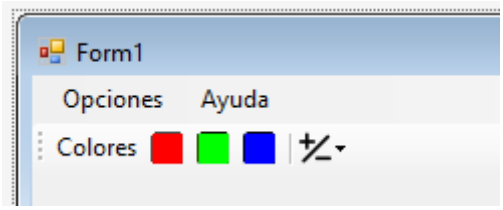
Para ver su funcionamiento vamos a agregar un Label en el formulario con el valor 1.



Ahora vamos a programar el Aumentar y reducir.

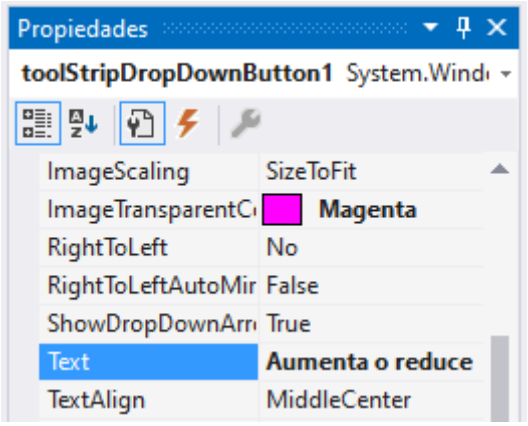
```
1 referencia
private void aumentarToolStripMenuItem_Click(object sender, EventArgs e)
{
    int valor = int.Parse(label1.Text);
    valor++;
    label1.Text = valor.ToString();
}

1 referencia
private void reducirToolStripMenuItem_Click(object sender, EventArgs e)
{
    int valor = int.Parse(label1.Text);
    valor--;
    label1.Text = valor.ToString();
}
```

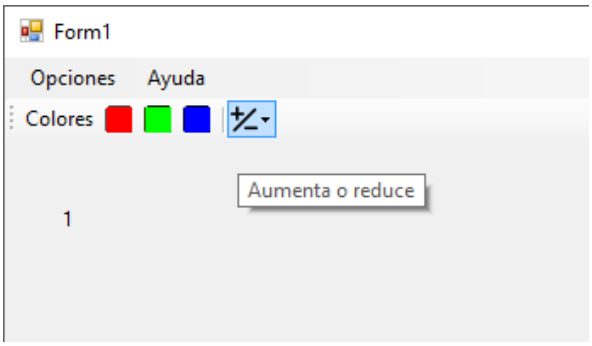
Ahora queremos agregar la imagen a +/-.



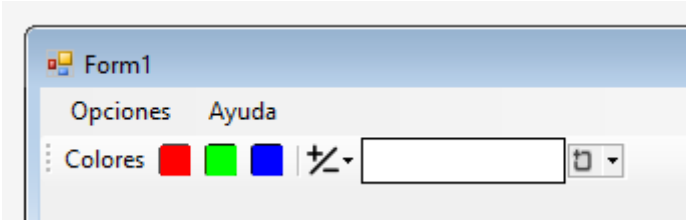
Si queremos que aparezca un texto dando información cuando nos colocamos con el mouse encima.

En el apartado de propiedades seleccionaremos Text.

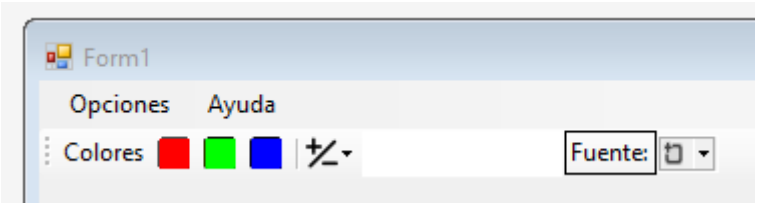




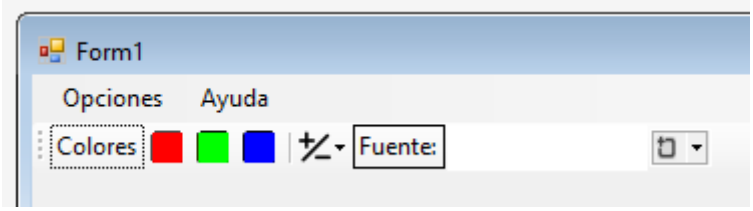
A continuación vamos a agregar un TextBox.



A continuación vamos a agregar un label con el texto Fuente.

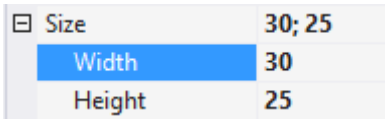


Ahora queremos mover el label que esté antes que el TextBox.

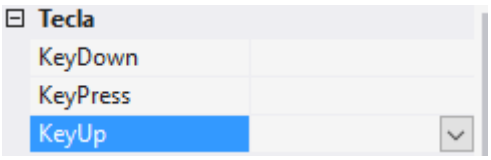


Seleccionamos el TextBox y en la propiedad Text escribimos un 12.

Y el ancho lo pondremos a 30.



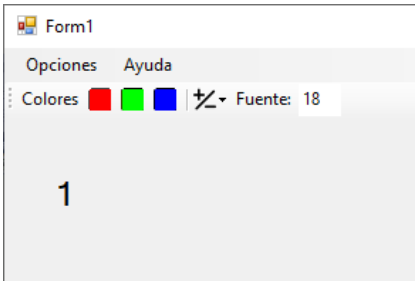
Ahora queremos que cuando seleccionemos la fuente presionando y levantando el dedo del botón que modifique el tamaño de la fuente.



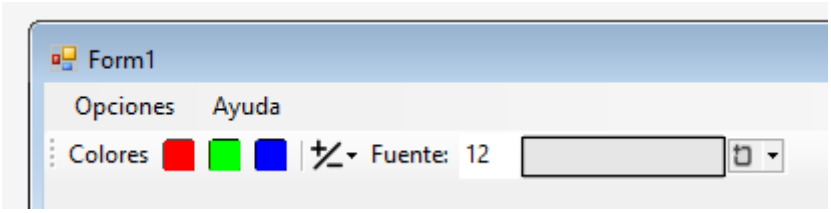
Seleccionaremos el evento KeyUp.

```
1 referencia
private void toolStripTextBox1_KeyUp(object sender, KeyEventArgs e)
{
    label1.Font = new Font(label1.Font.FontFamily, int.Parse(toolStripTextBox1.Text));
}
```

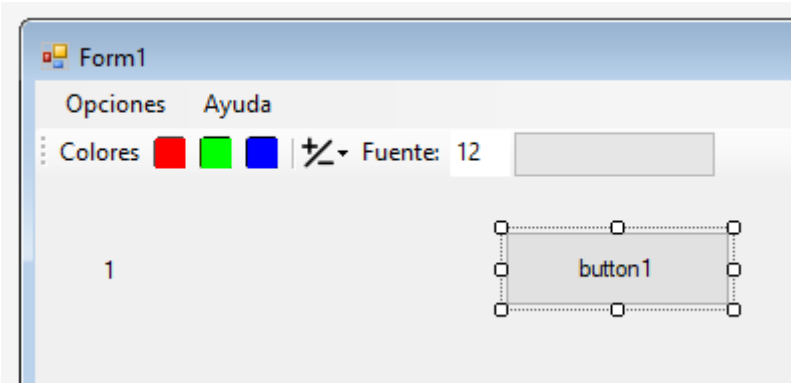
Vamos a cambiar el tamaño de la fuente.



Vamos a agregar un ProgressBar.



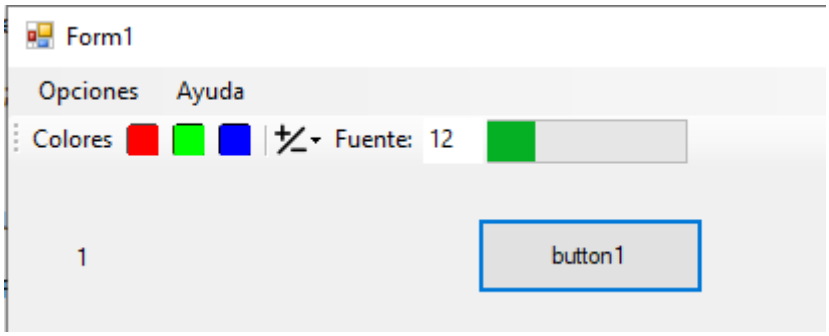
Para ver su funcionamiento vamos a agregar un botón a nuestro formulario.



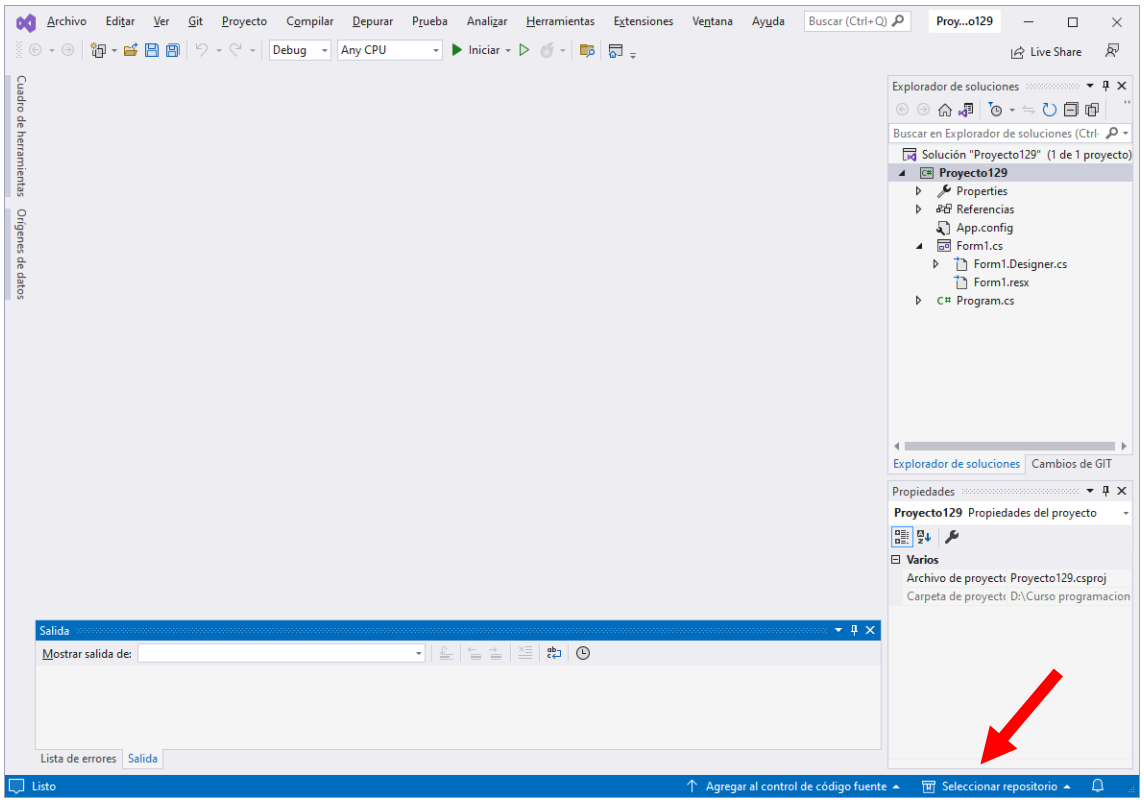
Vamos a programar el evento Click del botón.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    toolStripProgressBar1.Value = toolStripProgressBar1.Value + 5;
}
```

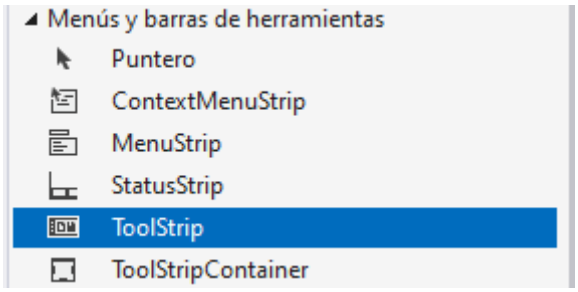
Vamos a ejecutar y presionamos el botón varias veces.



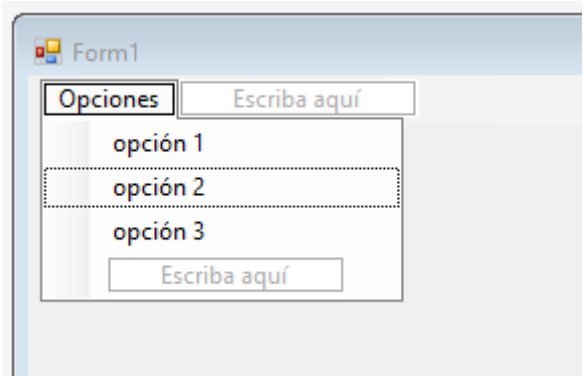
Capítulo 121.- Control visual StatusStrip (Windows Forms)



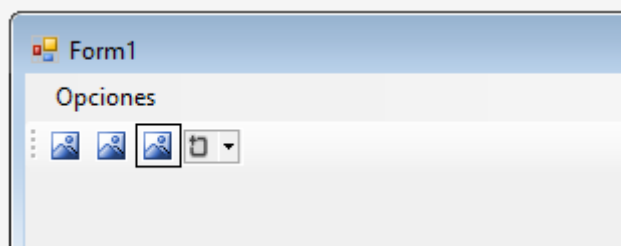
Ahora vamos a trabajar la barra de estado.



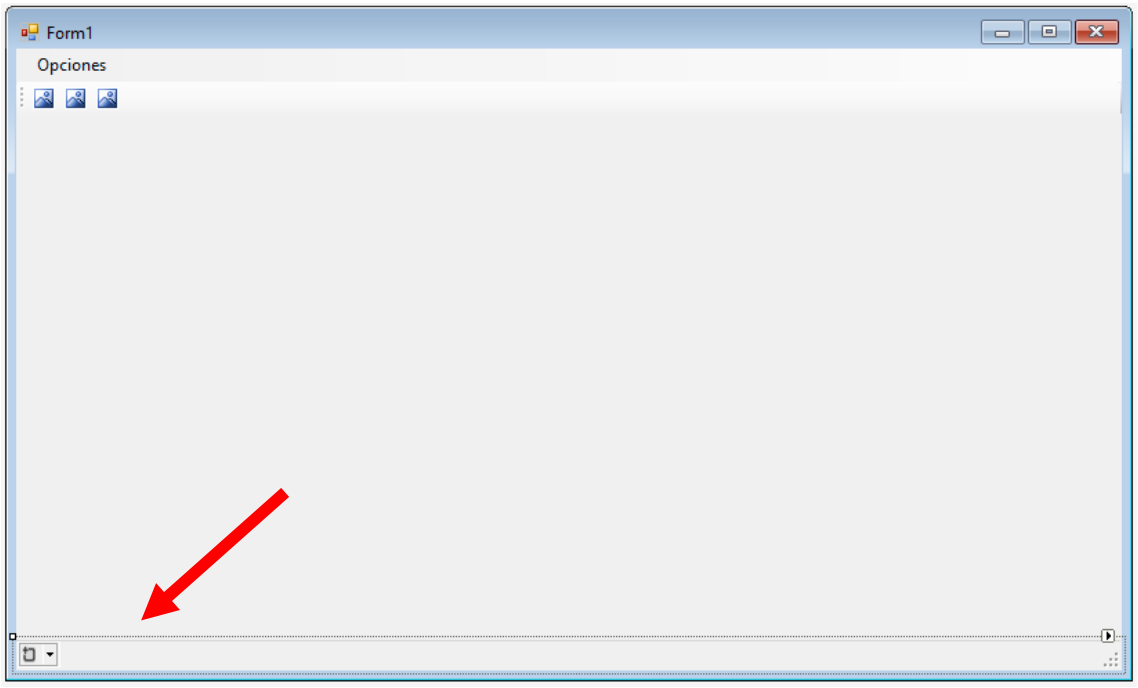
Para entender bien este capítulo primero vamos a crear un menú.



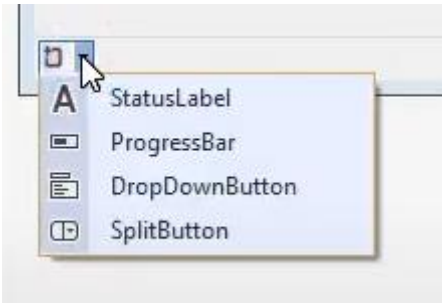
A continuación en la barra de herramientas vamos a crear 3 botones.



Ahora vamos a agregar el StatusStrip en la parte inferior del formulario.



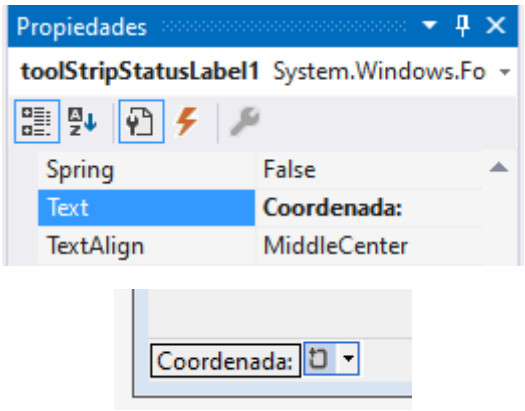
Estos son los objetos que podemos disponer.



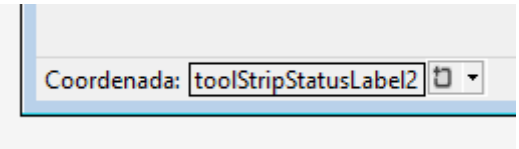
Vamos a agregar un StatusLabel.

Queremos visualizar las coordenadas x e y de nuestro puntero del ratón cuando lo movemos sobre el formulario.

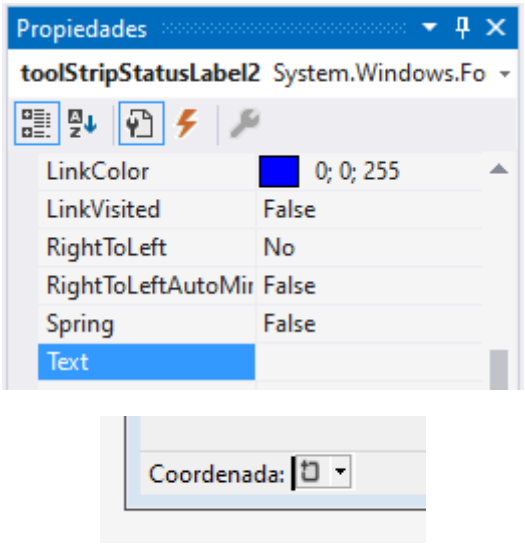
A esta Label como propiedad de Text pondremos Coordenada:



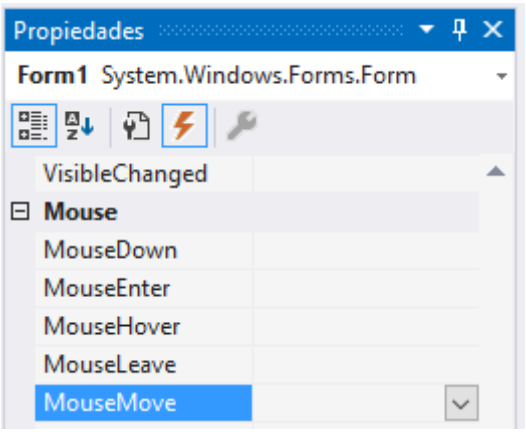
Vamos a agregar un segundo label.



En propiedades Text borramos el texto.



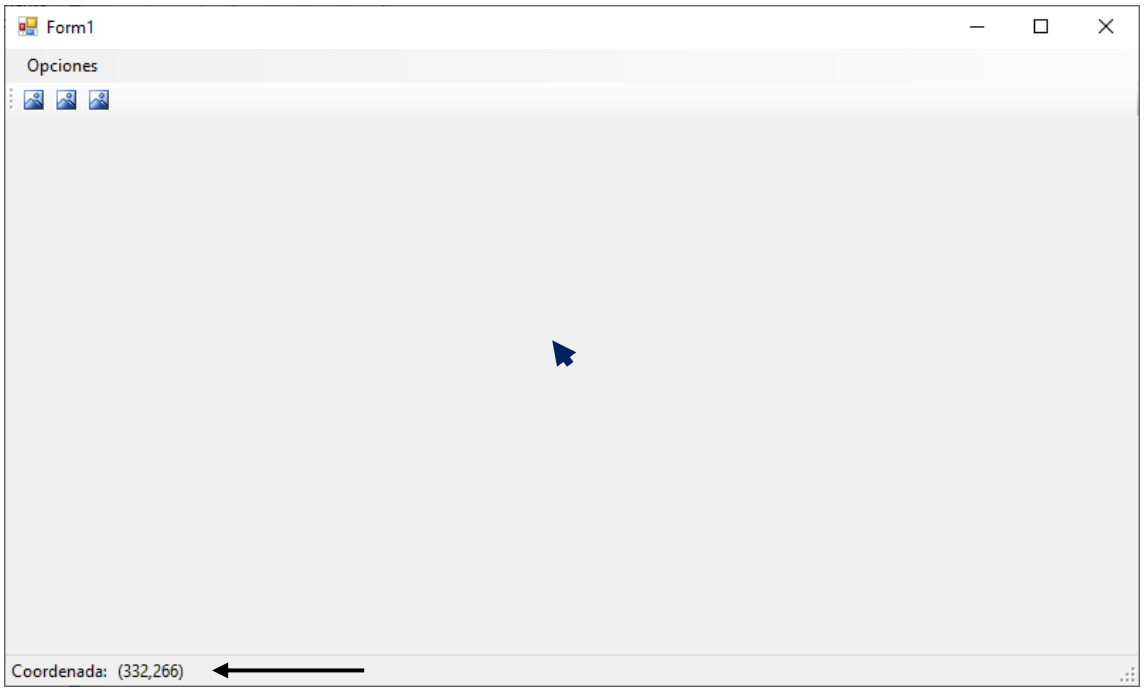
A continuación vamos a seleccionar el formulario, porque el mover el mouse en el es un evento del formulario.



Seleccionamos el evento MouseMove.

Este será el código:

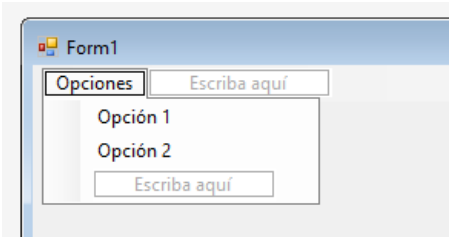
```
1 referencia
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    toolStripStatusLabel2.Text = "("+e.X+","+e.Y+")";
}
```



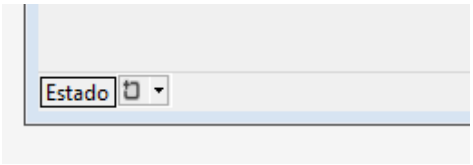
En la barra de estado muestra las coordenadas x e y de la posición del mouse en el formulario.

Capítulo 122.- Control Visual Container ToolStrip (Windows Forms)

Para realizar este capítulo primero vamos a realizar el MenuStrip.

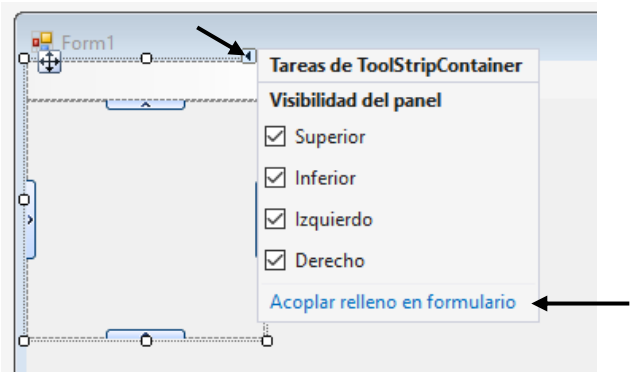


A continuación el StatusStrip.

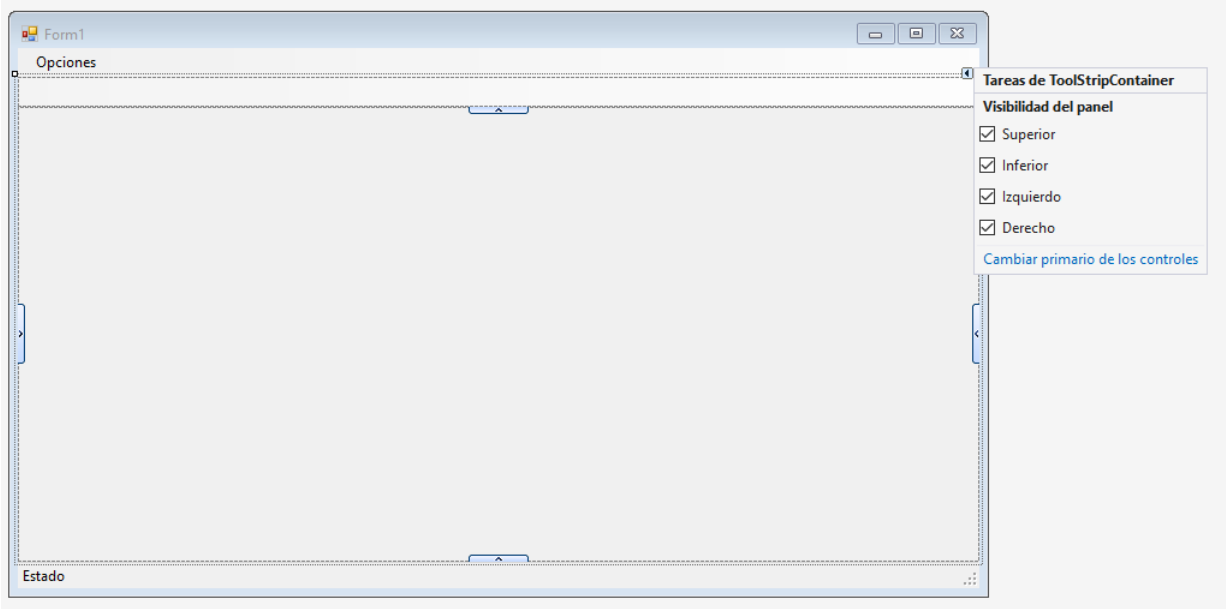


También vimos ToolStrip, pero ahora vamos a ver ToolStripContainer.

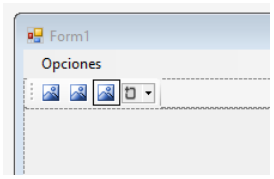
Haremos doble click en él.



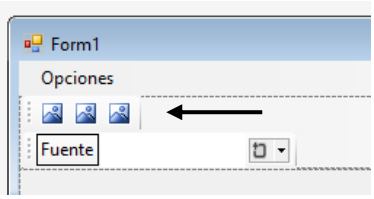
Le damos a acoplar el relleno del formulario.



Ahora vamos a añadir ToolStrip a la parte superior y agregaremos 3 botones.



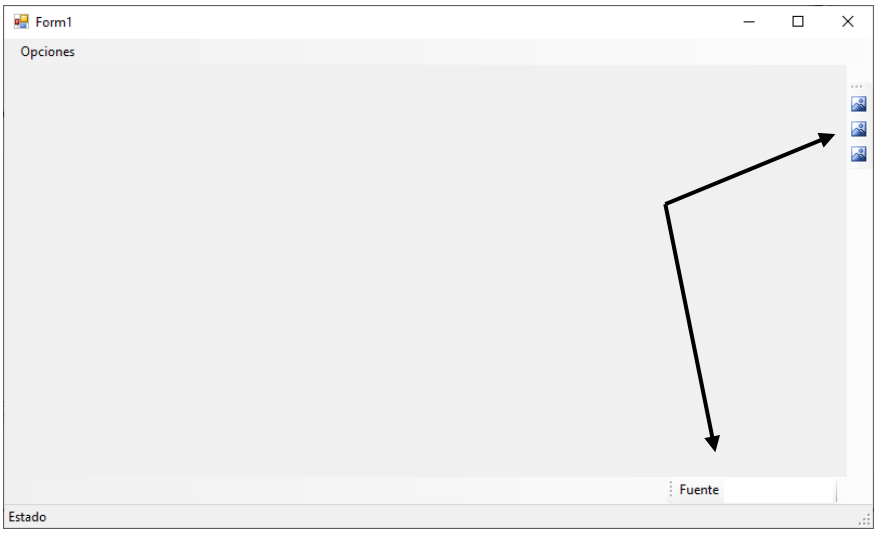
Ahora vamos a agregar un segundo ToolStrip al lado del primero, con un Label y un TextBox.



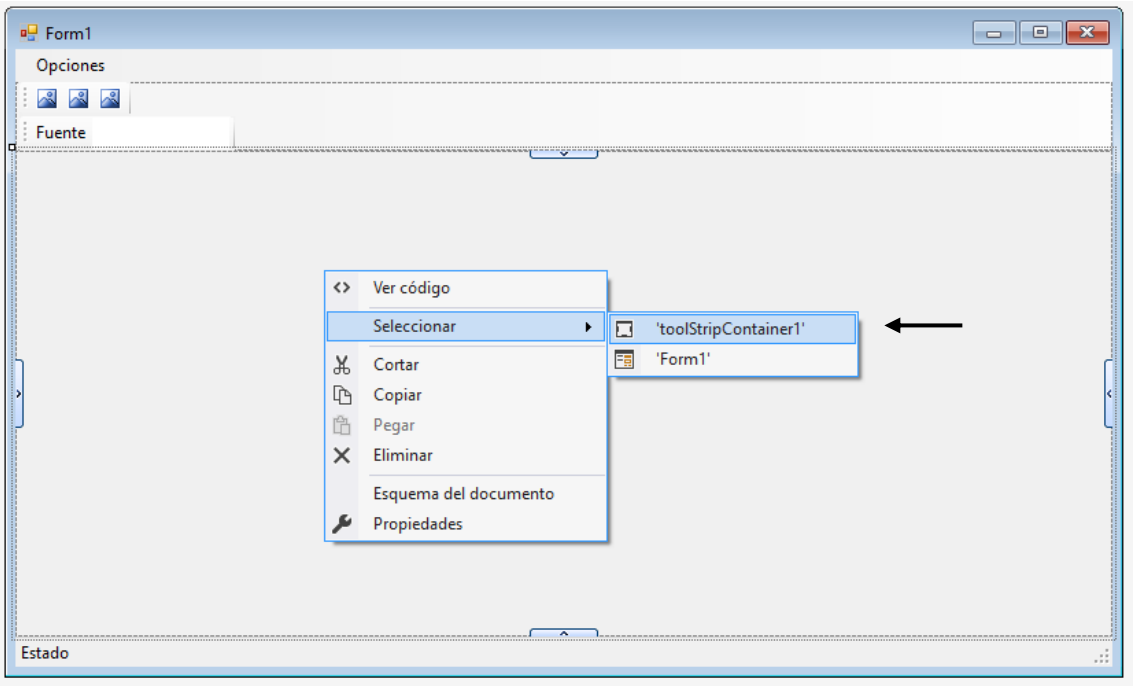
Este se situará en la parte inferior.

La ventaja de este objeto es que en tiempo de ejecución puedo desplazar los ToolStrip a cualquier lado de la ventana.

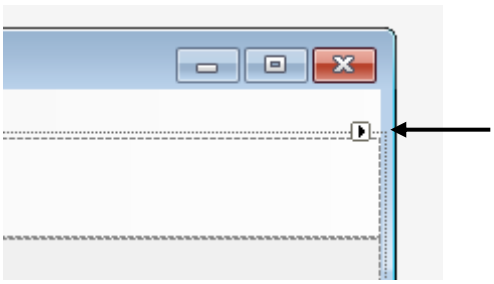
En tiempo de ejecución estas barras de herramientas las puedo desplazar a los 4 lados.



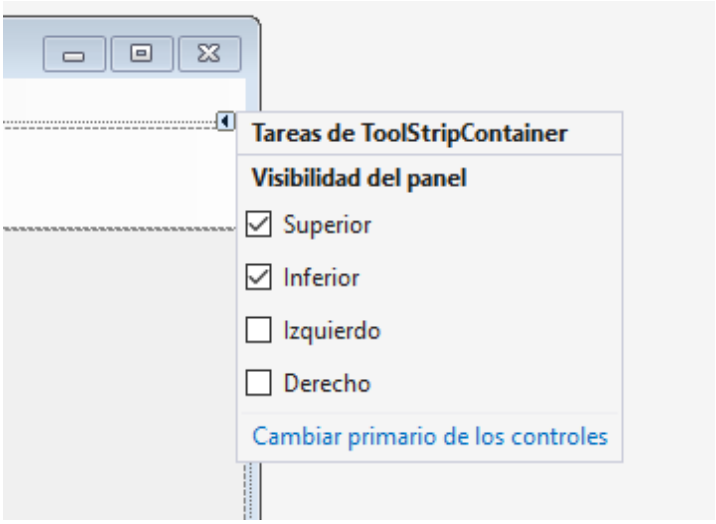
Si queremos liminar a las zonas donde los queremos mover dentro de ContainerToolStrip seleccionaremos con el botón derecho.



De menú seleccionaremos Seleccionar y de este toolStripContainer1.

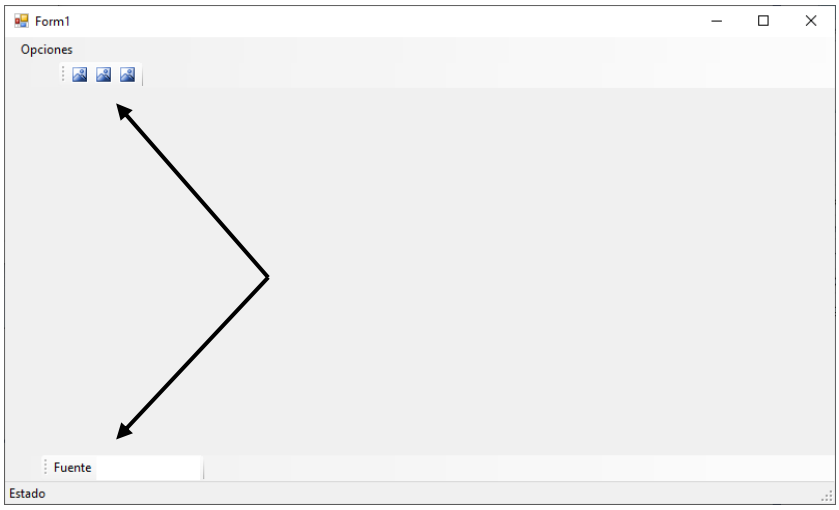


En la parte superior derecha observamos una flechita, la seleccionamos.

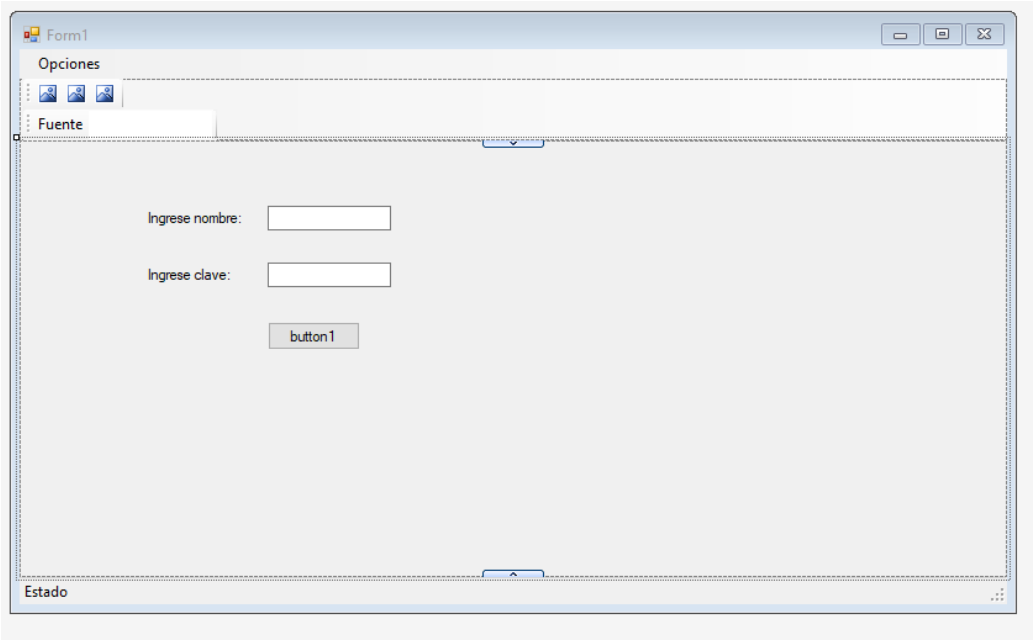


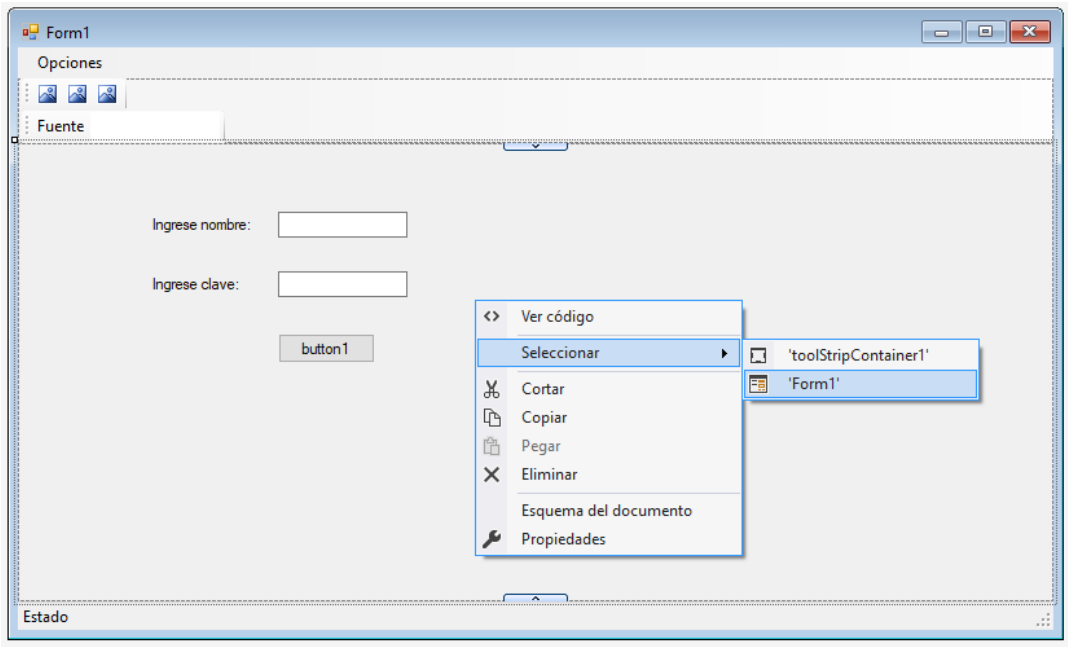
Desmarcamos Izquierdo y Derecho.

Ahor cuando ejecutemos solo podrán situarse en la parte Superior e inferior y no dejará colocarlos ni a izquierda ni a derecha.

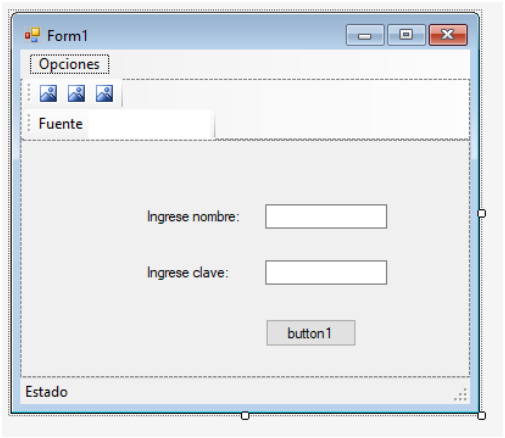


Para concluir este capítulo vamos a agregar los siguientes objetos:





Si queremos seleccionar el formulario para modificar su tamaño con el botón derecho del mouse elegiremos Seleccionar y de este Form.

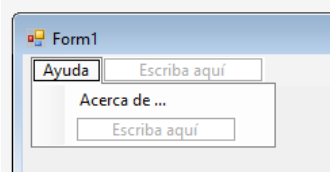


Podremos modificar el tamaño del formulario.

Capítulo 123.- Aplicación de varios Form (Windows Forms)

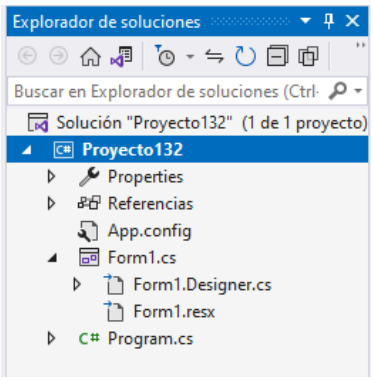
En este capitulo vamos a ver como realizar una aplicación con varios formularios.

En este capítulo vamos a crear un ventana principal una segunda ventana como la que se muestra cuando seleccionamos la opción de ayuda y de este Acerca de.. , hasta que esta segunda ventana no se cierre no tendremos acceso a la ventana principal.

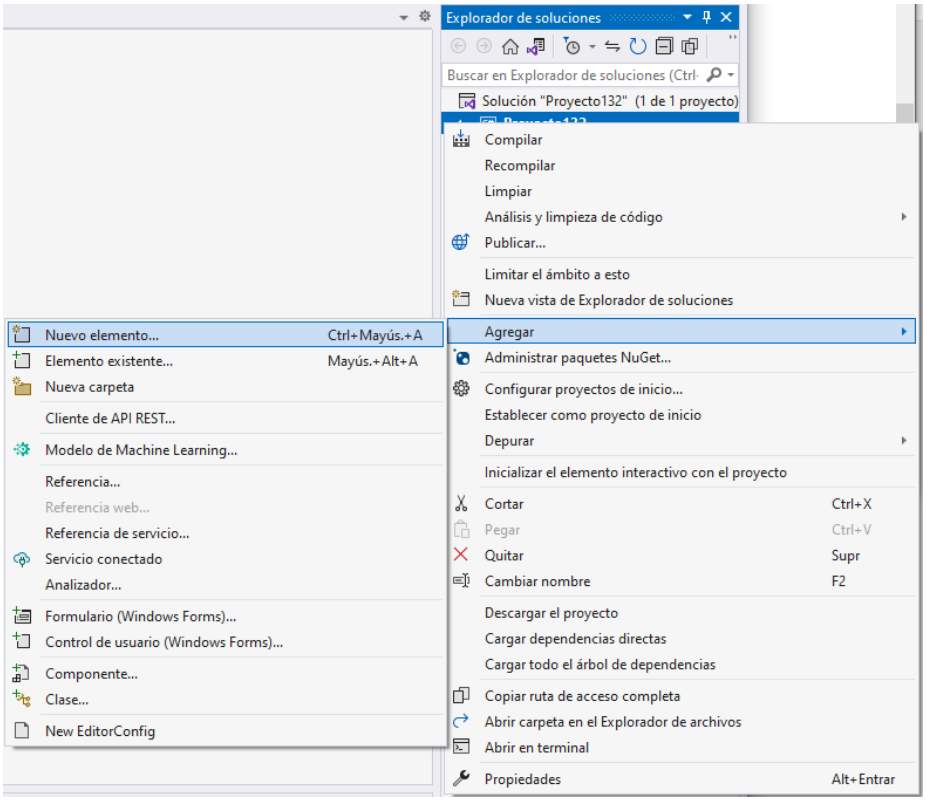


Para crear el segundo formulario.

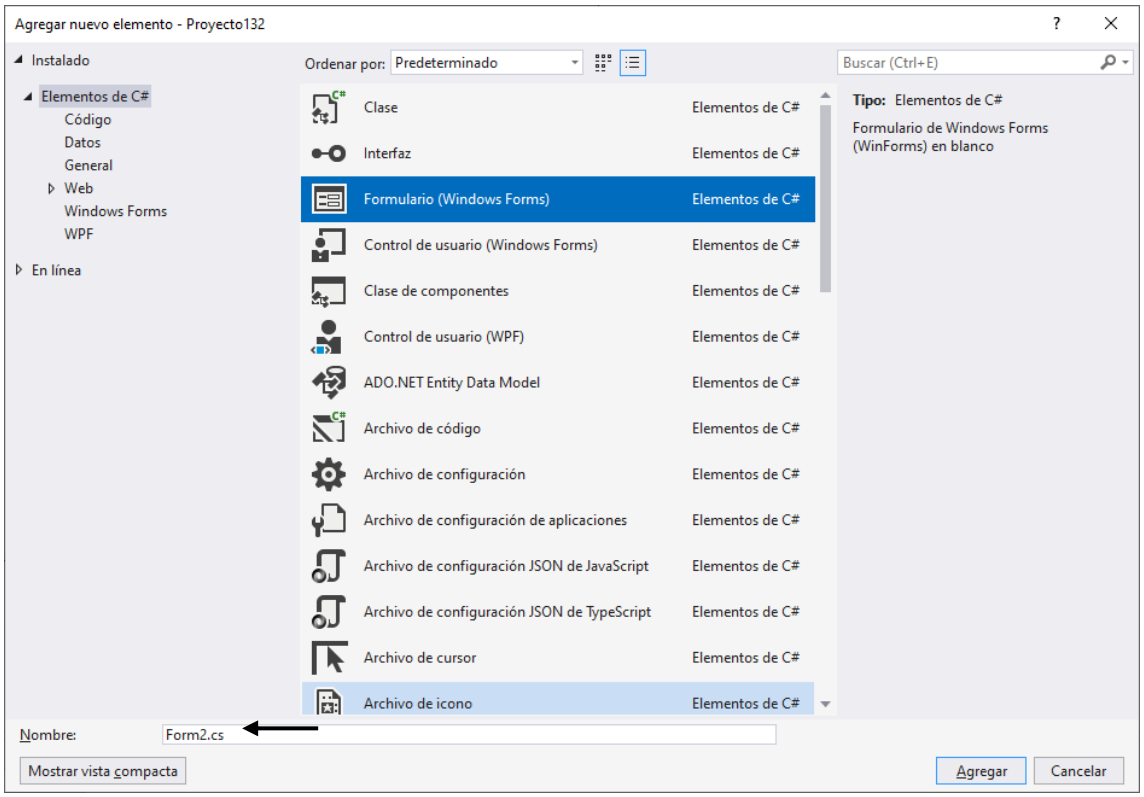
Sobre el Explorador de soluciones seleccionaremos el nombre del proyecto con el botón derecho del mouse.



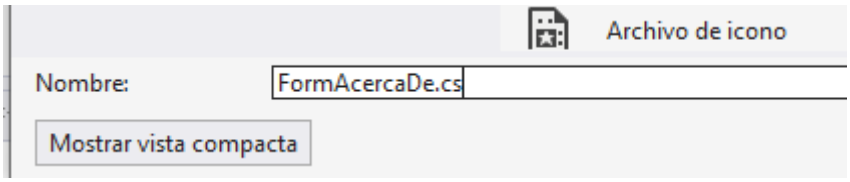
Del menú seleccionaremos agregar y de este nuevo elemento



Seleccionaremos Formulario (Windows Forms).



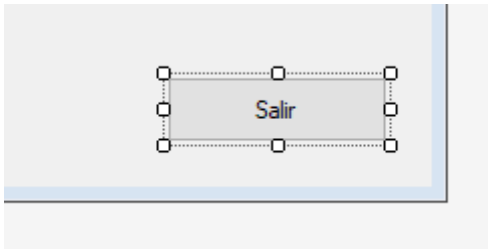
Le cambiamos el nombre que pone por defecto:



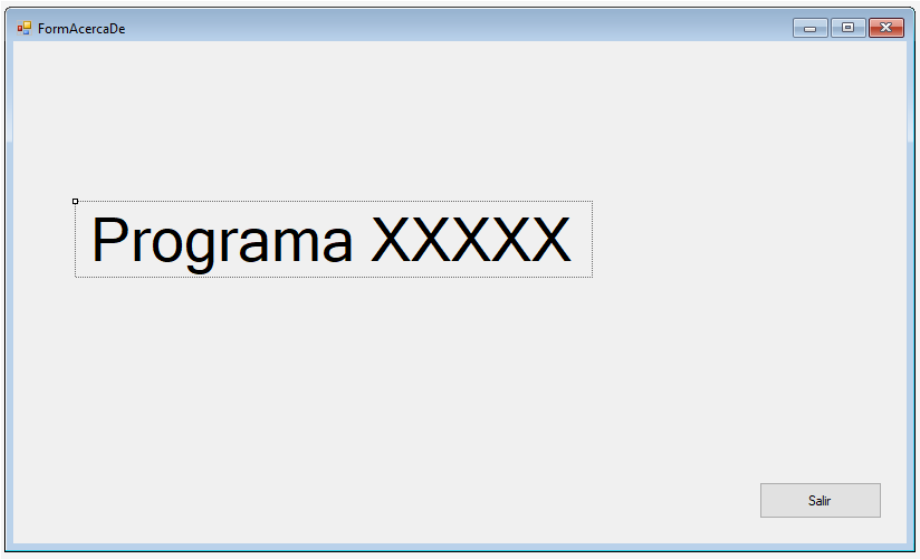
Seguido del botón Agregar.

Ahora ya tenemos dos formularios el Principal y el AcercaDe.

En el Formulario AcercaDe vamos a añadir un botón que diga 'Salir'.



A continuación agregaremos una etiqueta que ponga 'Programa: xxxxx', a la que modificaremos su tamaño.



A continuación vamos a programar el evento Click del botón Salir.

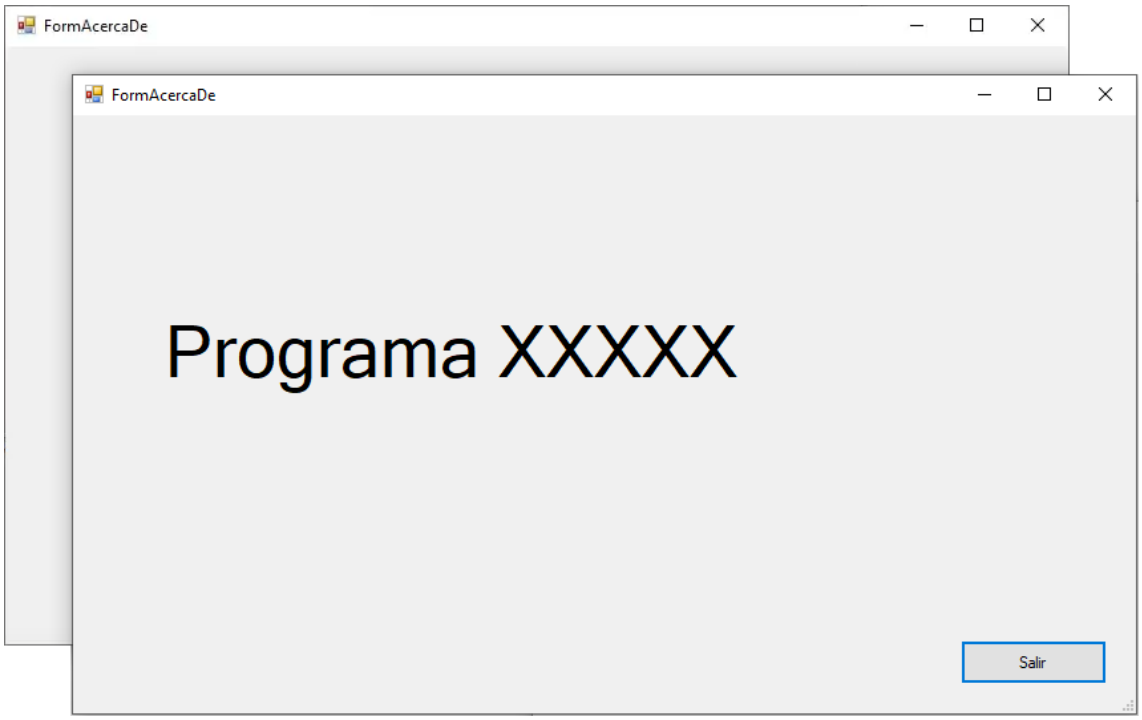
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    Close();
}
```

Ahora desde el formulario Principal vamos a programar el evento Click para la opción Acerca de…

```
1 referencia
private void acerdaDeToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormAcercaDe formulario = new FormAcercaDe();
    formulario.ShowDialog();
}
```

Vamos a Ejecutar:

Cuando seleccionemos la opción Acerca de…



Si intentamos seleccionar el formulario principal que se encuentra detrás no nos va a responder hasta que seleccionemos el botón salir del Formulario AcercaDe.

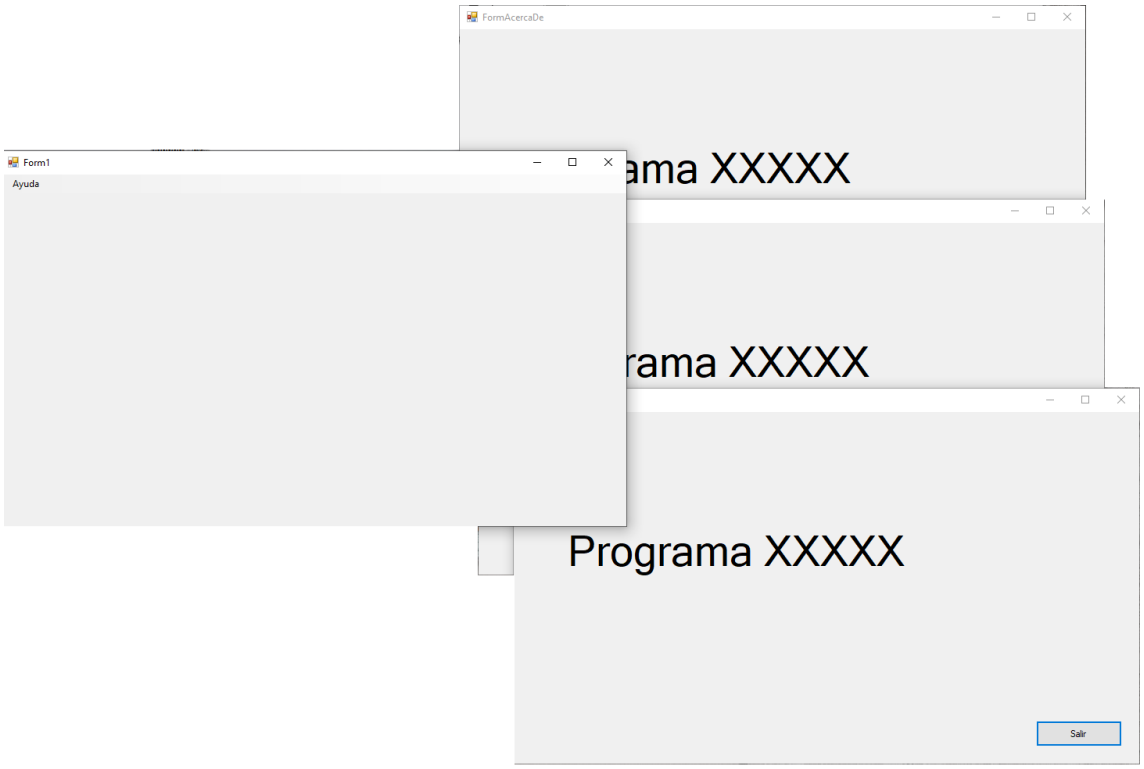
Si modificamos el código del evento AcercaDe, que el lugar de formulario.ShowDialog ponemos formulario.Show:

```
1 referencia
private void acerdaDeToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormAcercaDe formulario = new FormAcercaDe();
    formulario.Show();
}
```

Con esta opción cuando ejecutemos la aplicación y llamemos al formulario Acerca de sin necesidad de cerrar el Formulario AcercaDe podremos tener acceso a al formulario Principal.

Podemos tener actividad tanto en la ventana principal como la emergente.

Tenemos que tener mucho cuidado de que si no cerramos la ventana AcercaDe y de la ventana principal la volvemos a llamar puede pasar lo siguiente:

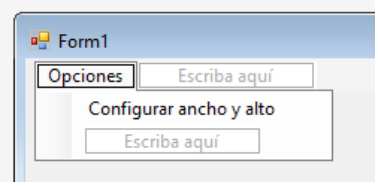

FormAcercaDe
ama XXXXX
Form1
Ayuda
rama XXXXX
Programa XXXXX
Salir

Capítulo 124.- Aplicación con varios Form – Comunicación de datos por propiedades (Windows Forms)

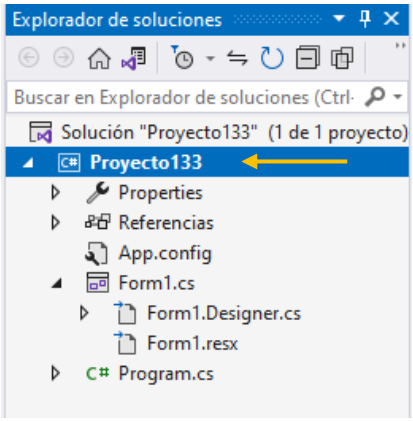
En este capítulo vamos a ver como hacer la comunicación de datos.

Vamos a crear un proyecto este proyecto será un formulario que llamará a un segundo formulario, en este segundo vamos a dar las dimensiones del formulario, para que el formulario principal realice su modificación.

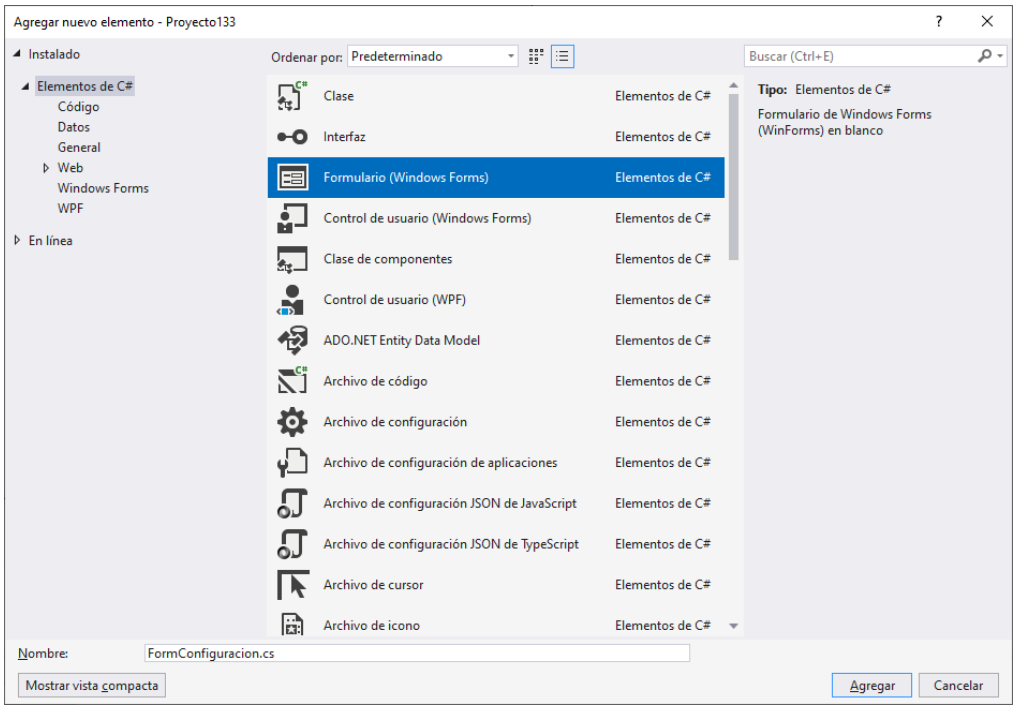
El menú principal confeccionaremos el correspondiente MenuStrip.



Vamos a agregar el segundo formulario, que ya hicimos en el capítulo anterior:



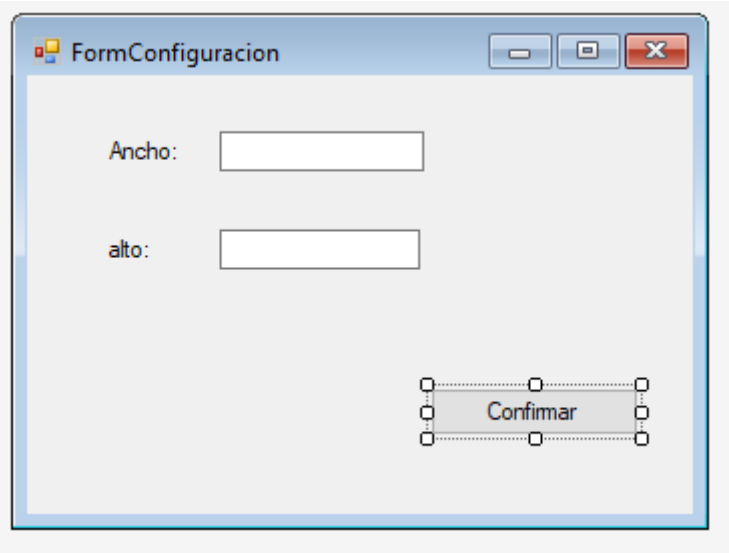
Botón derecho del mouse sobre el nombre del proyecto del menú seleccionaremos Agregar y de este Nuevo elemento…



Seleccionaremos 'Formulario (Windows Forms) y como nombre FormConfiguracion.cs, seguido del botón Agregar.

Ahora ya disponemos de dos formularios.

Este será el diseño del segundo formulario:



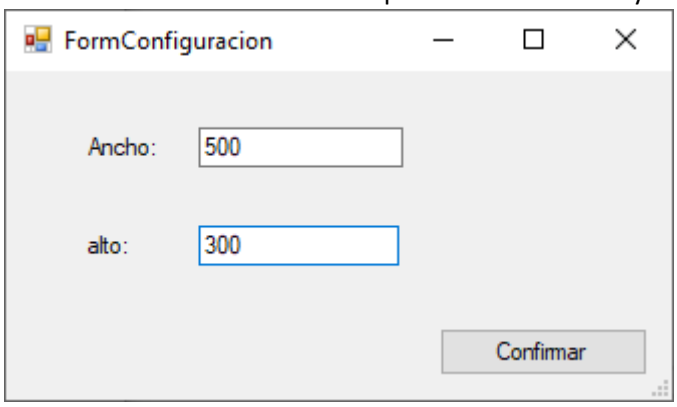
Hemos agregado dos Label con su respectivo, texto, dos TextBox para poder ingresar los nuevos valores y un botón para confirmar.
El código del botón Confirmar será:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    Close();
}
```

Ahora desde el formulario principal cuando hagamos clic sobre 'Configurar Ancho y Alto' con un evento click.

```
1 referencia
private void configurarAnchoYAltoToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormConfiguracion formulario = new FormConfiguracion();
    formulario.ShowDialog();
}
```

La pregunta es una vez hemos introducir los valore para el nuevo ancho y alto:



Como retornará al formulario principal.

```
namespace Proyecto133
{
    4 referencias
    public partial class FormConfiguracion : Form
    {
        0 referencias
        public int Ancho
        {
            get
            {
                return int.Parse(textBox1.Text);
            }
        }

        0 referencias
        public int Alto
        {
            get
            {
                return int.Parse(textBox2.Text);
            }
        }

        1 referencia
        public FormConfiguracion()
        {
            InitializeComponent();
        }

        1 referencia
        private void button1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

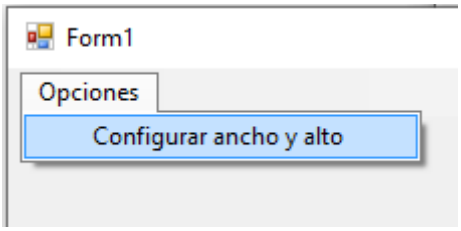
Hemos definido dos propiedades, de este modo tendremos acceso a la información de Ancho y Alto desde el formulario Principal.

Desde el formulario Principal en el evento click de la opción del menú Configurar Ancho y Alto vamos a modificar el código:

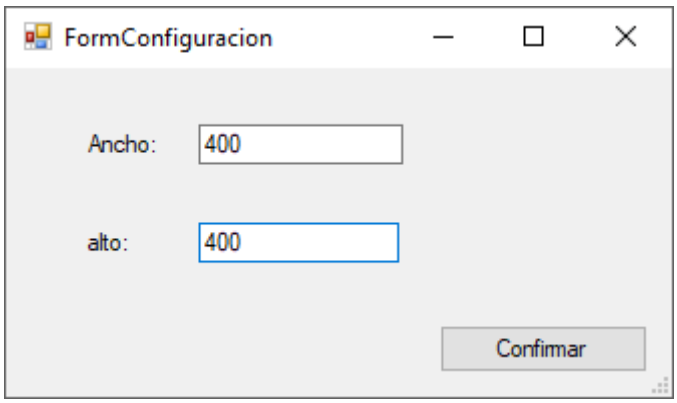
```
1 referencia
private void configurarAnchoYAltoToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormConfiguracion formulario = new FormConfiguracion();
    formulario.ShowDialog();
    Width = formulario.Ancho;
    Height = formulario.Alto;
}
```

Cuando cerremos el formulario de configuración asumirá los valores de ancho y alto.

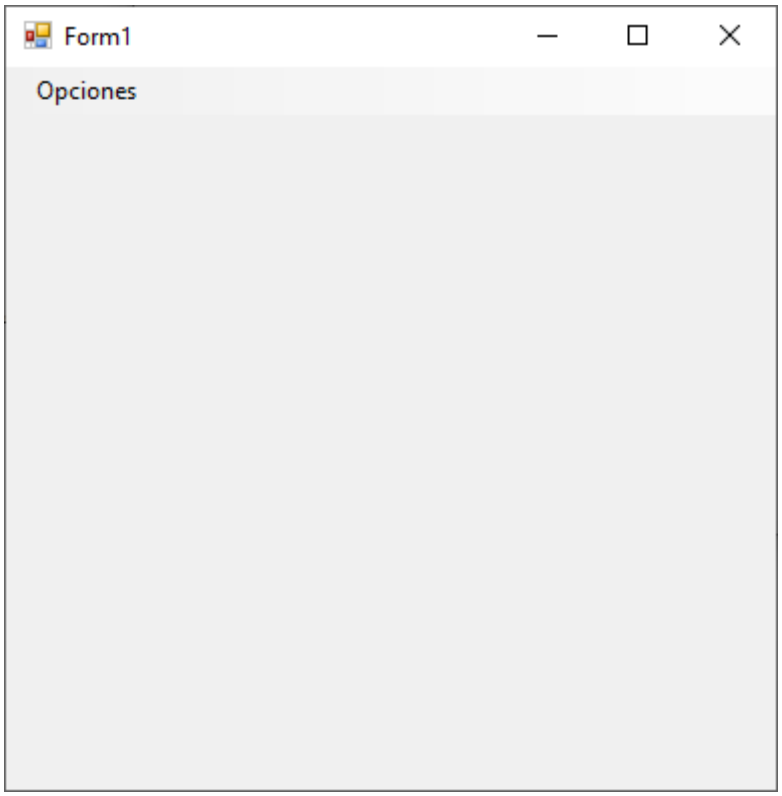
Vamos a ejecutar para modificar las dimensiones de la ventana principal.



Ejecutamos Configurar ancho y alto.

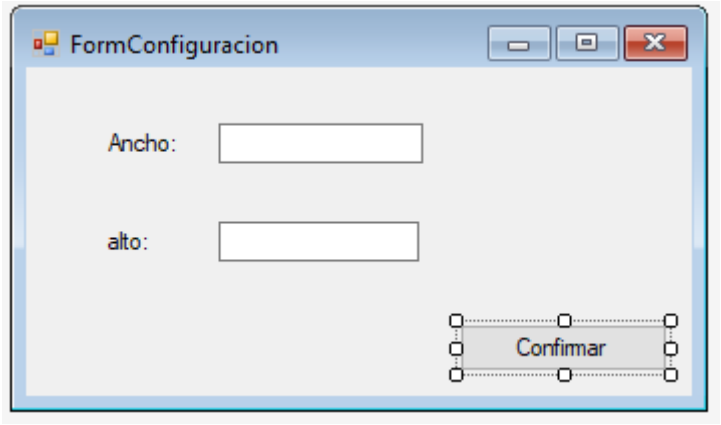


Ponemos las nuevas dimensiones seguido del botón Confirmar.



El formulario principal ya tiene las nuevas dimensiones.

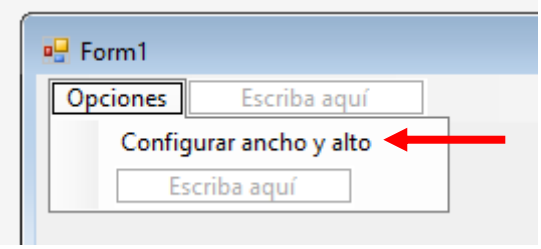
Queremos que cuando carguemos el formulario de configuración nos muestre las dimensiones que tiene, por este motivo además de escritura tiene que ser de lectura.



Vamos a modificar el código del evento click del botón confirmar.

```
1 referencia
public int Ancho
{
    get
    {
        return int.Parse(textBox1.Text);
    }
    set
    {
        textBox1.Text = value.ToString();
    }
}

1 referencia
public int Alto
{
    get
    {
        return int.Parse(textBox2.Text);
    }
    set
    {
        textBox2.Text = value.ToString();
    }
}
```

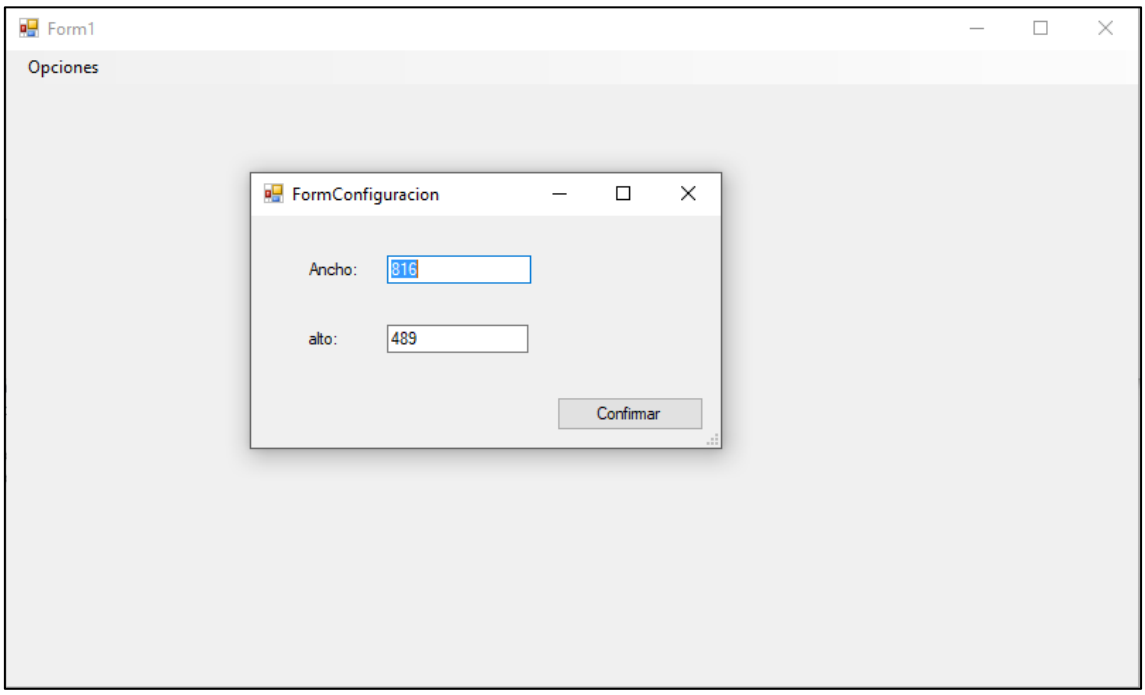
A continuación vamos a modificar el código de la opción Configurar Ancho y Alto.



```
1 referencia
private void configurarAnchoYAltoToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormConfiguracion formulario = new FormConfiguracion();
    formulario.Ancho = Width;
    formulario.Alto = Height;
    formulario.ShowDialog();
    Width = formulario.Ancho;
    Height = formulario.Alto;
}
```

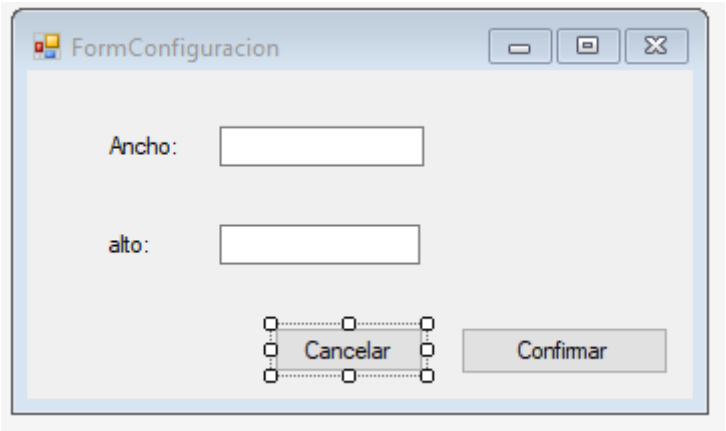
De este modo antes de abrir el formulario de configuración almacena en las propiedades Ancho y alto los valores de ancho y alto que tiene el formulario principal.

Al ejecutar y llamar a la ventana de configuración mostrará las dimensiones que tiene la primera Ventana.



Y tenemos la opción de poder cambiar las dimensiones de ancho y alto.

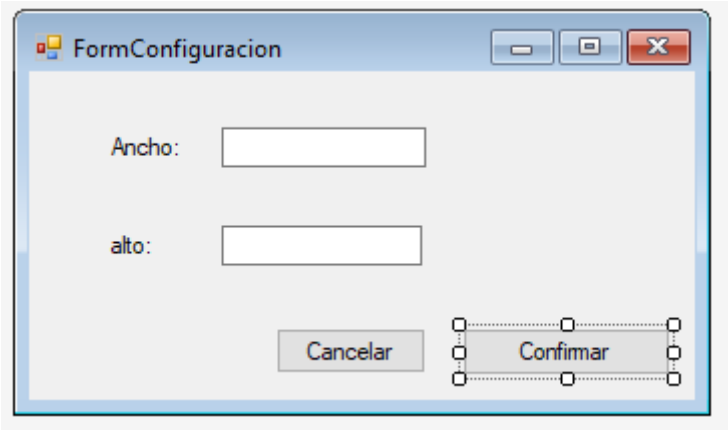
Vamos a modificar el formulario de configuración con un segundo botón llamado Cancelar.



El código del Botón Cancelar con el evento click será el siguiente:

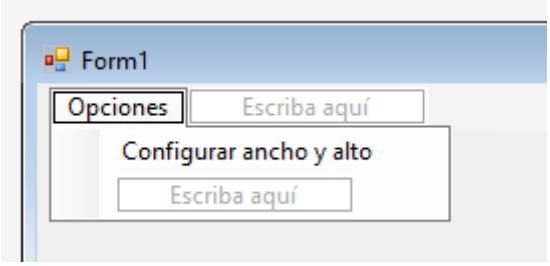
```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.Cancel;
    Close();
}
```

Y al botón Confirmar:




```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.OK;
    Close();
}
```

Vamos a modificar el código de la opción Configurar ancho y alto:




```
1 referencia
private void configurarAnchoYAltoToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormConfiguracion formulario = new FormConfiguracion();
    formulario.Ancho = Width;
    formulario.Alto = Height;
    formulario.ShowDialog();
    if(formulario.DialogResult==DialogResult.OK)
    {
        Width = formulario.Ancho;
        Height = formulario.Alto;
    }
}
```

Controlamos si formulario.DialogResult tiene almacenado DialogResult.OK en caso afirmativo modificará las dimensiones del formulario principal y si no lo cumple porque hemos presionado el botón Cancelar, no se modificará las dimensiones del formulario principal.

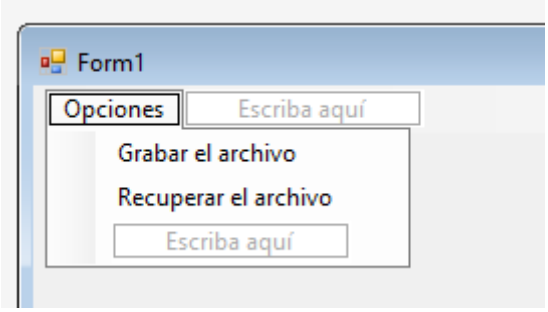
Ahora cuando ejecutemos de nuevo el programa y intentamos modificar las dimensiones del formulario Principal, pero en el último momento nos arrepentimos y le damos al botón Cancelar el formulario Principal no se modificará.

Capítulo 125.- Controles visuales OpenFileDialog, SaveFileDialog, FontDialog y ColorDialog

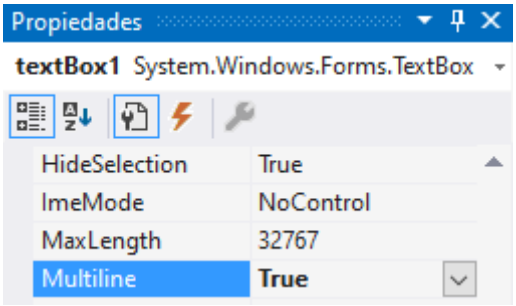
Vamos a ver una serie de controles visuales clásicos en cualquier entorno con ventanas, lo que son los diálogos para abrir un archivo, guardar un archivo, para seleccionar fuentes y para la definición del color.

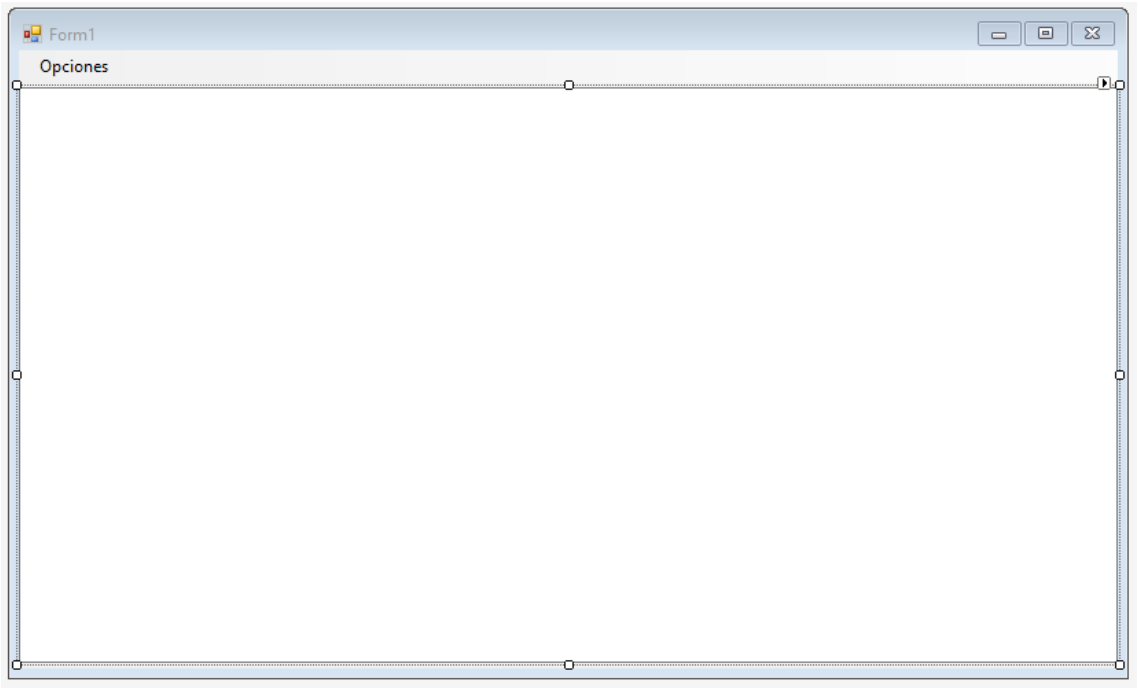
En este capítulo vamos a crear un pequeño editor de textos.

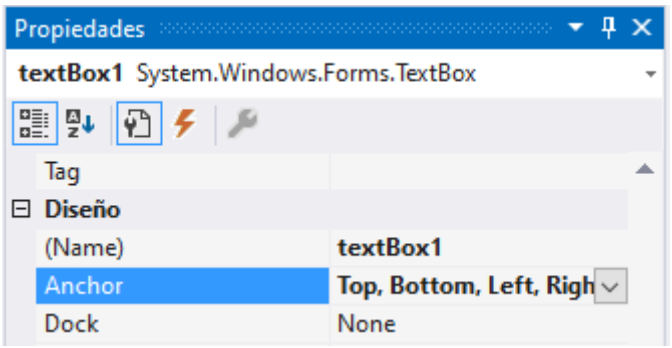
El menú de opciones MenuStrip será el siguiente:



A continuación vamos a agregar un textBox de múltiples líneas.

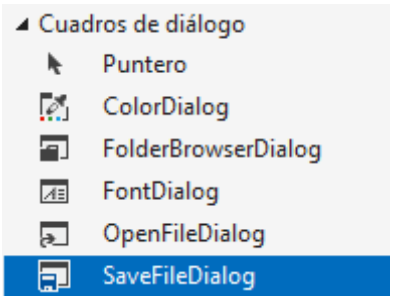






Al configurar Anchor Top, Bottom, Left y Right el textbox se ajustará al ancho y alto de la ventana.

Para agregar un cuadro de diálogo:

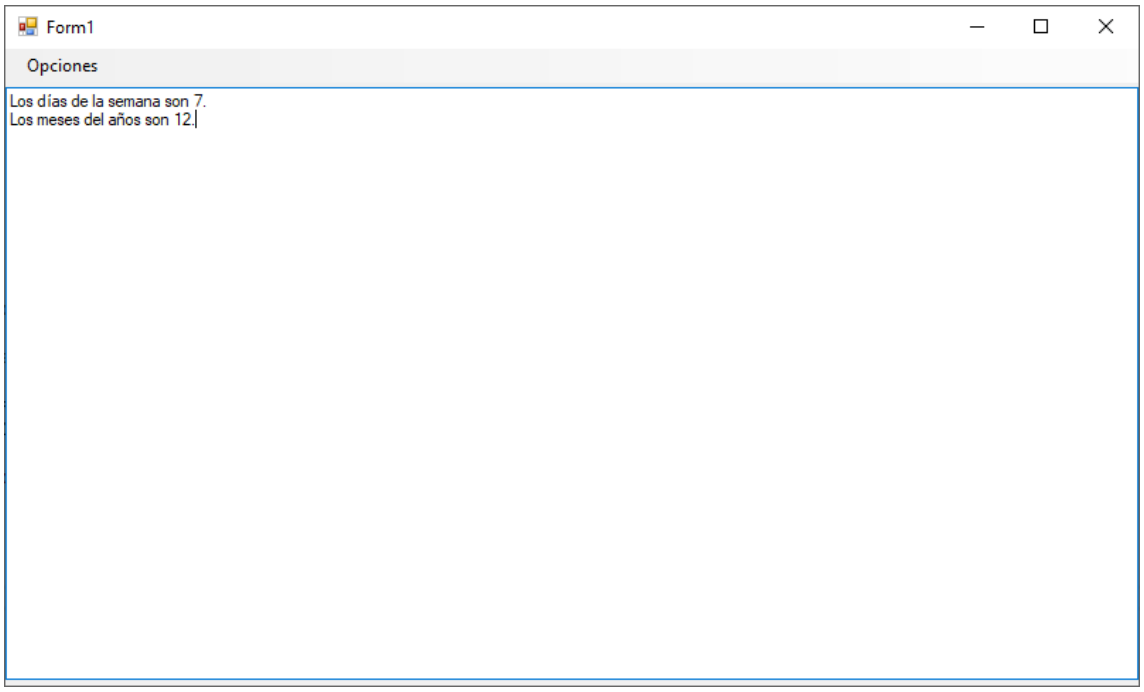


Aunque los arrastremos hacia el formulario este se colocará en la parte inferior, ya que es un control no visual.

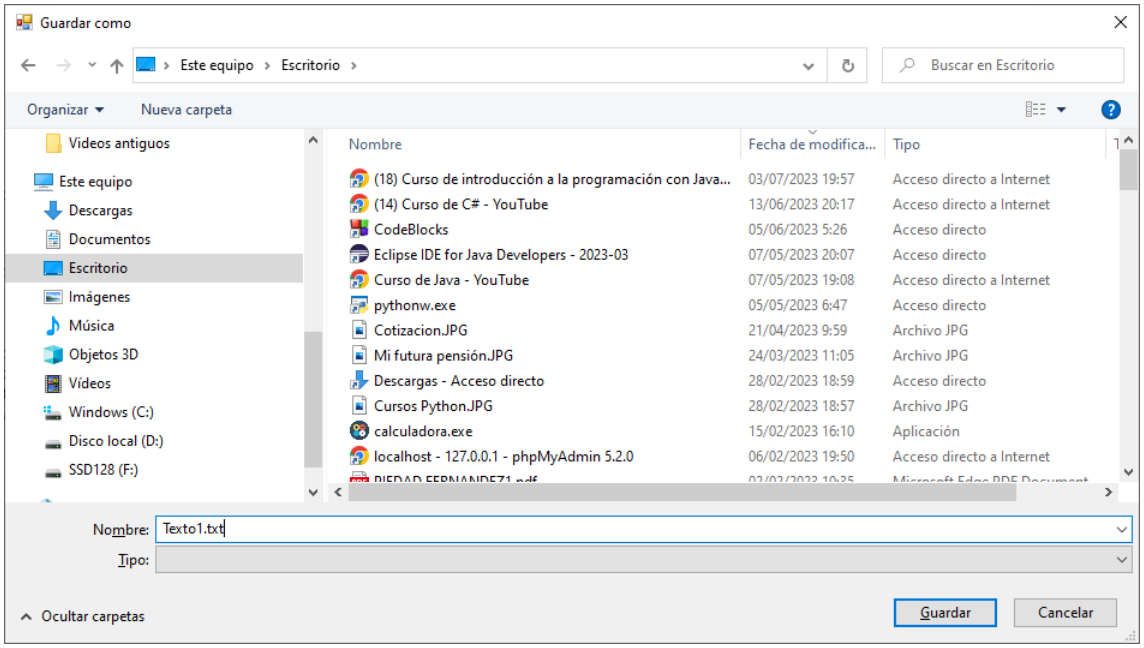
A continuación vamos a codificar el evento Grabar el archivo.

```
1 referencia
private void grabarElArchivoToolStripMenuItem_Click(object sender, EventArgs e)
{
    if(saveFileDialog1.ShowDialog()==DialogResult.OK)
    {
        StreamWriter archivo = new StreamWriter(saveFileDialog1.FileName);
        archivo.Write(textBox1.Text);
        archivo.Close();
        MessageBox.Show("Los datos fueron grabados");
    }
}
```

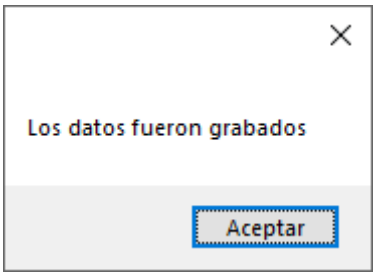
Vamos a ejecutar el programa, escribimos unas cuantas líneas de texto y a continuación Guardamos el archivo.



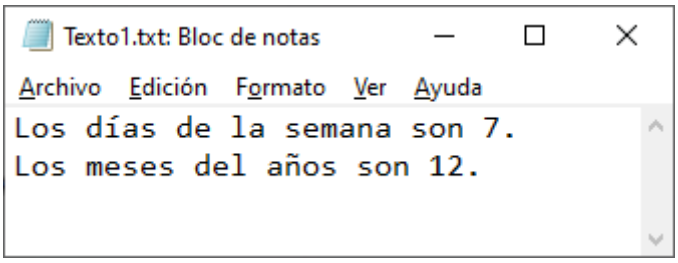
Del menú Opciones seleccionaremos Grabar el archivo.



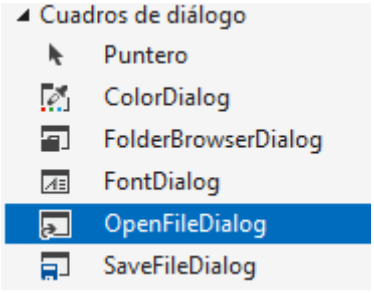
Podemos seleccionar la ubicación del archivo y su nombre lo hemos llamado "Texto1.txt" y guardado en el Escritorio para poder ver su contenido.



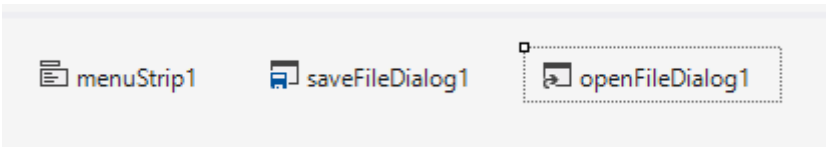
Ahora vamos a ver su contenido, este se abrirá con el bloc de notas.



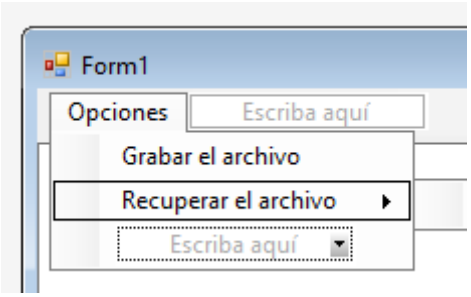
Ahora para recuperar un archivo vamos a necesitar de un control llamado OpenFileDialog.



Que a continuación arrastraremos a nuestro formulario.



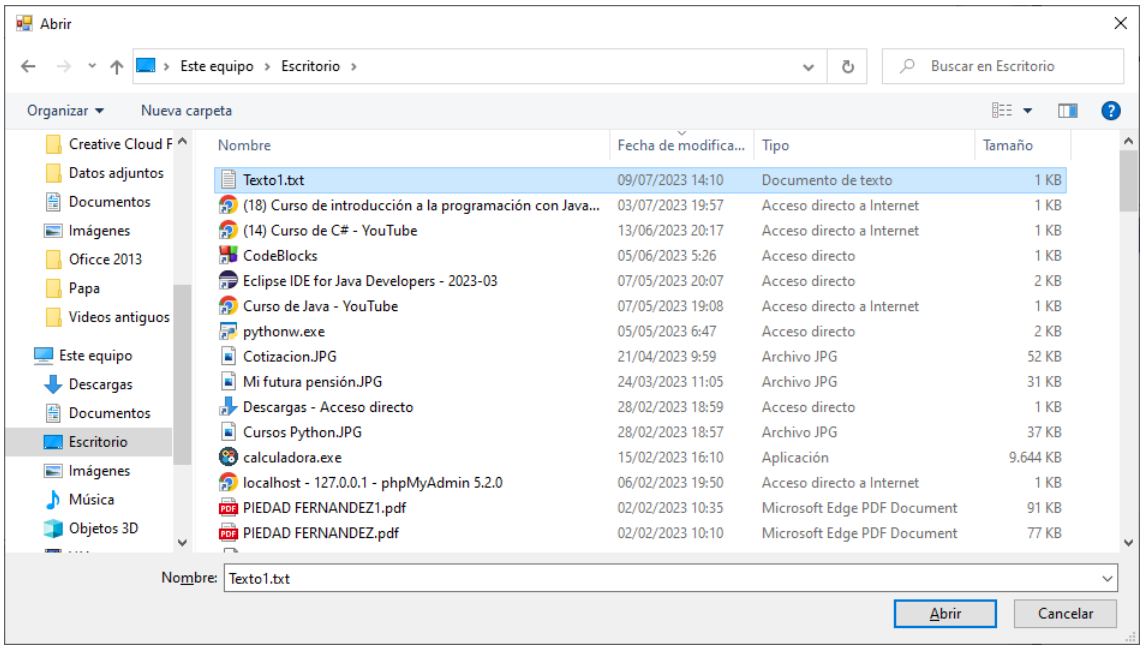
Como objeto no visible se mostrará en la parte inferior del formulario.



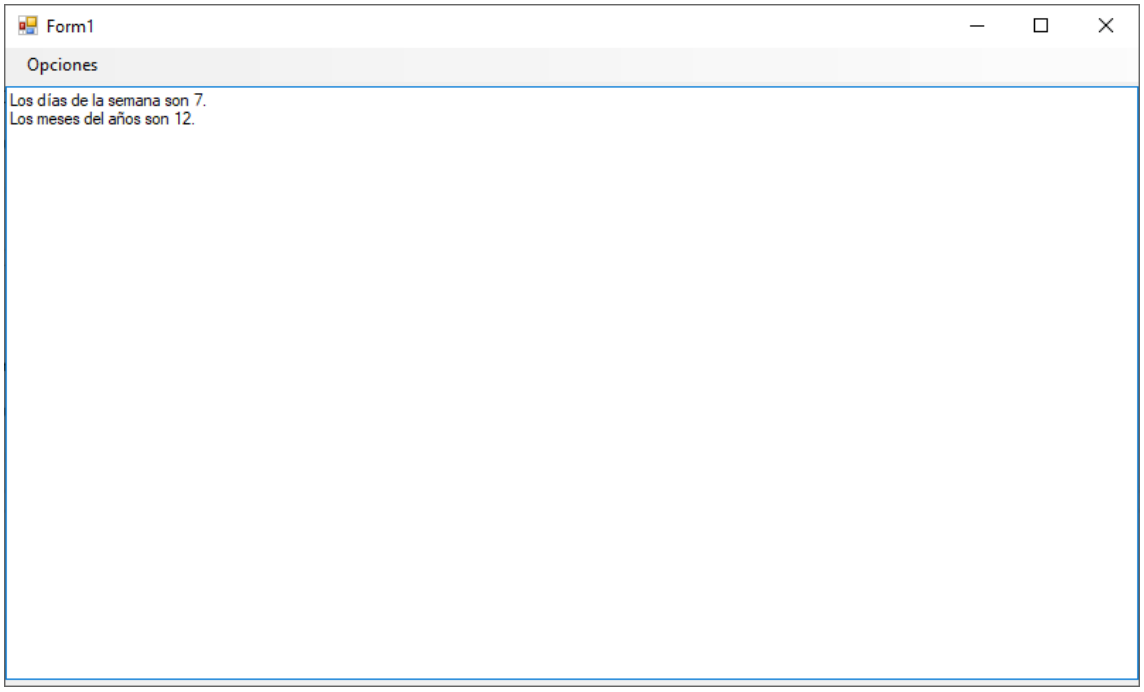
Vamos a agregar el código para el evento click en la opción 'Recuperar el archivo'.

```
1 referencia
private void recuperarElArchivoToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog()==DialogResult.OK)
    {
        StreamReader archivo = new StreamReader(openFileDialog1.FileName);
        textBox1.Text = archivo.ReadToEnd();
        archivo.Close();
    }
}
```

A continuación vamos e ejecutar el programa y recuperamos el archivo que grabamos con anterioridad.



Una vez seleccionado le damos al botón Abrir.

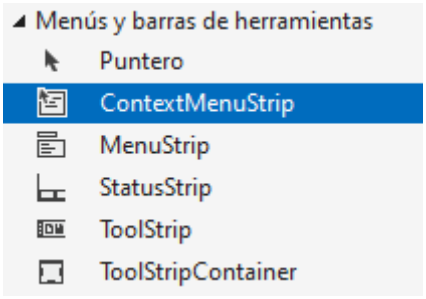


Ya hemos recuperado el archivo de texto.

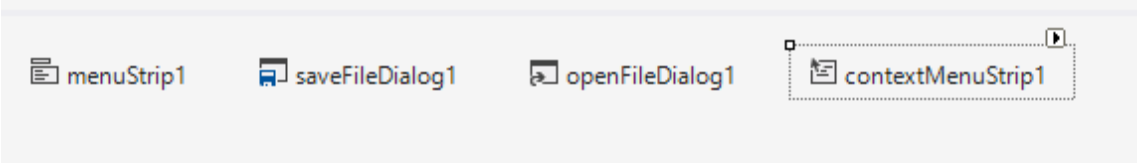
Ya hemos visto los controles de guardar y abrir archivos.

Para poder seleccionar una fuente vamos a utilizar el control FontDialog.

Para esto queremos que con el botón derecho sobre textBox aparezca un menú con la opción para poder cambiar la fuente.

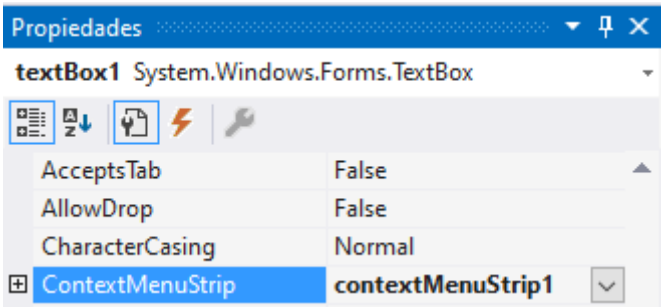


Para ellos seleccionaremos el control ContexMenuStrip.



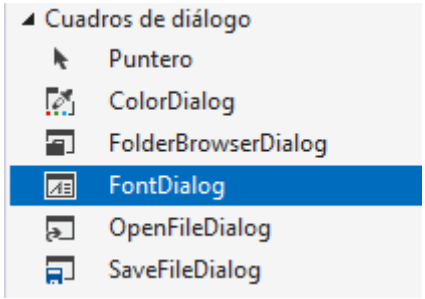
Lo arrastramos a la parte inferior del formulario y a continuación lo vamos a asociar.

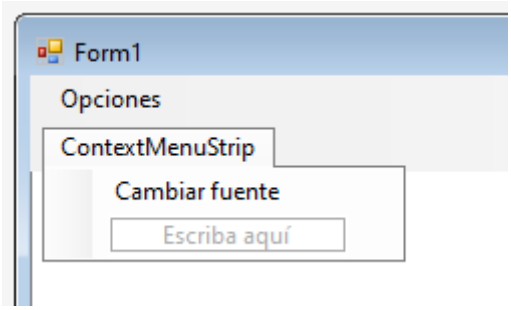
Seleccionamos el textBox y nos vamos a las propiedades.



Una vez seleccionado ya está asociado.

Agregamos el cuadro de diálogo FontDialog a la parte inferior del formulario.



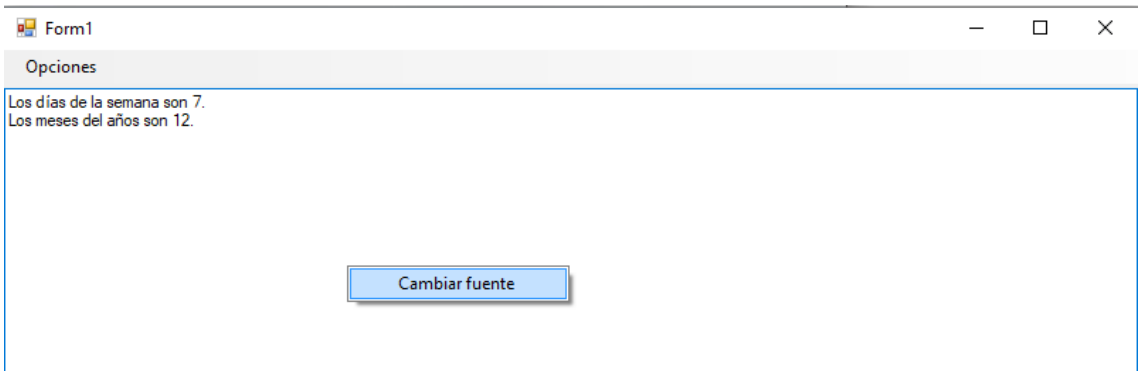


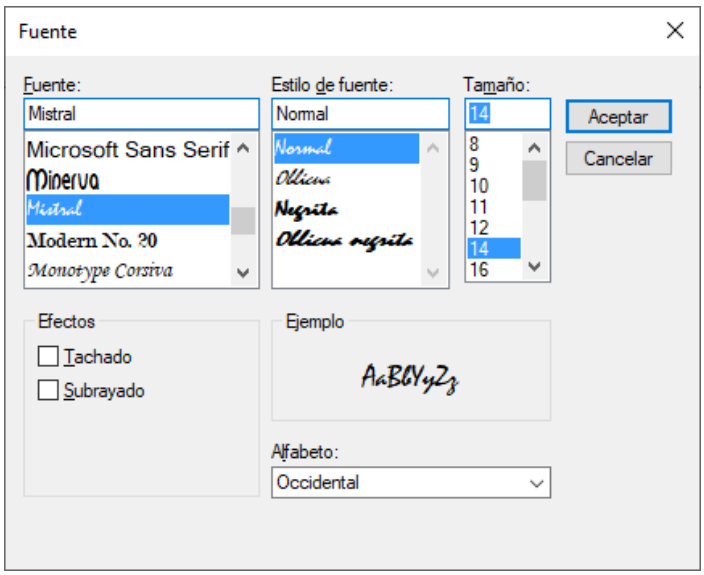
Seleccionamos Cambiar fuente y vamos a programar el evento click.

Desde diseño lo seleccionamos y agregamos el texto Cambiar fuete.

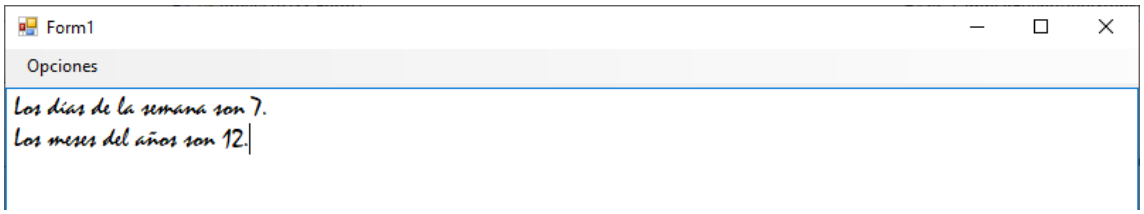
```
1 referencia
private void cambiarFuenteToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (fontDialog1.ShowDialog()==DialogResult.OK)
    {
        textBox1.Font = fontDialog1.Font;
    }
}
```

Vamos a ejecutar el programa, recuperar un archivo y con el botón derecho del mouse en textBox del menú seleccionaremos 'Cambiar fuente'.

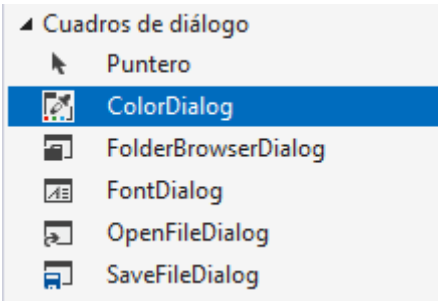




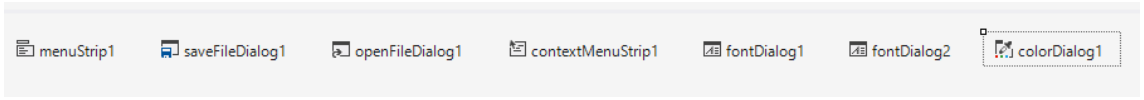
Seleccionamos otra fuente y tamaño seguido del botón Aceptar.



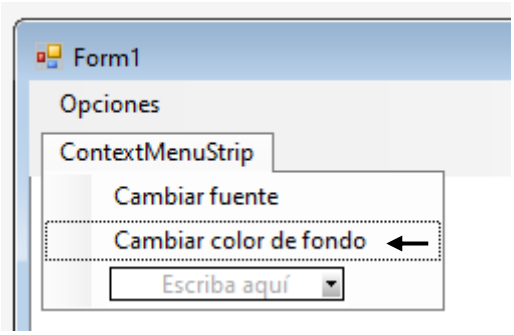
Otro cuadro de diálogo ColorDialog.



Lo arrastramos a la parte inferior como control no visible.



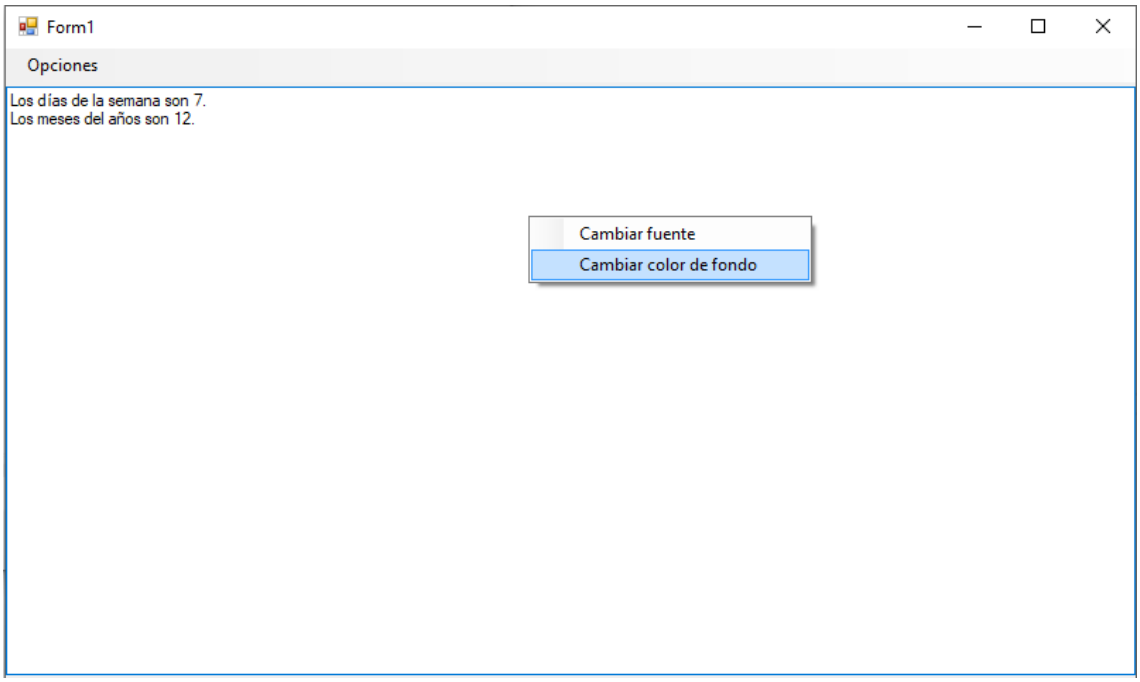
Seleccionaremos contextMenuStrip para agregar una nueva opción.



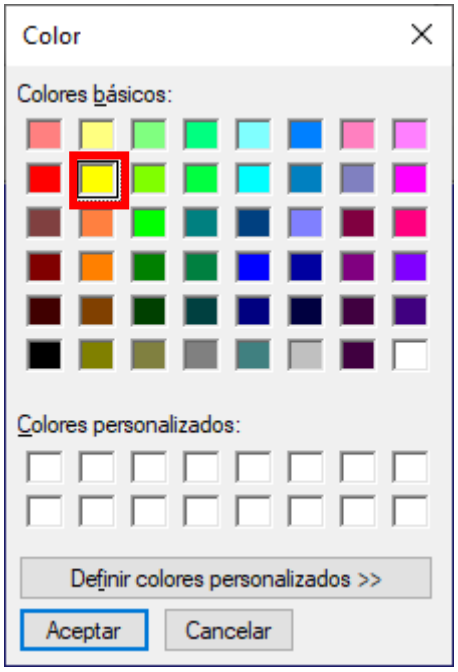
Ahora vamos a agregar el correspondiente código el evento click.

```
1 referencia
private void cambiarColorDeFondoToolStripMenuItem_Click(object sender, EventArgs e)
{
    if(colorDialog1.ShowDialog()==DialogResult.OK)
    {
        textBox1.BackColor = colorDialog1.Color;
    }
}
```

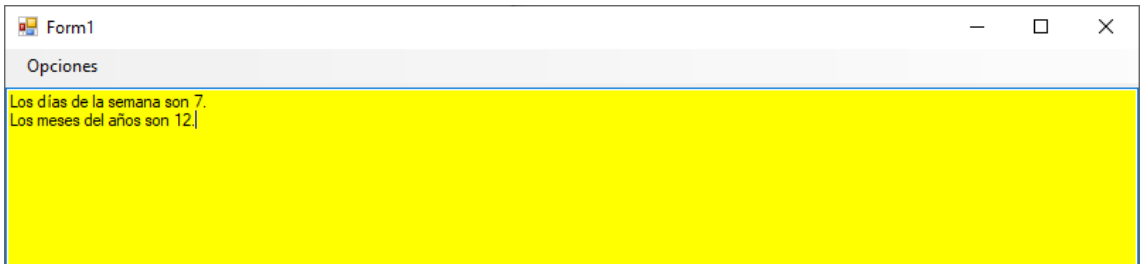
Vamos a ejecutar el programa, recuperar un archivo y por último cambiar el color.



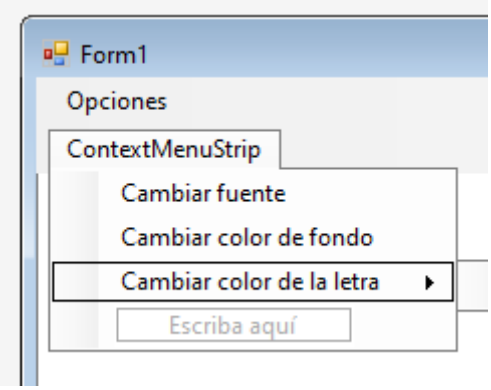
Seleccionamos Cambiar color de fondo.



Seleccionamos un color seguido del botón Aceptar. Hemos seleccionado el color amarillo.



Por último vamos a agregar cambiar color de la letra.



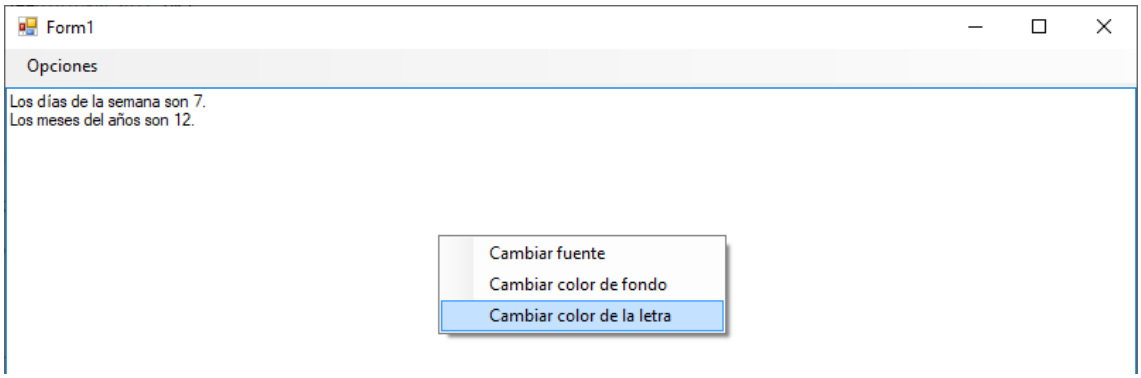
Agregamos otra opción a ContextMenuStrip.

Agregamos otro ColorDoialog.

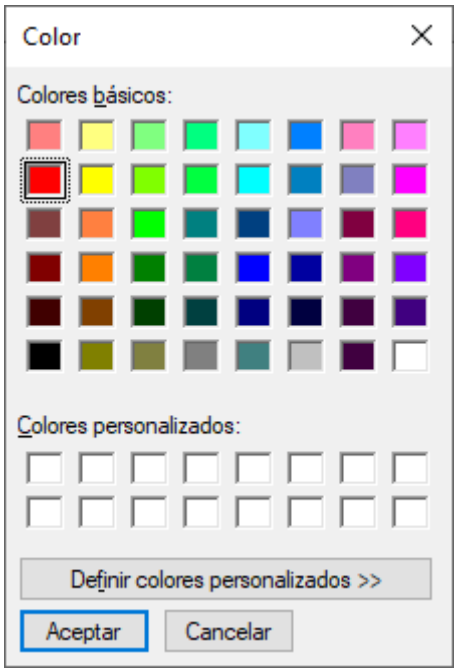
Vamos a programar la opción 'Cambiar color de la letra' con el evento click.

```
1 referencia
private void cambiarColorDeLaLetraToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (colorDialog2.ShowDialog()==DialogResult.OK)
    {
        textBox1.ForeColor = colorDialog2.Color;
    }
}
```

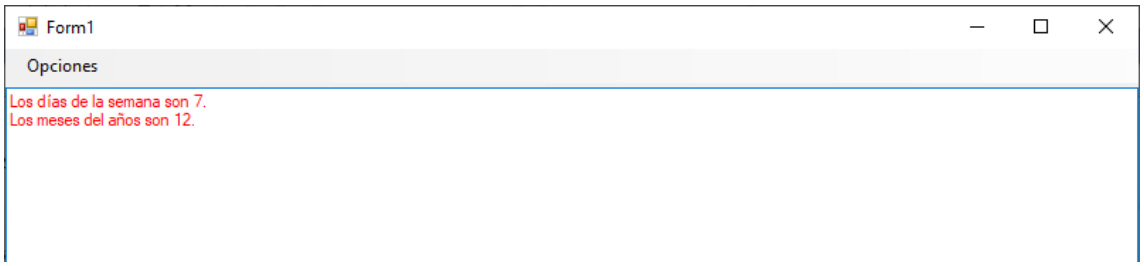
Ejecutamos de nuevo el programa, abrimos un archivo y por último cambiamos el color del texto.



Seleccionamos 'Cambiar color de la letra'.



Seleccionamos el color rojo, seguido del botón aceptar.

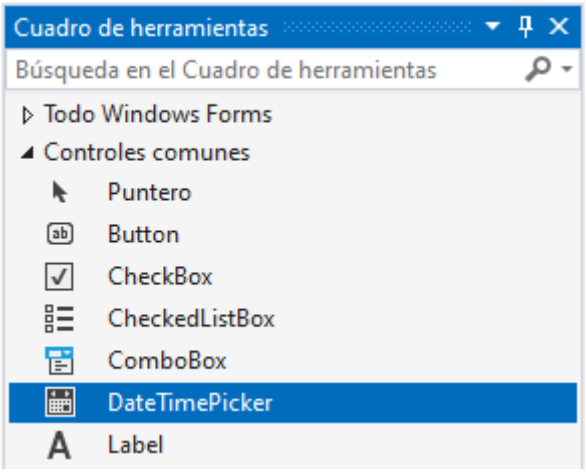


Capítulo 126.- Control visual DateTimePicker (Windows Forms)

Propiedades:

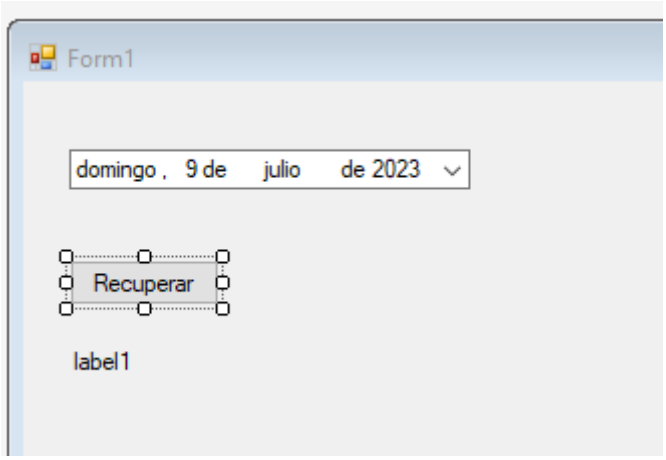
- Format
- Value
- MinDate
- MaxDate

El entorno visual para visualizar una fecha o una hora.



Lo encontraremos en controles comunes.

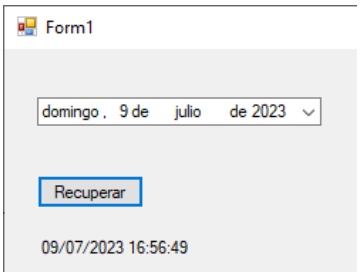
Lo vamos a arrastrar a nuestro formulario con un botón y un label.



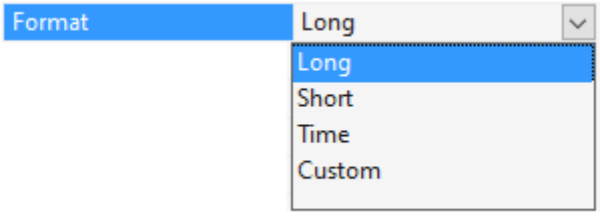
Vamos a programar el botón con el evento click.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = dateTimePicker1.Value.ToString();
}
```

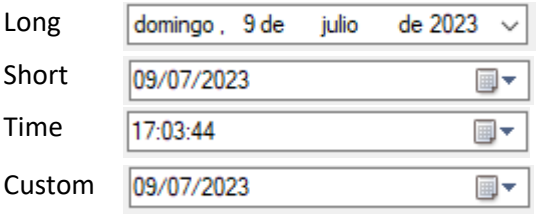
Vamos a ejecutar y presionar el botón Recuperar.



Si seleccionamos el objeto DateTimerPicker y nos vamos a la propiedad Format:



Podemos cambiarlos a Short, Time y Custom.

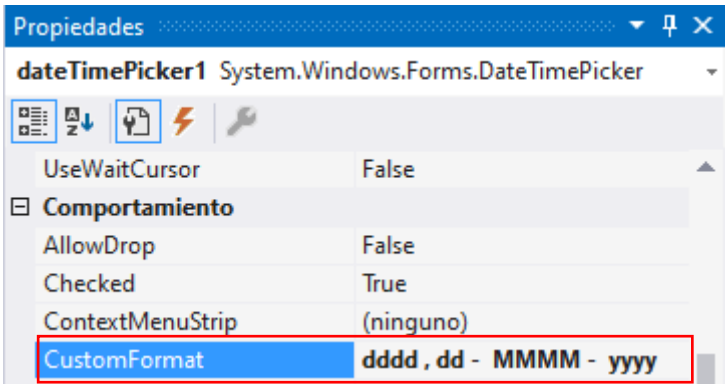


Con Custom tenemso que ir a la propiedad CustomFormat para poner el formato.

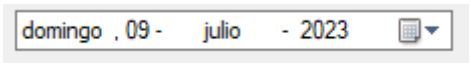
Con los siguientes formatos

Formato string	Descripción
d	De uno o dos dígitos para el día.
dd	Dos dígitos para el día si tiene 1 dígito le agrega un 0.
ddd	Los tres caracteres de la semana abreviado
dddd	El nombre completo del día de la semana.
h	De uno o dos dígitos hora en formato de 12 horas.
hh	Dos dígitos para la hora formato de 12 horas si tiene un digito agrega un 0.
H	Un digito o 2 dígitos para formato de 24 horas.
HH	Dos dígitos para formato de 24 horas, si tiene un digito agrega un 0.
m	Uno o dos dígitos para los minutos.
mm	Dos dígitos para los minutos si solo tiene un dígito agrega un 0.
M	Uno o dos dígitos para el mes.
MM	Dos dígitos para el mes si solo tiene un dígito agrega un 0.
MMM	Tres caracteres del mes en abreviado.
MMMM	El nombre completo del mes.
s	Uno o dos dígitos para los segundos.
ss	Dos dígitos para los segundos si solo tiene un dígito agrega un 0.
t	Una letra A.M/P.M Abreviación (A.M. se muestra "A").
tt	Dos letras A.M/P.M. Abreviación (A.M. se muestra "AM").
y	Un dígito para el año (2001 se muestra "1").
yy	Dos dígitos para el año (2001 se muestra "01").
yyyy	El año completo (2001 se muestra "2001").

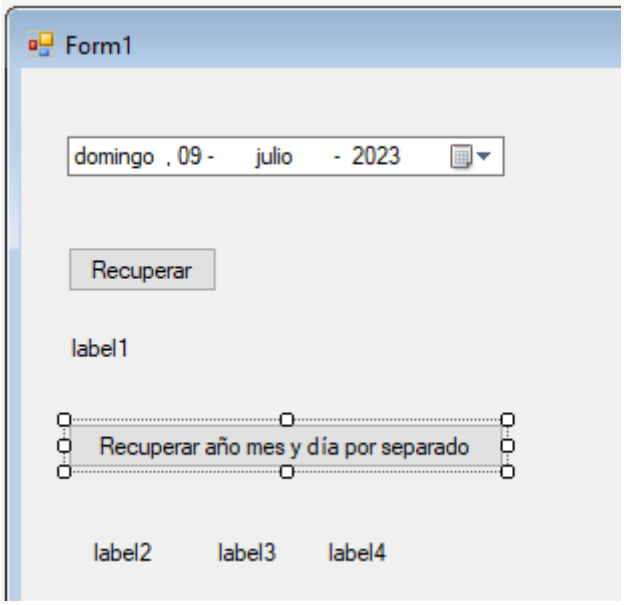
Si hemos seleccionado en format Custom podemos personalizar el formato:



Este será el resultado:



Ahora vamos a ver como recuperar año, mes y día por separado.

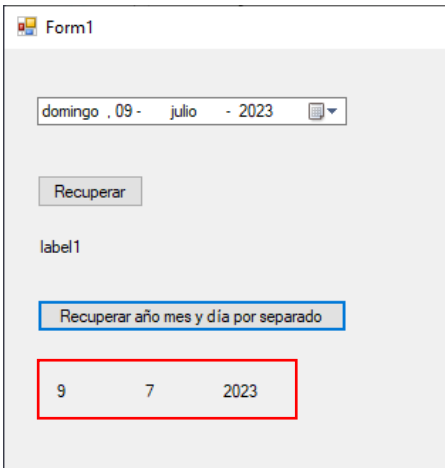


En label2 muestre el día, label3 muestre el mes y label4 muestre el año.

El código del botón 'Recuperar año mes y día por separado' con el evento click este será el código:

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    label2.Text = dateTimePicker1.Value.Day.ToString();
    label3.Text = dateTimePicker1.Value.Month.ToString();
    label4.Text = dateTimePicker1.Value.Year.ToString();
}
```

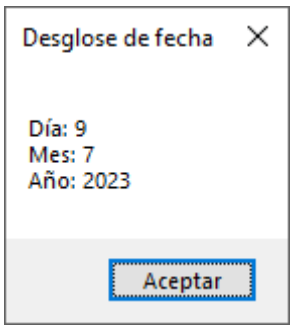
Si ejecutamos este será el resultado:



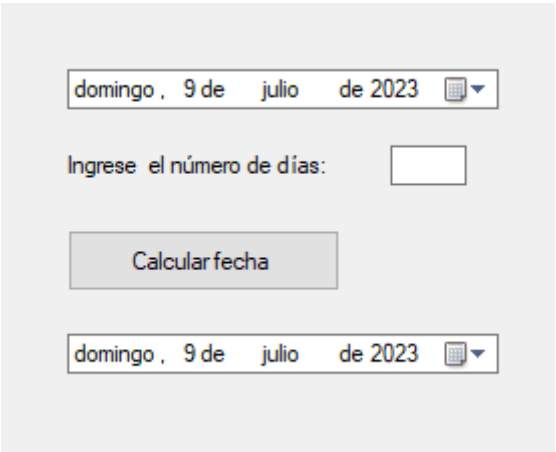
Vamos a realizar otro ejemplo:

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    string mensaje = "";
    mensaje = mensaje + "Día: " + dateTimePicker1.Value.Day.ToString();
    mensaje = mensaje + "\nMes: " + dateTimePicker1.Value.Month.ToString();
    mensaje = mensaje + "\nAño: " + dateTimePicker1.Value.Year.ToString();
    MessageBox.Show(mensaje, "Desglose de fecha");
}
```

Si ejecutamos y presionamos el botón "Recuperar año mes y día por separado" mostrará el siguiente cuadro de diálogo:



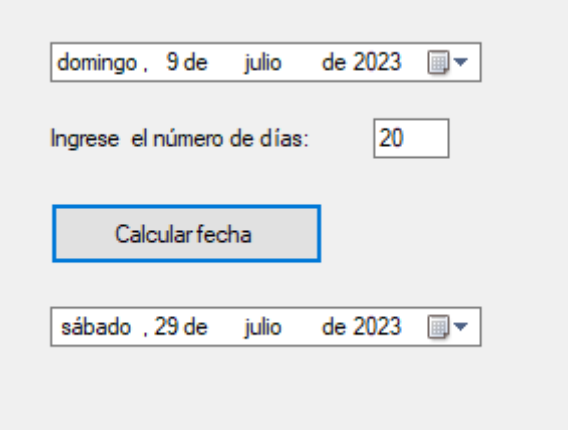
Vamos a ver otro ejemplo donde partiendo de una fecha que nos pregunte una cantidad de días y que nos muestre que fecha será.



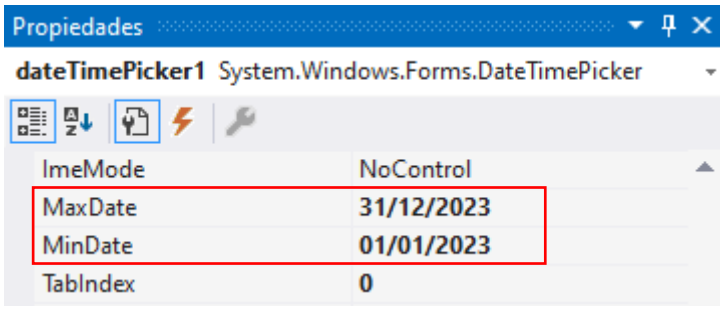
Vamos a programar el botón "Calcular fecha" con el evento click.

```
1 referencia
private void button3_Click(object sender, EventArgs e)
{
    int dias = int.Parse(textBox1.Text);
    DateTime fecha = dateTimePicker1.Value.AddDays(dias);
    dateTimePicker2.Value = fecha;
}
```

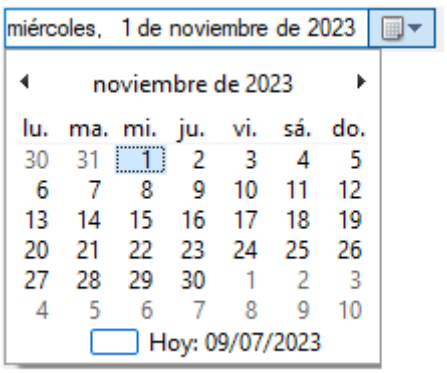
Vamos a ejecutar y queremos que nos calcule que fecha será dentro de 20 días.



También podemos ingresar fechas que se encuentre en un determinado rango. Si queremos introducir fechas comprendidas solo para el año 2023.



Fijamos la fecha mínima y máxima.



Solo nos dejará seleccionar fechas que estén en el rango que le hemos especificado.

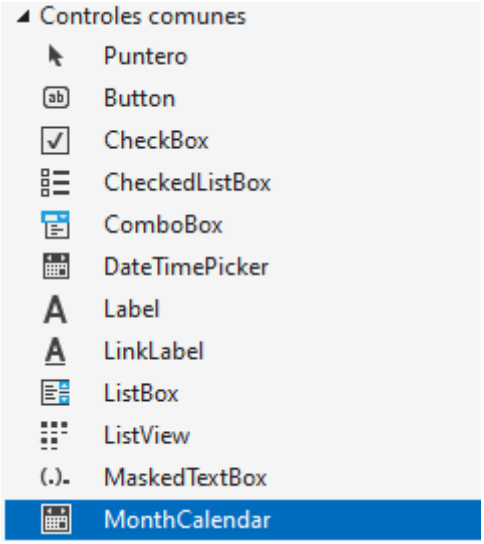
Capítulo 127.- Control visual MonthCalendar (Windows Forms)

Propiedades:

- CalendarDimensions
- MinDate
- MaxDate
- FirstDateOfWeek
- MaxSelectionCount

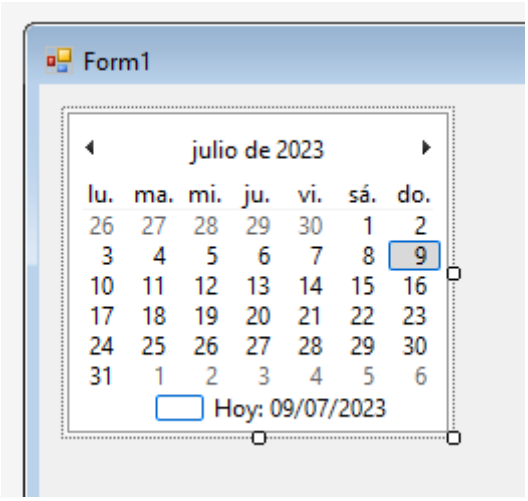
Evento:

- DateSelected

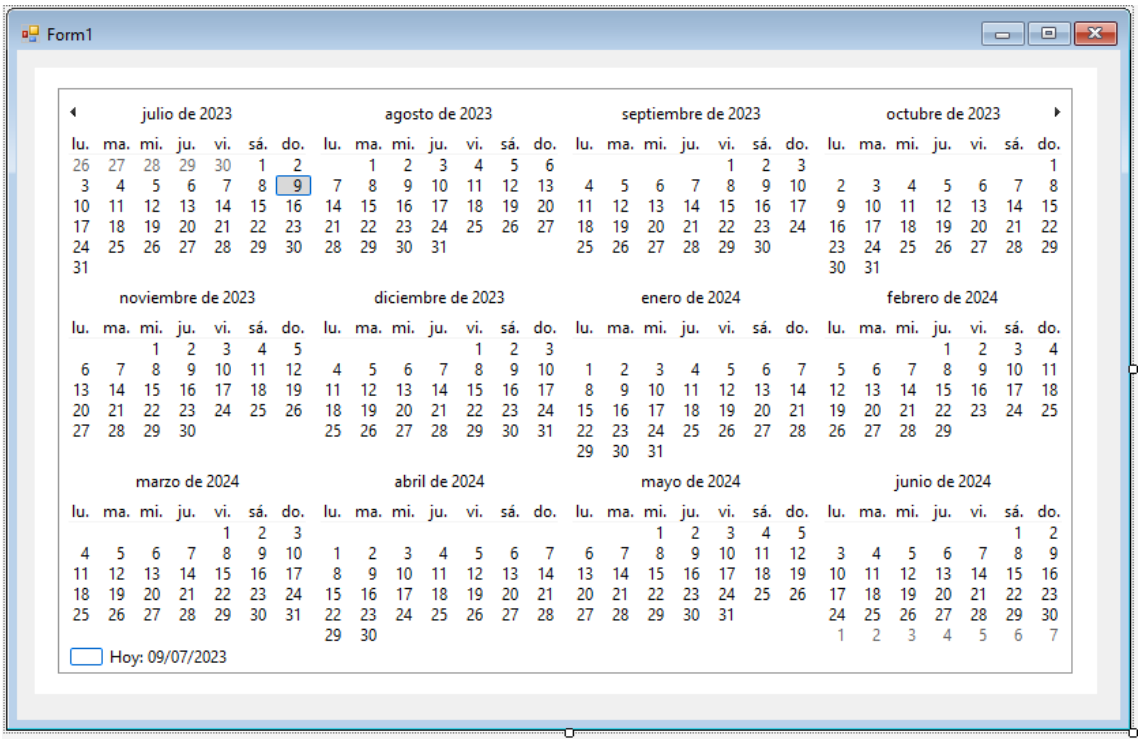


Se encuentra en controles comunes.

Lo vamos a arrastrar al formulario.

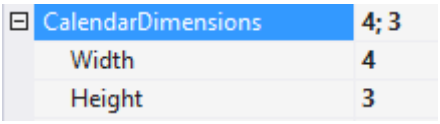


Lo podemos expandir si queremos que muestre todos los meses.

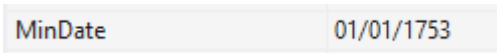


La propiedad:

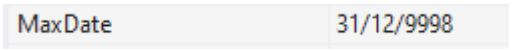
CalendarDimensions: Configurar las columnas y filas.



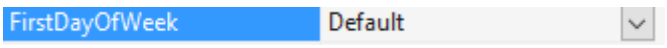
MinDate: Fecha mínima.



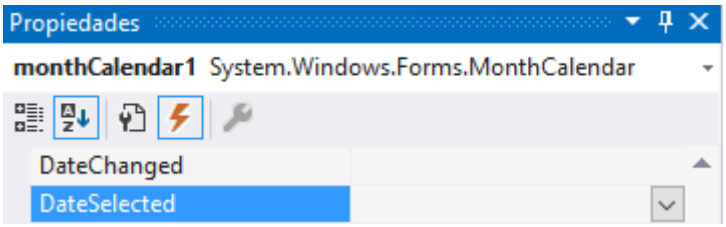
MaxDate: Fecha máxima.



FirstDateOfWeek: Configurar el primer día de la semana.



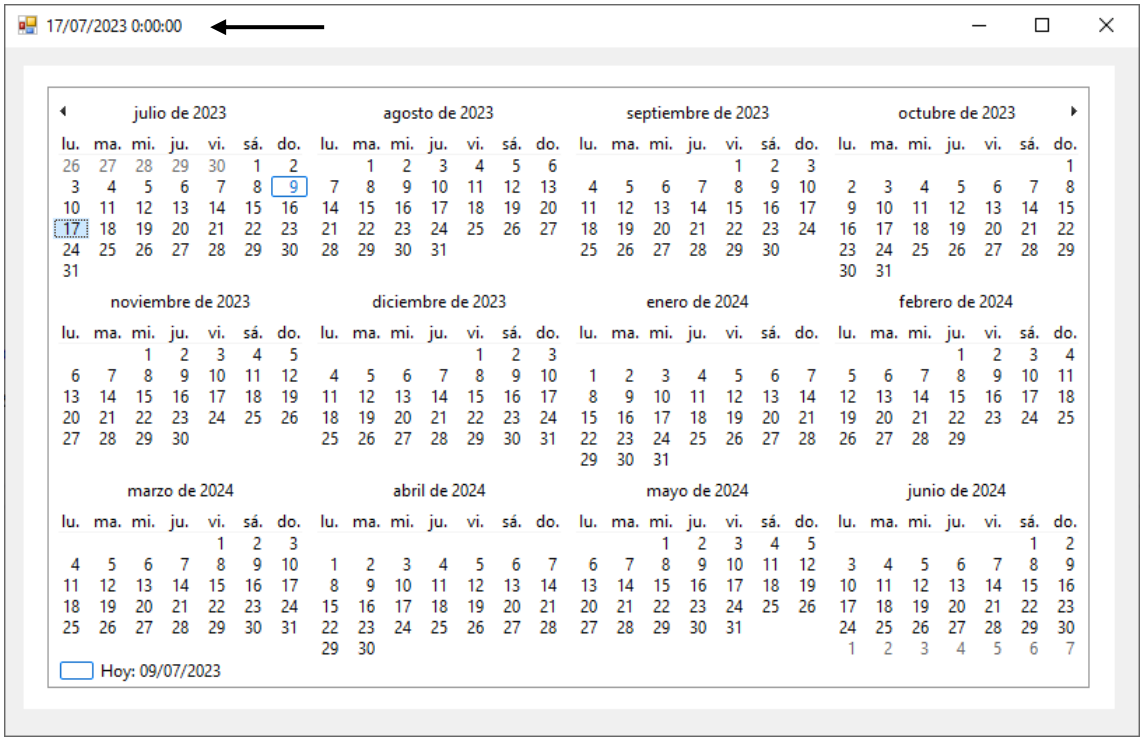
MaxSelectionCount: esto lo encontraremos en eventos.



Es cuando seleccionamos una fecha.

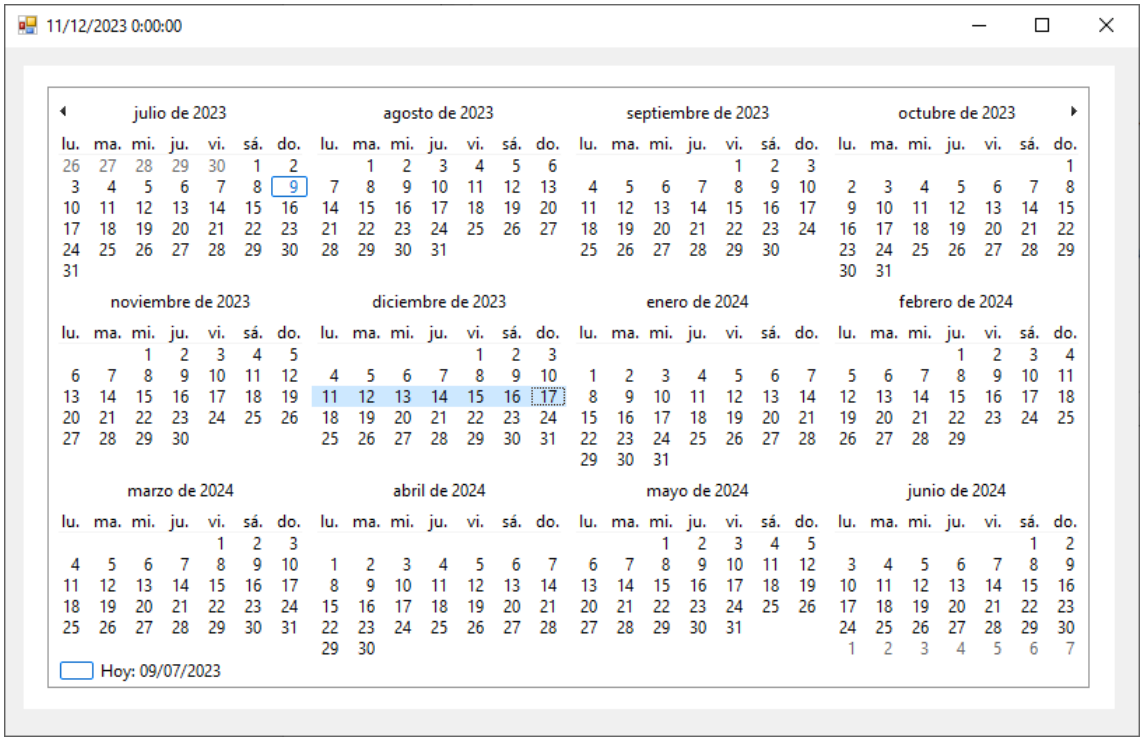
```
1 referencia
private void monthCalendar1_DateSelected(object sender, DateRangeEventArgs e)
{
    Text = monthCalendar1.SelectionRange.Start.ToString();
}
```

Vamos a ejecutar:

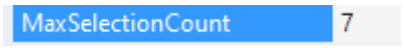


Cuando seleccionamos una fecha esta la muestra en el título de la ventana.

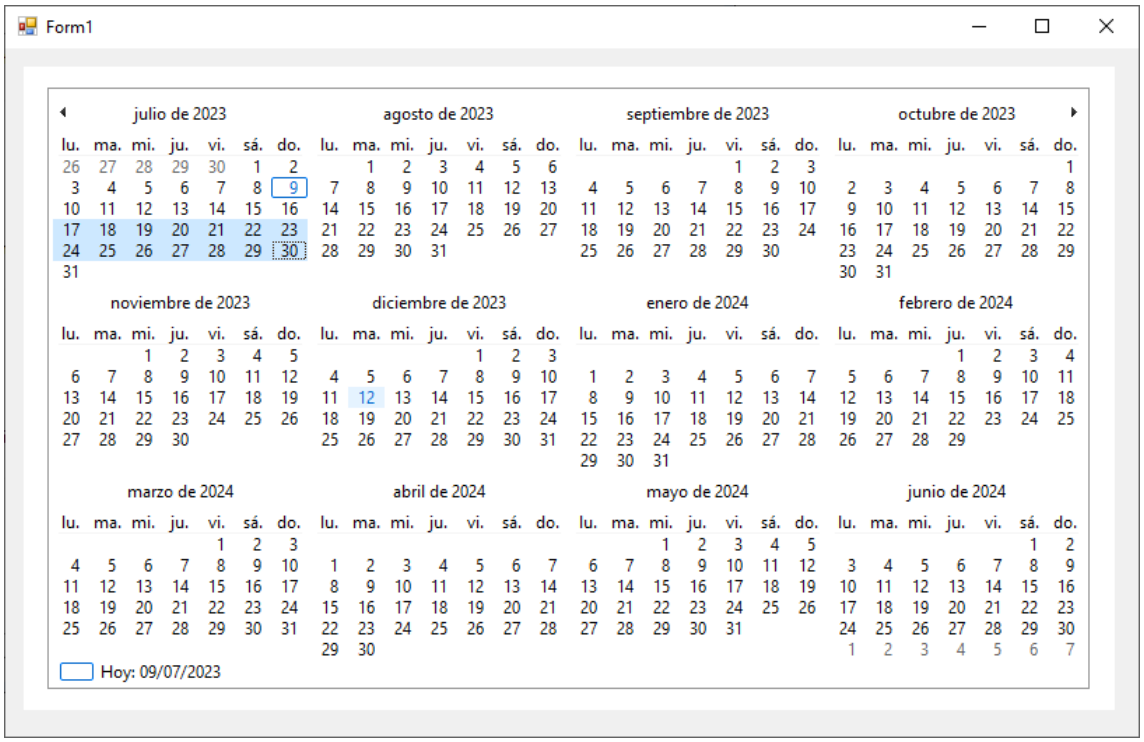
Como configurar los días que se pueden seleccionar.



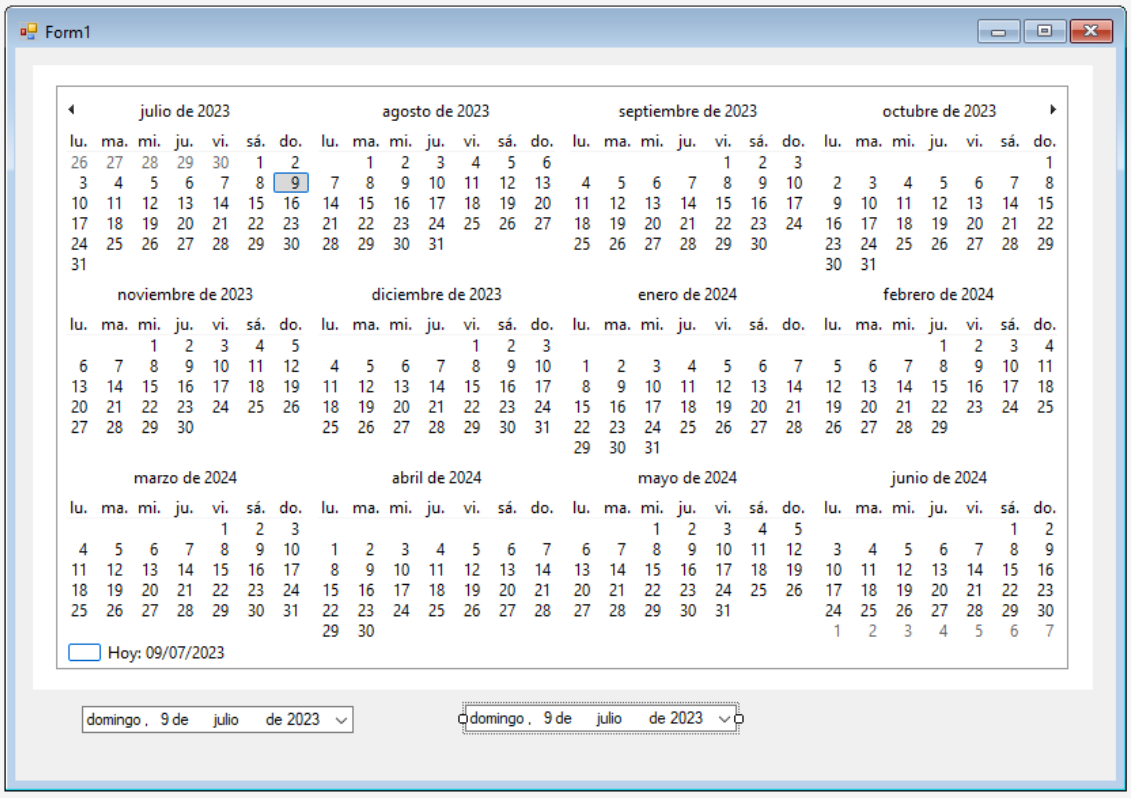
Por defecto me deja siete días.



Vamos a cambiarlo a 14 días.



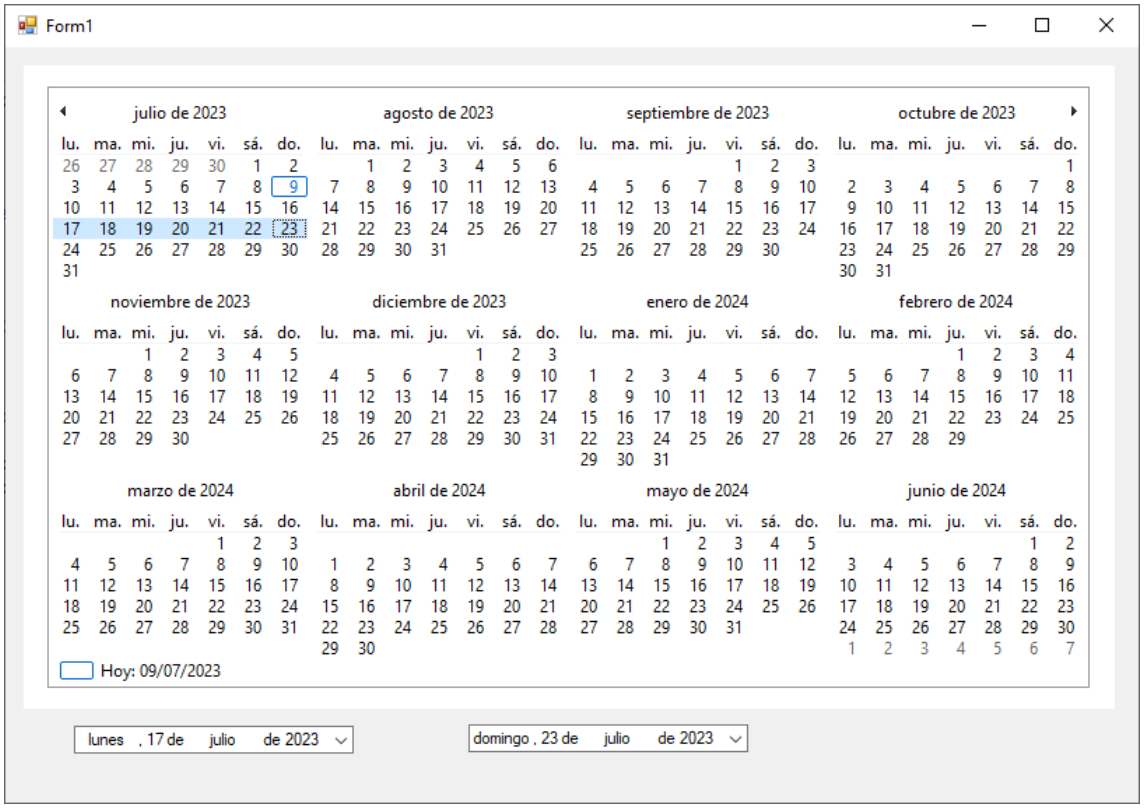
Ahora queremos que nos muestre dentro de una selección la fecha inicio y la fecha fin, para ello vamos a añadir dos DateTimeSicker.



Vamos al evento DateSelected.

```
1 referencia
private void monthCalendar1_DateSelected(object sender, DateRangeEventArgs e)
{
    dateTimePicker1.Value = monthCalendar1.SelectionRange.Start;
    if (monthCalendar1.SelectionRange.Start!= monthCalendar1.SelectionRange.End)
    {
        dateTimePicker2.Value = monthCalendar1.SelectionRange.End;
    }

}
```

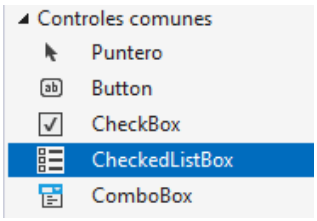
Vamos a ejecutar y seleccionar un rango.



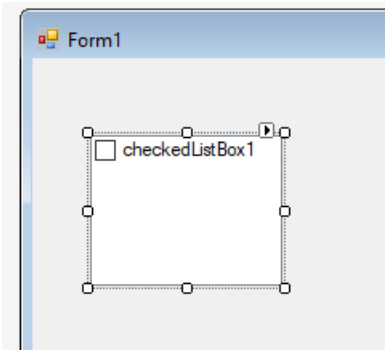
Nos muestra la fecha inicio y fecha final siempre que tengamos seleccionado más de un día.

Capítulo 128.- Control visual CheckedListBox (Windows Forms)

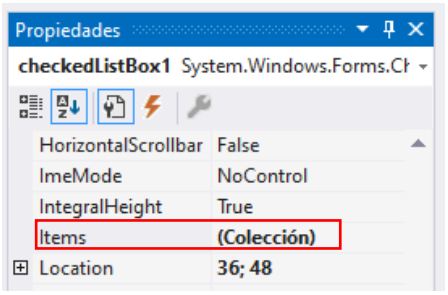
En este capítulo vamos a aprender a utilizar el control visual CheckedListBox.

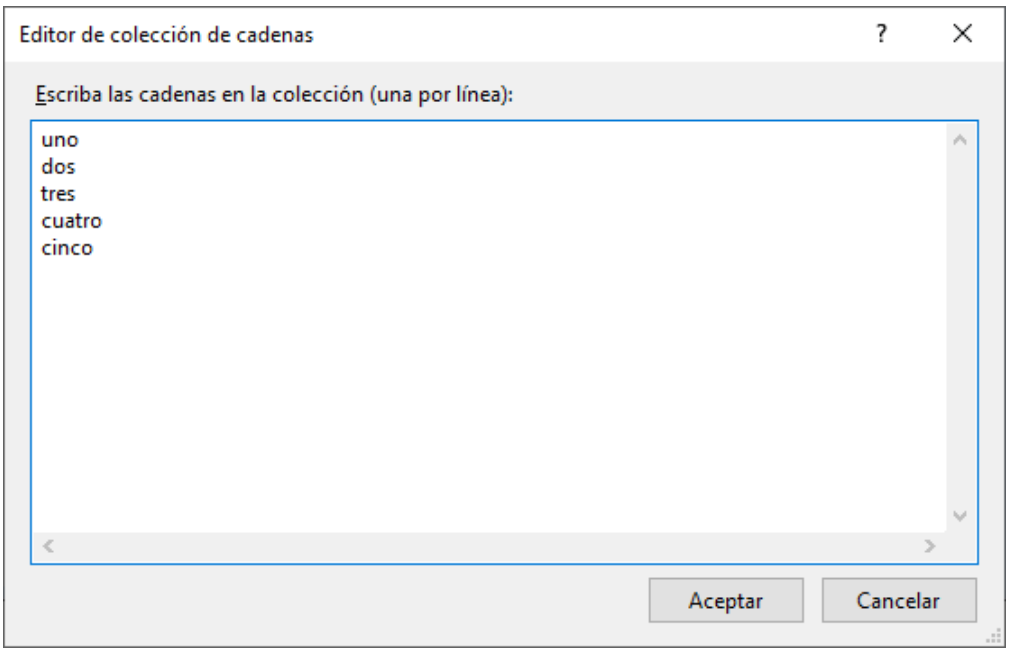


Lo seleccionamos y lo arrastramos a nuestro formulario.

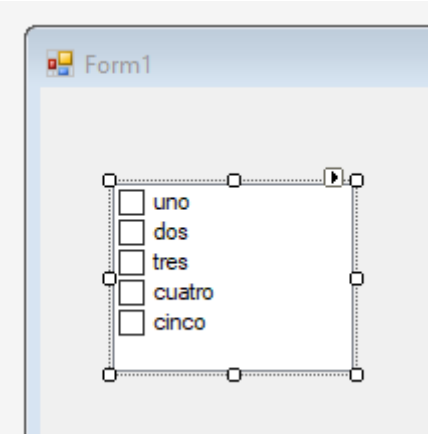


Para agregar Items desde el panel de propiedades:



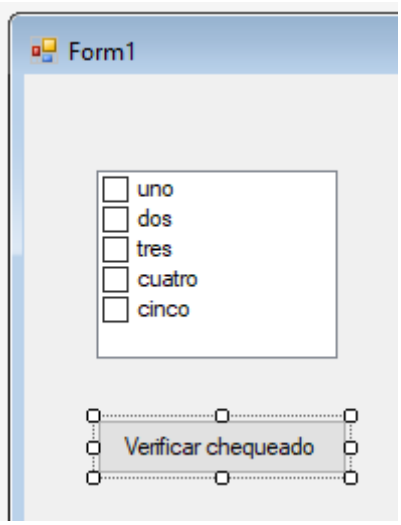


Seguido del botón Aceptar.



En tiempo de ejecución podremos seleccionar los elementos que deseemos.

Para saber los elementos que se han chequeado vamos a agregar un botón.

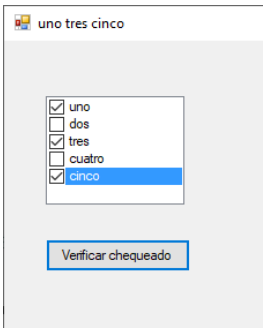


Queremos que nos muestre en el titulo del formulario aquellos ítems que hemos chequeado.

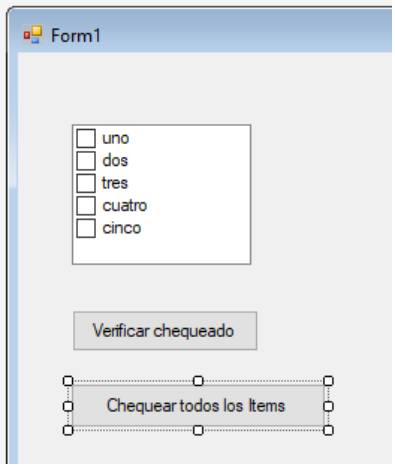
Para ello vamos a programar el botón con el evento click.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    Text = "";
    for (int f=0; f<checkedListBox1.Items.Count; f++)
    {
        if (checkedListBox1.GetItemChecked(f))
        {
            Text += checkedListBox1.Items[f].ToString() + " ";
        }
    }
}
```

Vamos a ejecutar y seleccionar algunos de los Items.



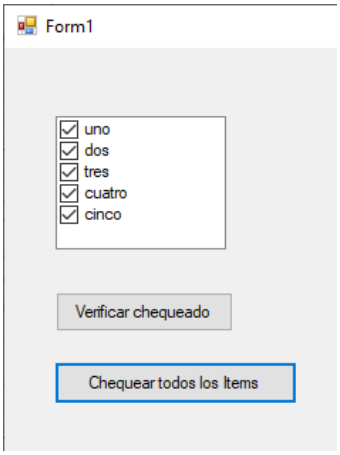
Ahora vamos a agregar un botón para poder chequear todo los Items del control visual DheckedListBox.



Vamos a programar el botón con el evento click.

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    for (int f = 0; f < checkedListBox1.Items.Count; f++)
    {
        checkedListBox1.SetItemChecked(f, true);
    }
}
```

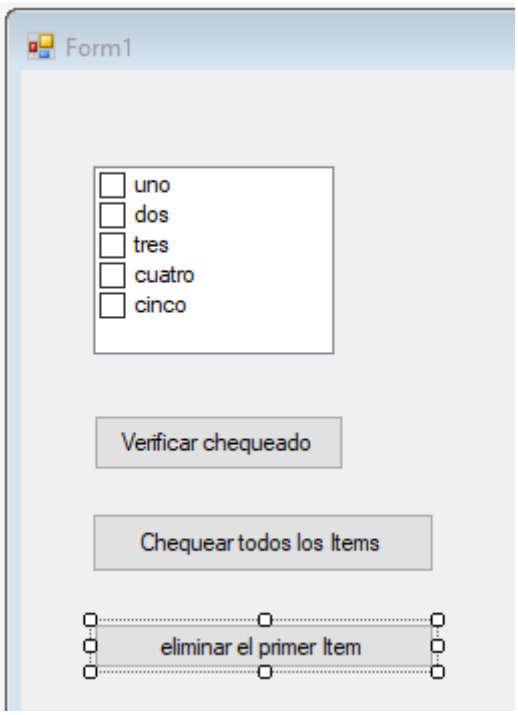
Vamos a ejecutar y seleccionaremos el botón "Chequear todos los Items".



Si queremos quitar todos los cheque cambiaremos el true por un false.

Ahora en tiempo de ejecución queremos eliminar el primer Item.

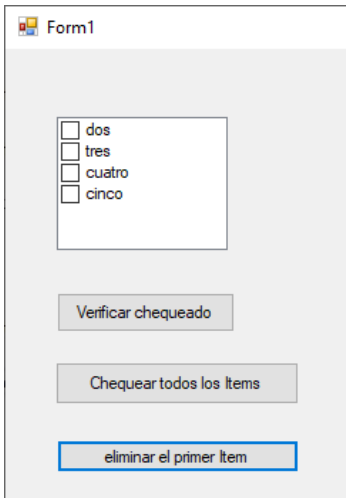
Para ello vamos a agregar un nuevo botón.



Este será el código del botón "eliminar el primer item" del evento click.

```
1 referencia
private void button3_Click(object sender, EventArgs e)
{
    checkedListBox1.Items.RemoveAt(0);
}
```

Vamos a ejecutar y presionar el botón "eliminar el primer item".



Si presionamos de nuevo el botón volveremos a eliminar el primer Item.

Vamos a modificar el código:

```
1 referencia
private void button3_Click(object sender, EventArgs e)
{
    if (checkedListBox1.Items.Count > 0)
    {
        checkedListBox1.Items.RemoveAt(0);
    }
}
```

Vamos a ejecutar de nuevo para eliminar todos los ítems.

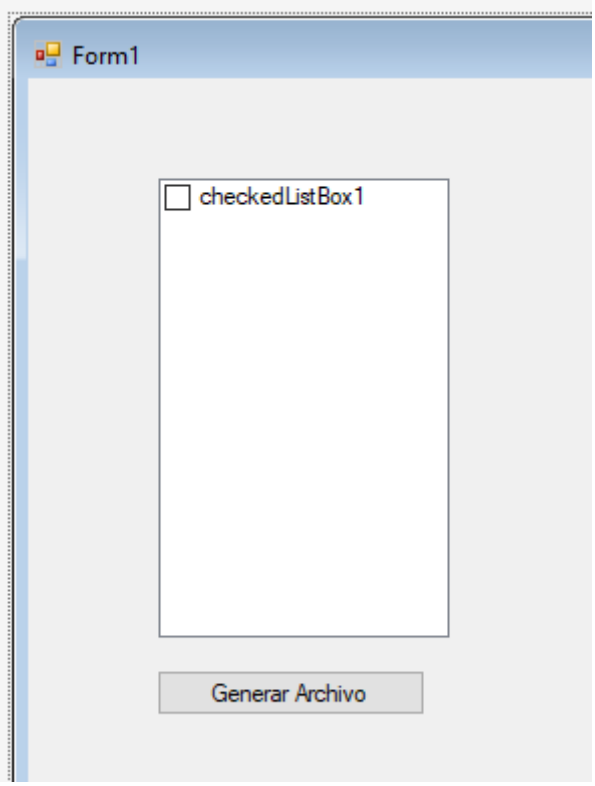
De este modo evitamos que se produzca un error en tiempo de ejecución si intentamos eliminar el primer Item y checkedListBox se encuentra vacio.

Problema propuesto

Almacenar en un archivo de texto en cada fila los productos que compramos generalmente en un supermercado.

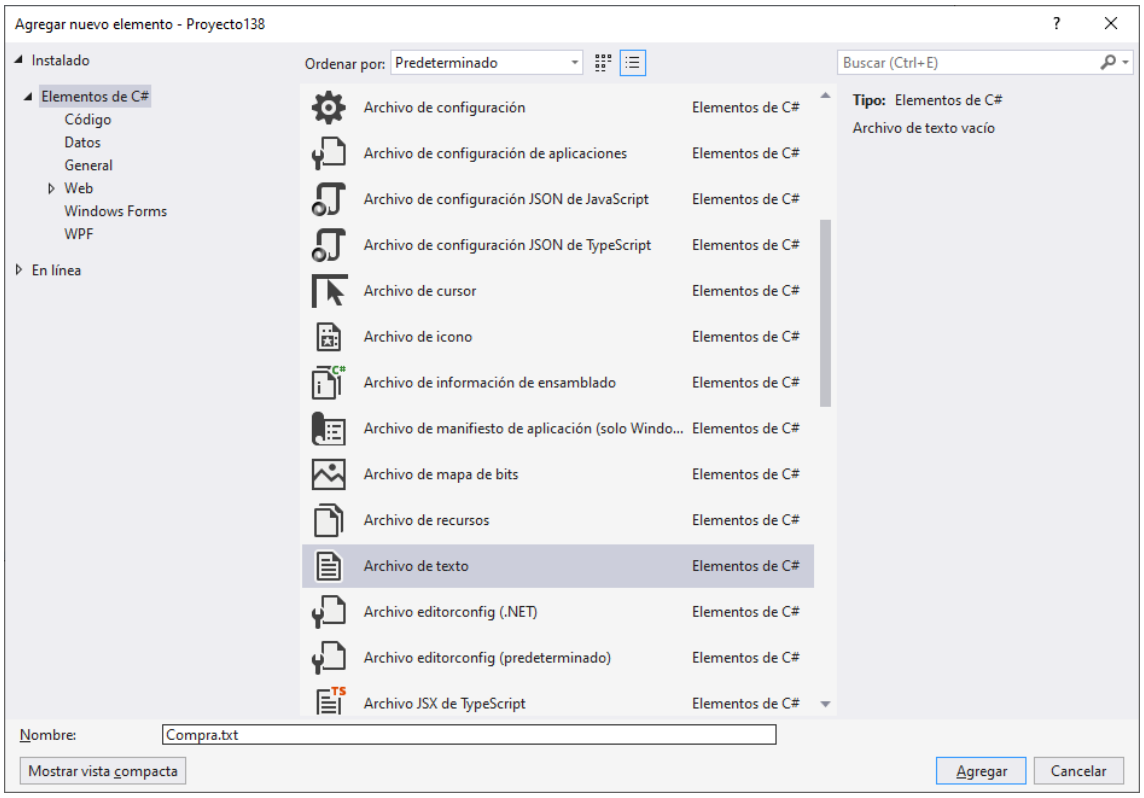
La aplicación al iniciarse debe cargar cada fila del archivo de texto y generar un checkbox en un control de tipo chekedListBox.

El operador selecciona los ítems a comprar y al presionar un botón genera otro archivo de texto con los artículos seleccionados del control CheckListBox.

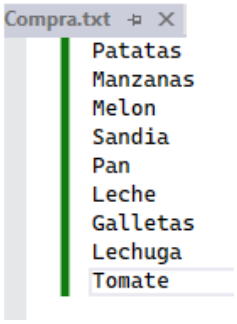


Botón derecho sobre el nombre del proyecto, agregar y de este nuevo elemento.

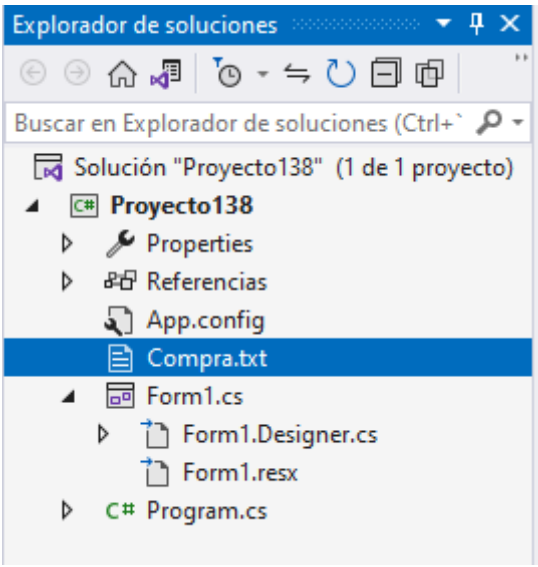
Vamos a crear el archivo, para ellos vamos a agregar un nuevo elemento:



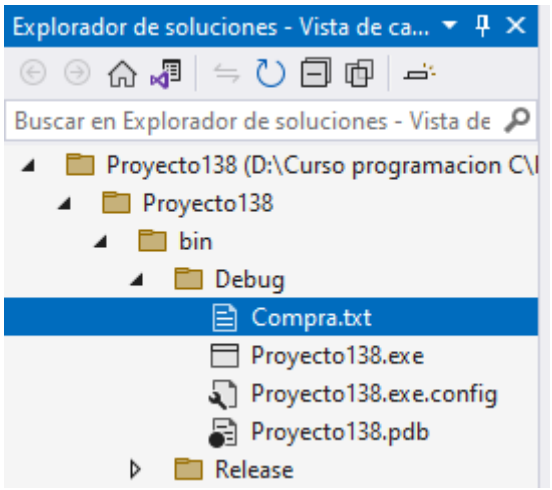
Como nombre "compra.txt".



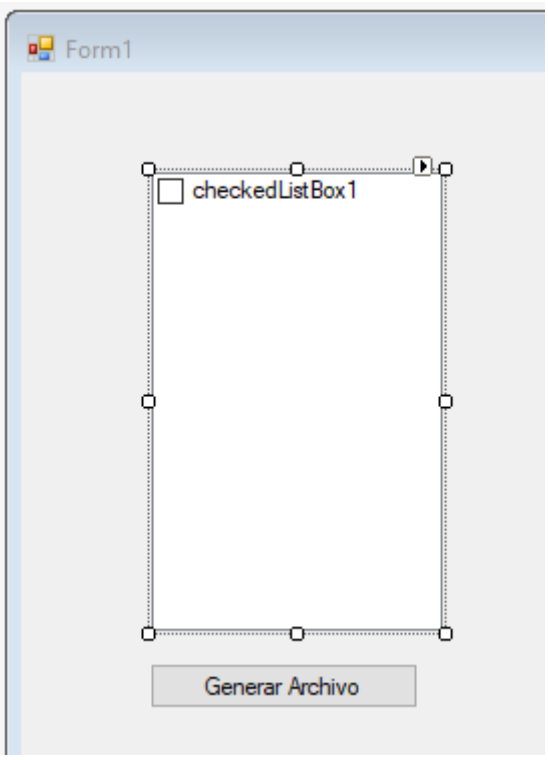
Lo guardamos y ya lo tenemos en nuestro proyecto.



Vamos a mover el archivo a donde se encuentra el ejecutable.



Vamos a diseñar el formulario:



Lo que queremos que cuando ejecutemos la aplicación automaticamente se cargue el ChekedListBox.

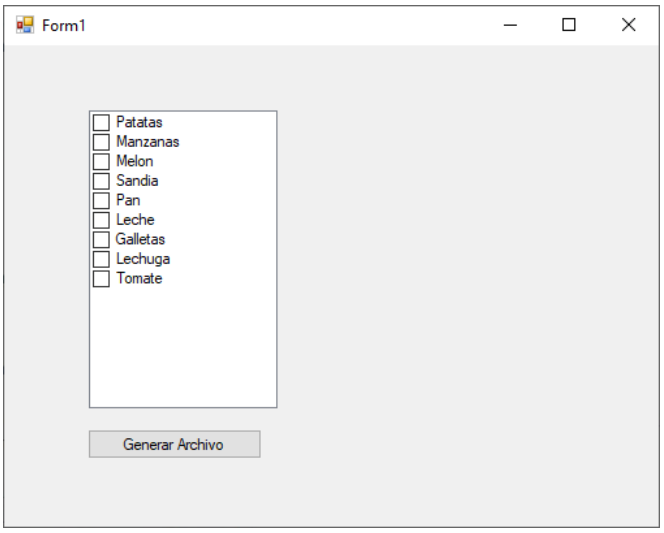
Cuando se carga el formulario hay un evento llamado Load.

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    StreamReader archivo = new StreamReader(Directory.GetCurrentDirectory() + "\\compra.txt");
    while (!archivo.EndOfStream)
    {
        string linea = archivo.ReadLine();
        checkedListBox1.Items.Add(linea);
    }
    archivo.Close();
}
```

Directorio donde se encuentra el ejecutable.

Carácter control.

Si ejecutamos este será el resultado:

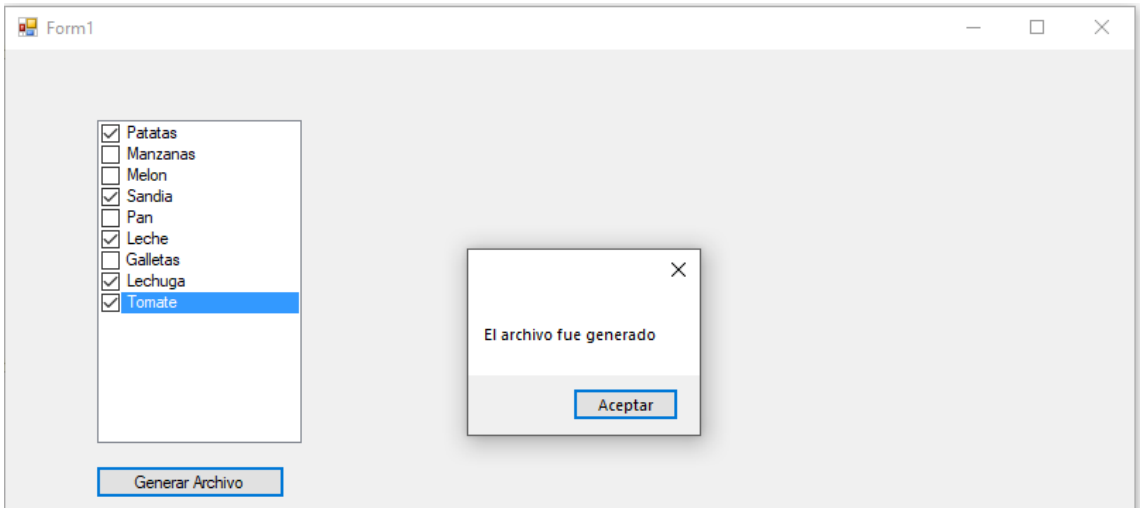


El siguiente paso es que cuando se presione el botón genere un nuevo archivo con los artículos que previamente hemos seleccionado.

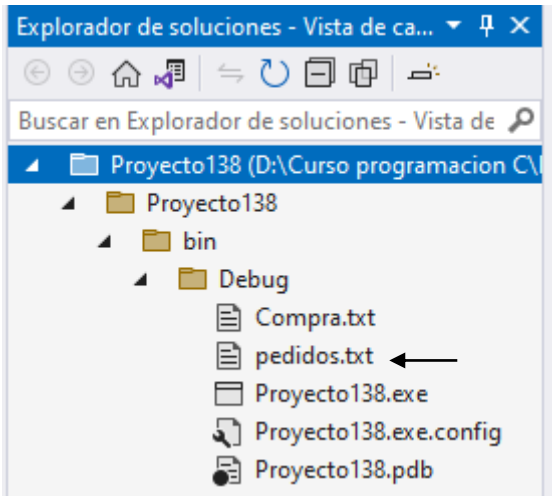
Vamos a programar el botón "Generar Archivo" el evento click.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    StreamWriter archivo = new StreamWriter(Directory.GetCurrentDirectory() + "\\pedidos.txt");
    for(int f= 0; f< checkedListBox1.Items.Count; f++)
    {
        if (checkedListBox1.GetItemChecked(f))
        {
            archivo.WriteLine(checkedListBox1.Items[f].ToString());
        }
    }
    archivo.Close();
    MessageBox.Show("El archivo fue generado");
}
```

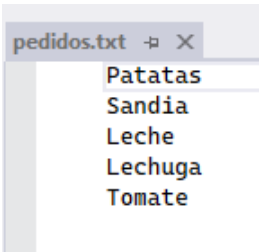
Vamos a ejecutar, seleccionar los elementos que deseamos comprar y generamos el archivo.



Buscamos el archivo pedidos:

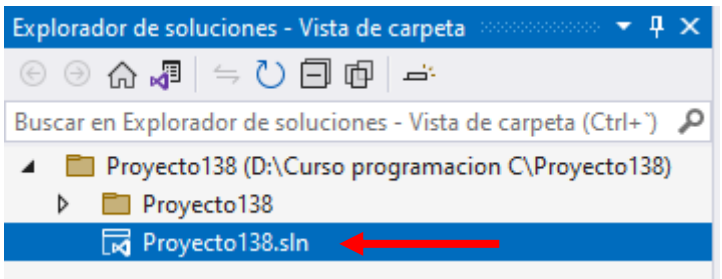


Doble clic para abrirlo.



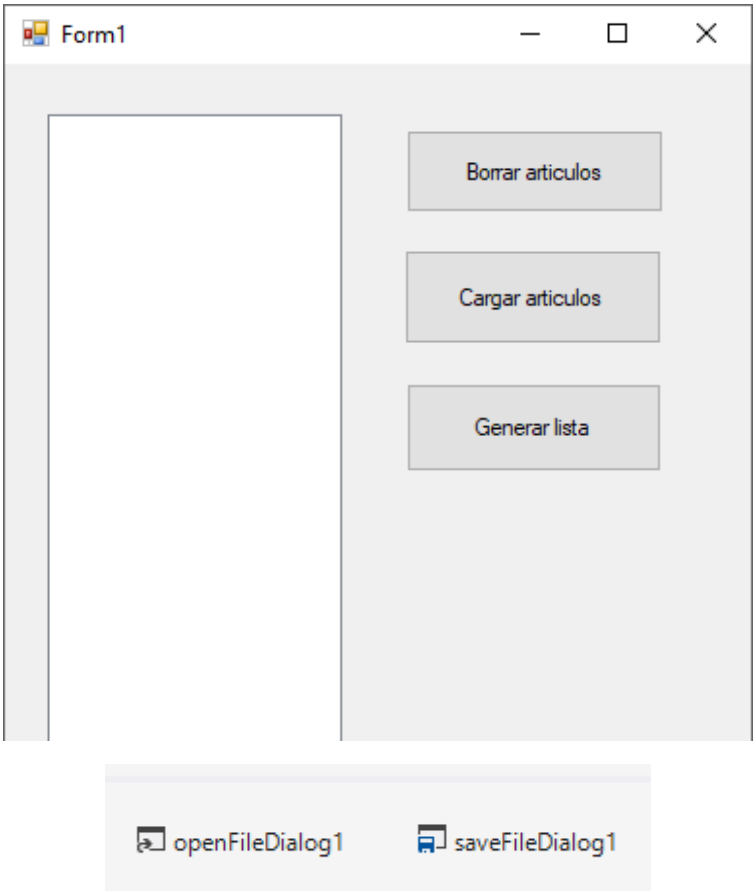
Son los Items que habíamos seleccionado.

Recuerda que cuando abrimos un archivo que queremos continuar con el tenemos que hacer doble click al archivo.

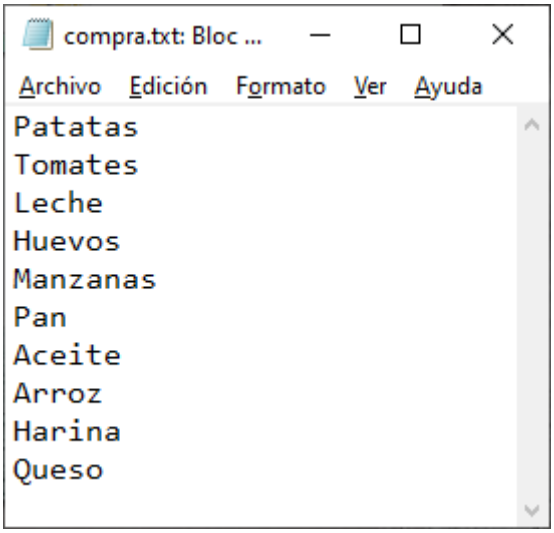


Propuesta para mejorar el proyecto anterior

Modifica el formulario de la siguiente manera:



El botón "Borrar artículos" irá eliminando el primer item de la lista mientras tenga elementos.
El botón "Cargar Artículos" nos abrirá el control de dialogo OpenFileDialog para buscar en nuestro ordenador un archivo txt que contenga una lista.
El botón "Generar lista" nos abrirá el control de dialogo SaveFileDialog para decir donde queremos guardar el archivo y con qué nombre.
A continuación antes de ejecutar el programa crea el siguiente archivo de texto que guardaremos en nuestro Escritorio.



Lo guardaremos con el nombre compra.txt.

El código del botón "Borrar artículos" con el evento Click.

```
private void button1_Click(object sender, EventArgs e)
{
    if(checkedListBox1.Items.Count > 0)
    {
        checkedListBox1.Items.RemoveAt(0);
    }
}
```

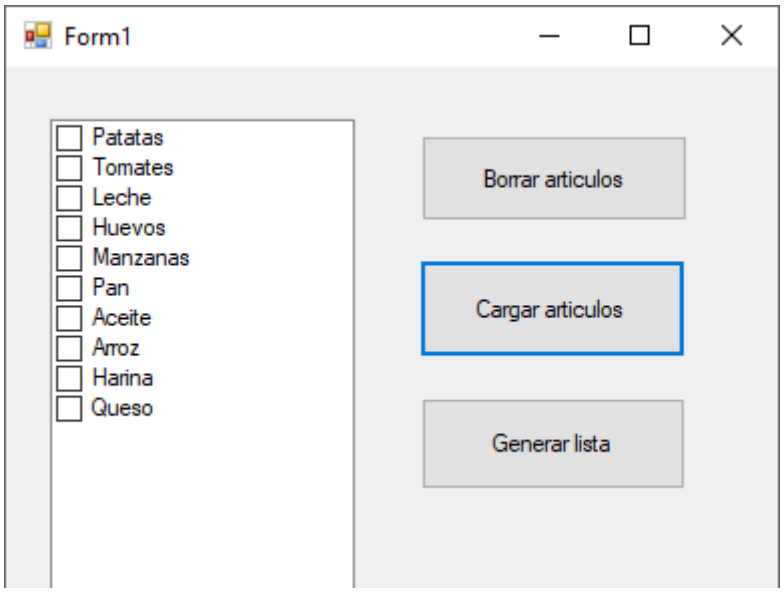
El código del botón "Cargar artículos" con el evento Click.

```
private void button2_Click(object sender, EventArgs e)
{
    if(openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        StreamReader archivo = new StreamReader(openFileDialog1.FileName);
        while (!archivo.EndOfStream)
        {
            string linea = archivo.ReadLine();
            checkedListBox1.Items.Add(linea);
        }
        archivo.Close();
    }
}
```

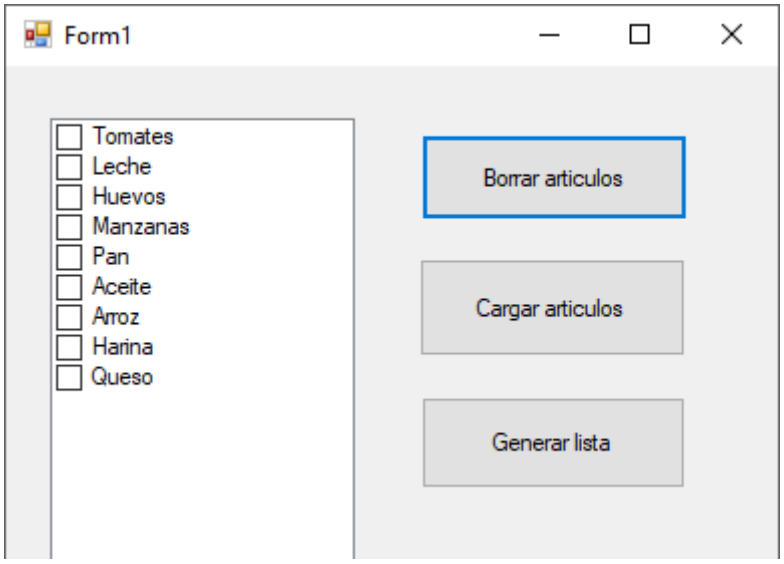
El código del botón "Generar lista" con el evento Click.

```
private void button3_Click(object sender, EventArgs e)
{
    if(saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        StreamWriter archivo = new StreamWriter(saveFileDialog1.FileName);
        for (int f=0; f<checkedListBox1.Items.Count; f++)
        {
            if(checkedListBox1.GetItemChecked(f))
            {
                archivo.Write(checkedListBox1.Items[f].ToString() + "\n");
            }
        }
        archivo.Close();
        MessageBox.Show("Se ha generado la lista");
    }
}
```

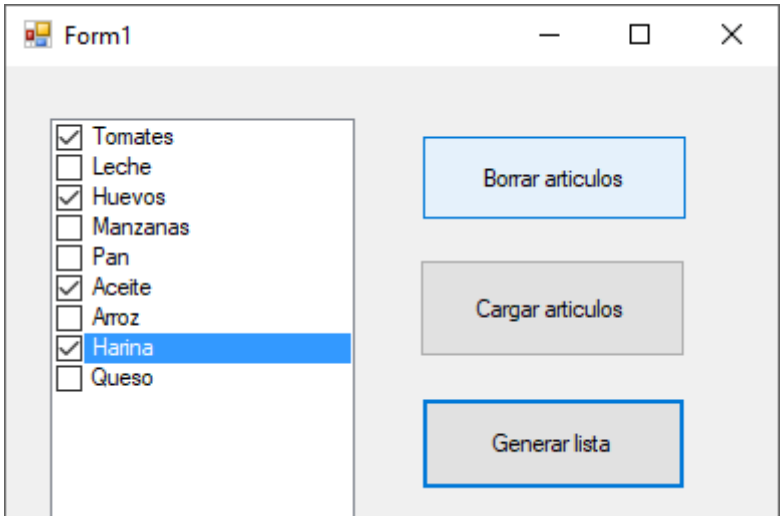
Vamos a ejecutar y seleccionar "Cargar Lista" para cargar el archivo "compra.txt".



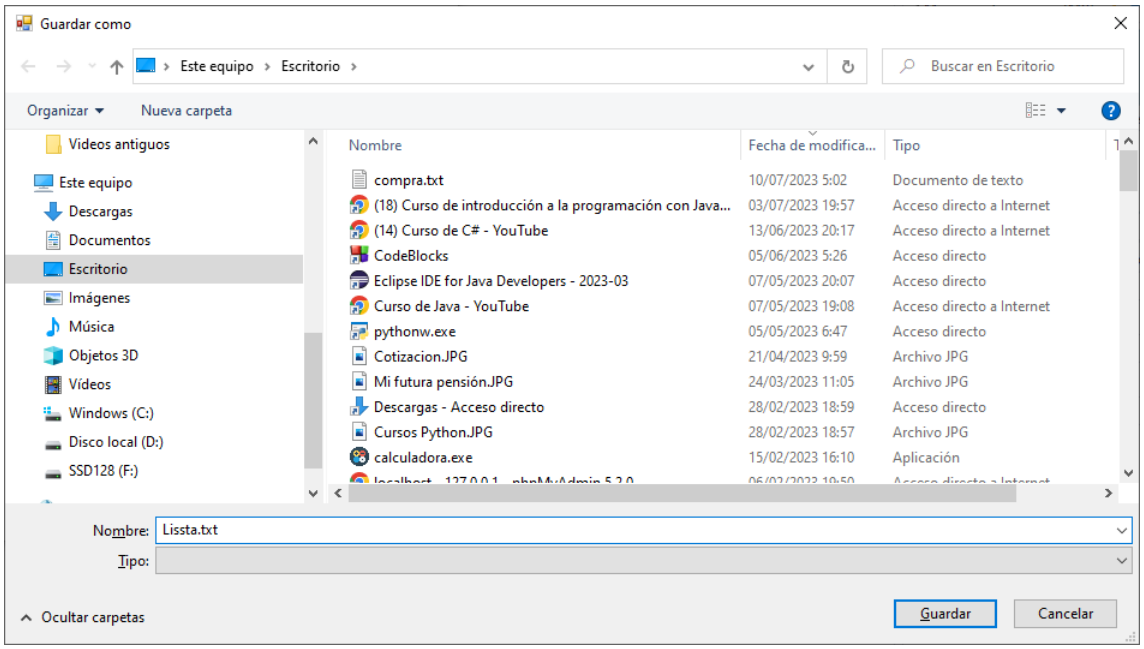
Borramos el artículo Patatas.



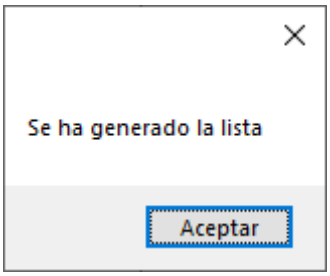
Seleccionamos cuatro artículos para generar lista.



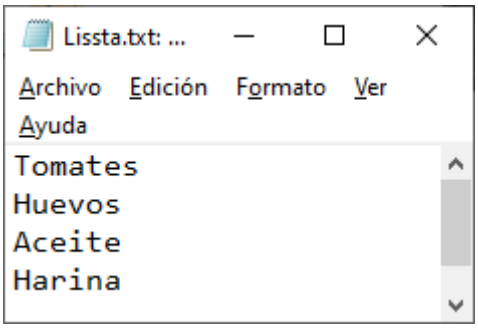
Le damos al botón generar lista.



Lo guardamos en el Escritorio con el nombre de lista.txt, seguido del botón guardar.

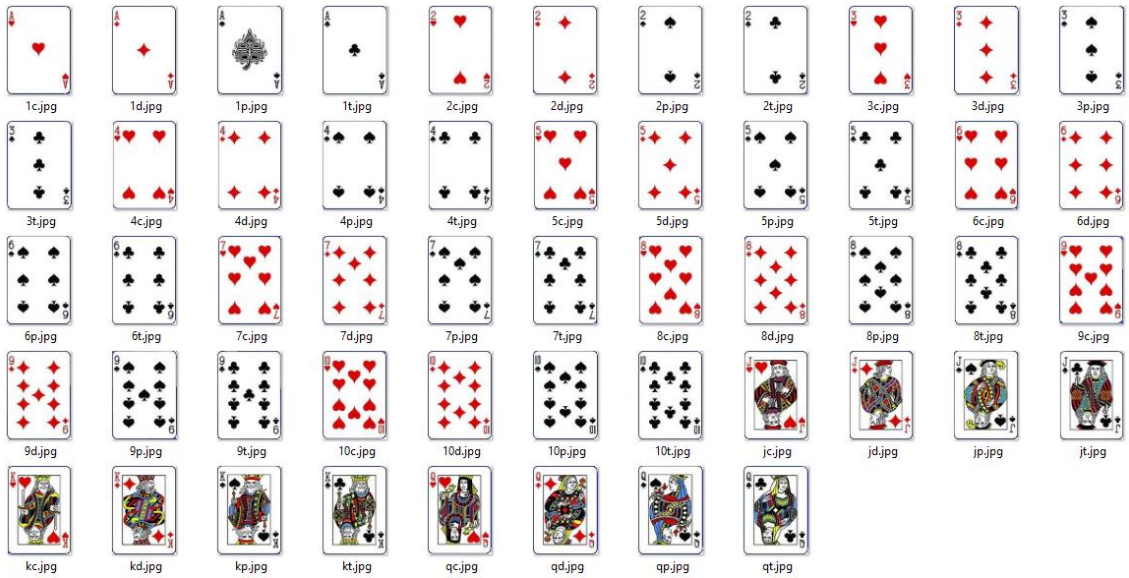


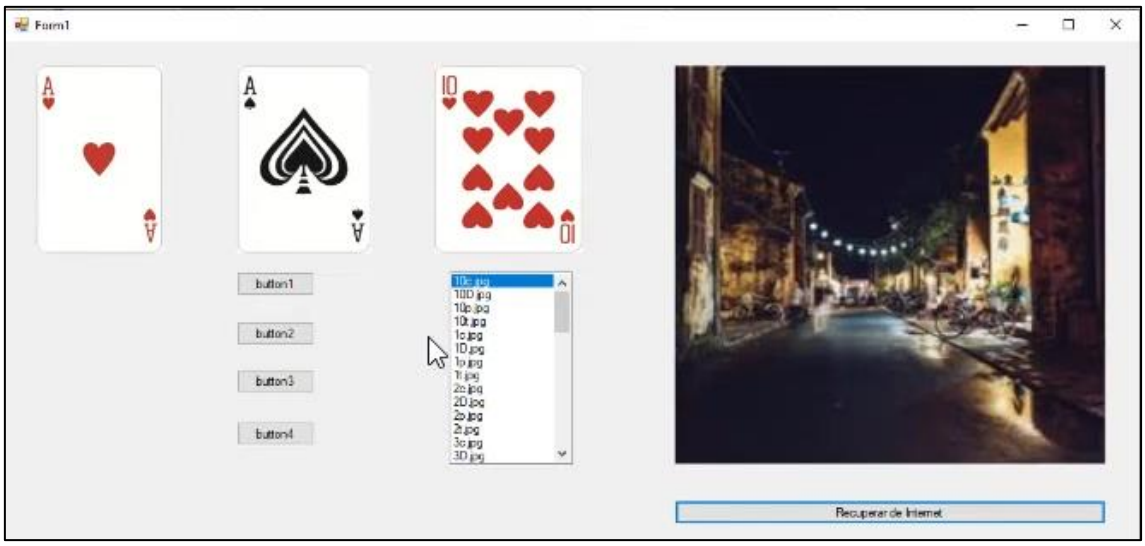
Se genero el archivo, ahora vamos a buscar el archivo lista.txt para ver su contenido.

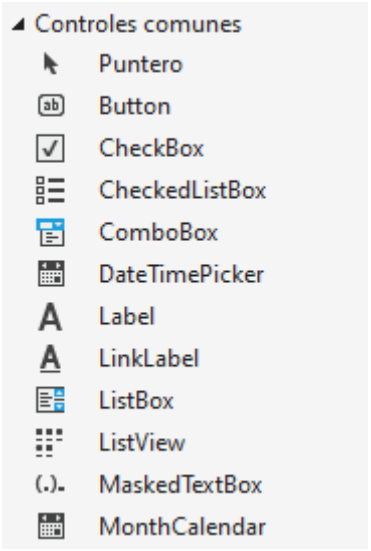


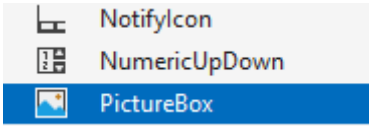
Capítulo 129.- Control visual PictureBox (Windows Forms)

Para este capítulo necesitamos todas las imágenes de la baraja de poker.

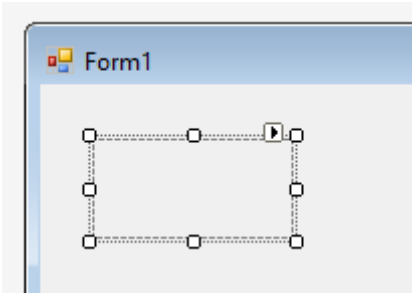




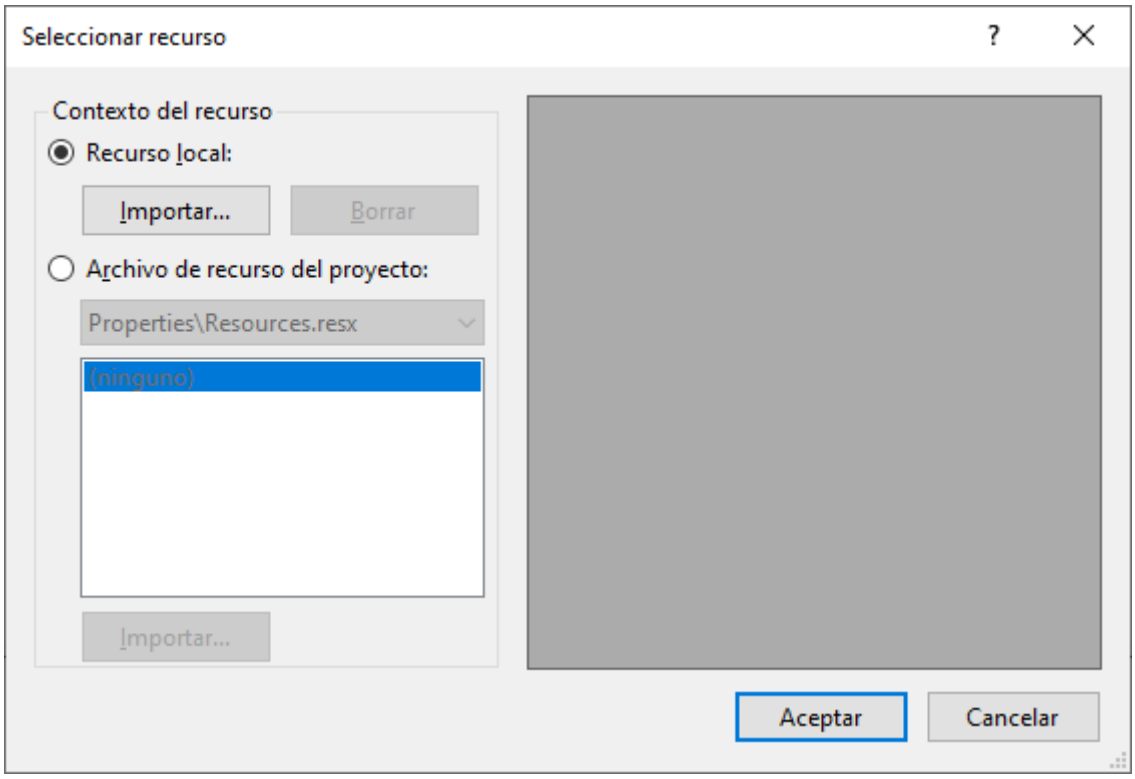




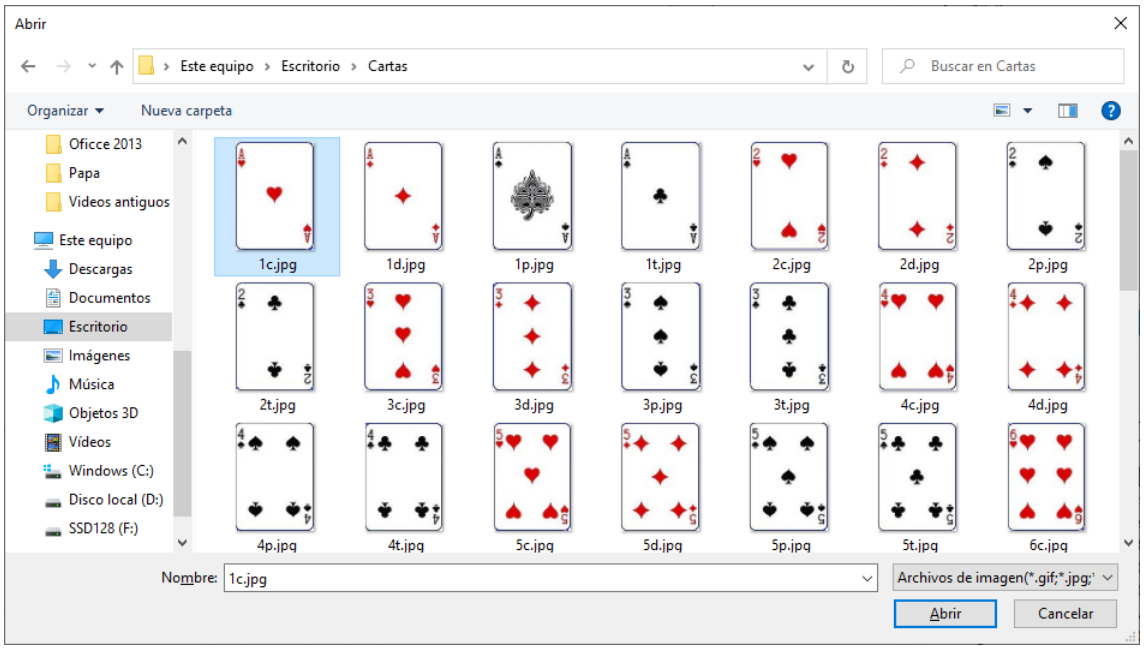
Lo encontraremos en Controles comunes.



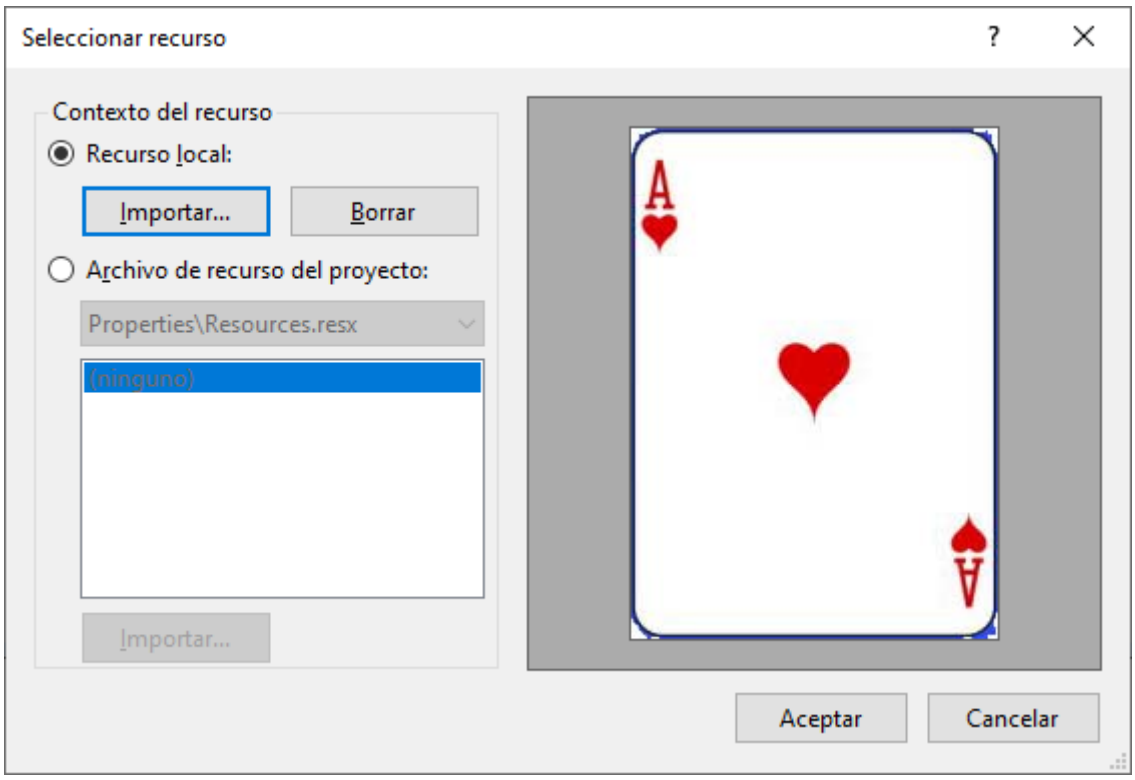
Ahora iremos a propiedades y seleccionaremos Image.



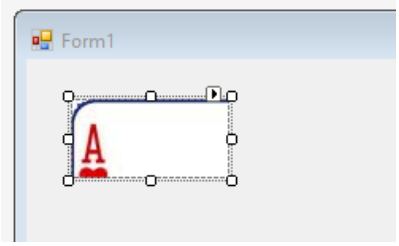
Seleccionaremos Recurso local seguido del botón importar iremos a la carpeta donde tenemos las imágenes.



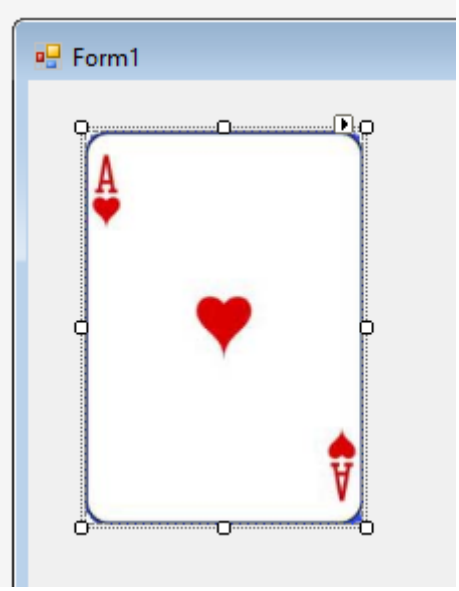
Seleccionamos la imagen seguido del botón abrir.



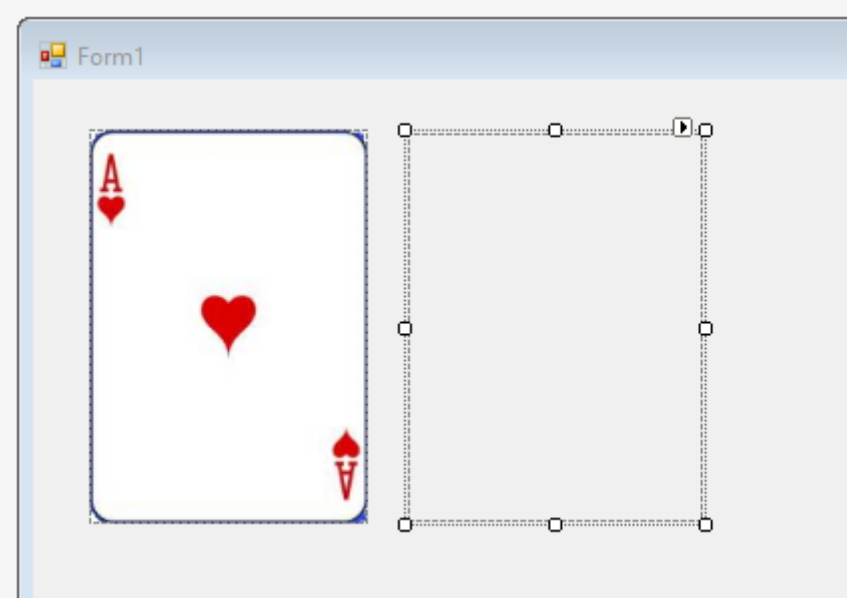
A continuación le damos a aceptar.



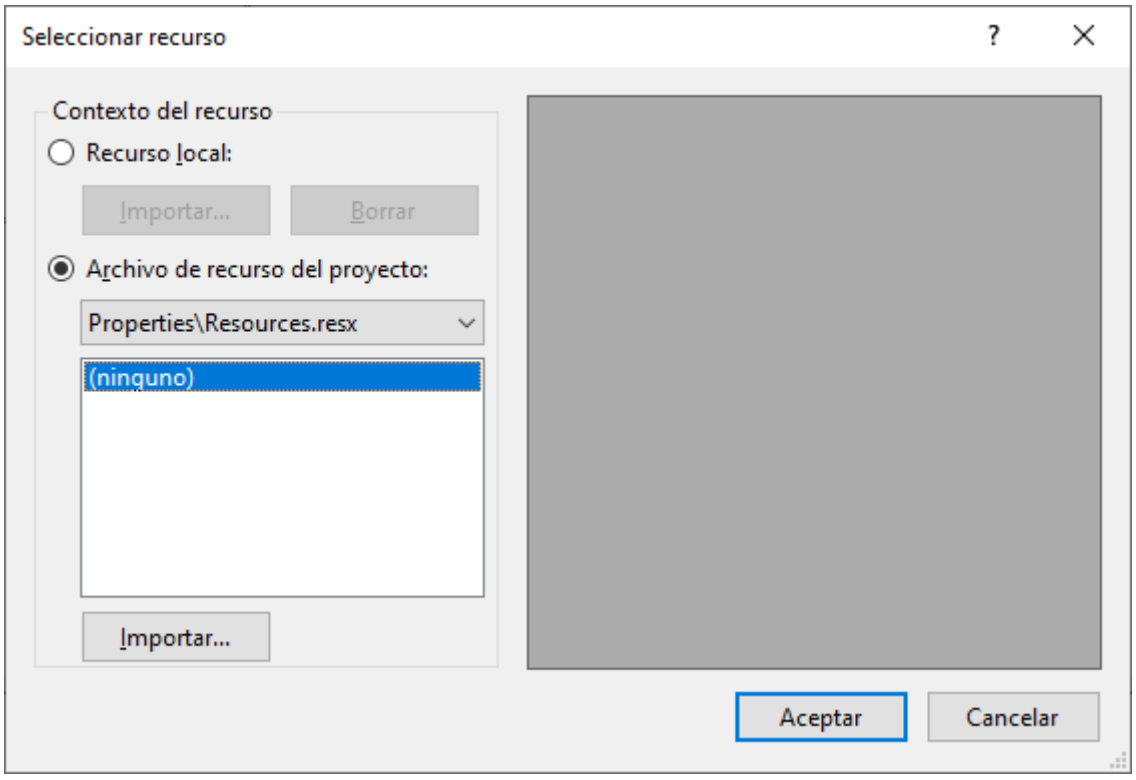
Vamos a seleccionar la propiedad SizeMode y seleccionaremos StretchImage, esto hará que la imagen se ajuste al control Image.



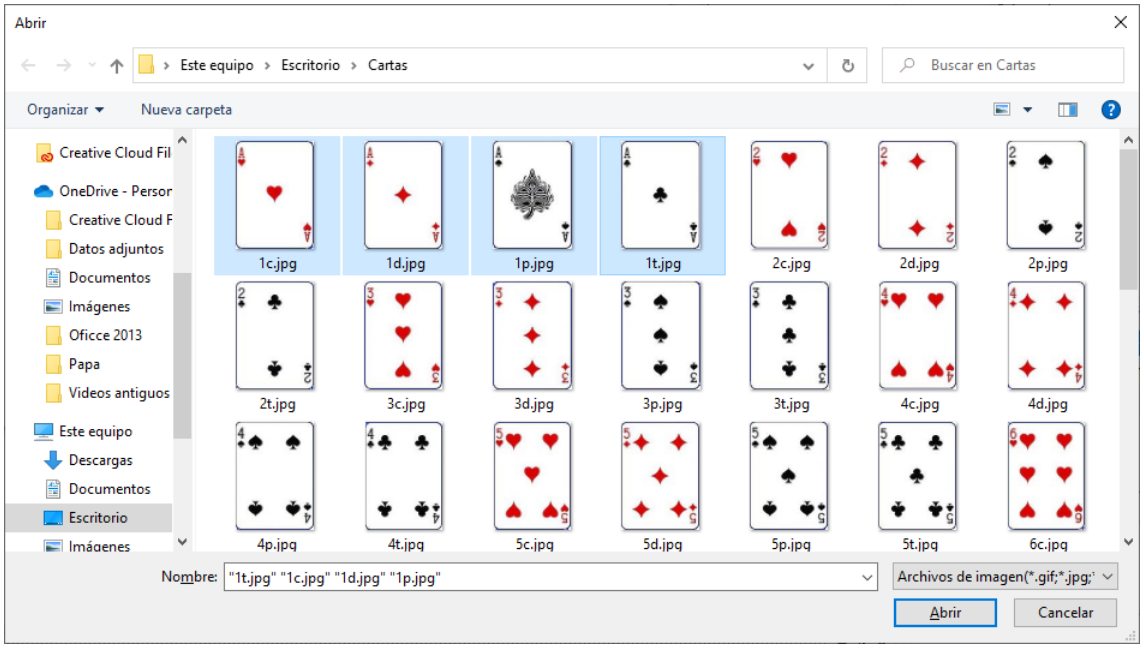
Vamos a crear otro control de image para ver otra forma de importar.



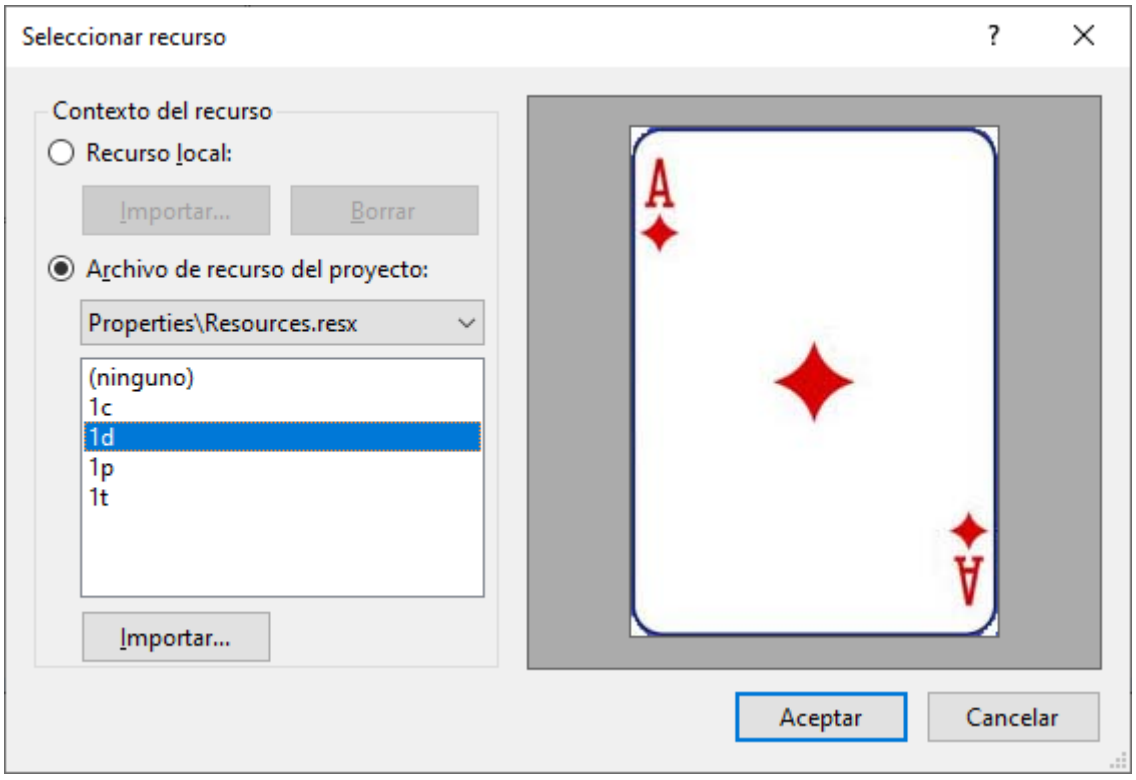
Iremos de nuevo a propiedades y seleccionaremos Image.



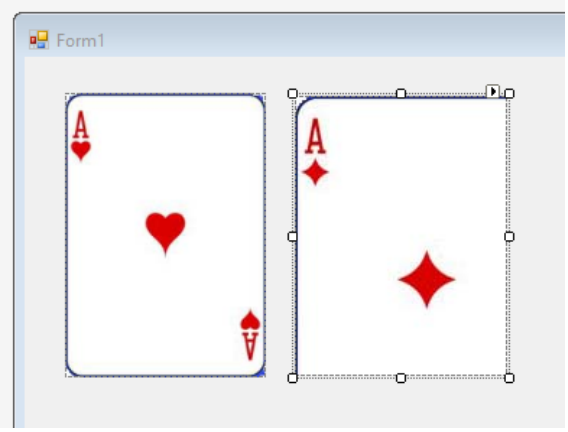
Dejaremos activo Archivo de recurso del proyecto y le daremos al botón importar.



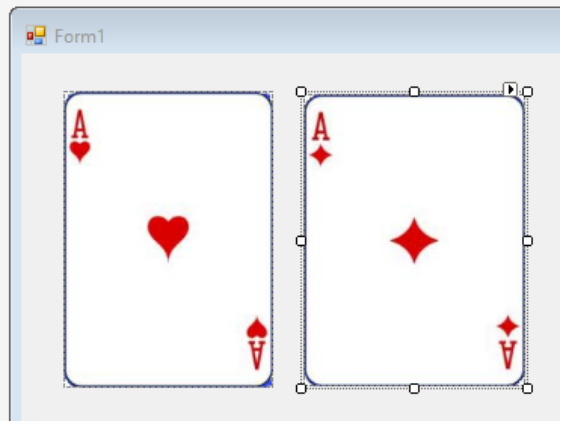
Importamos las 4 imágenes seleccionadas, seguido del botón Abrir.

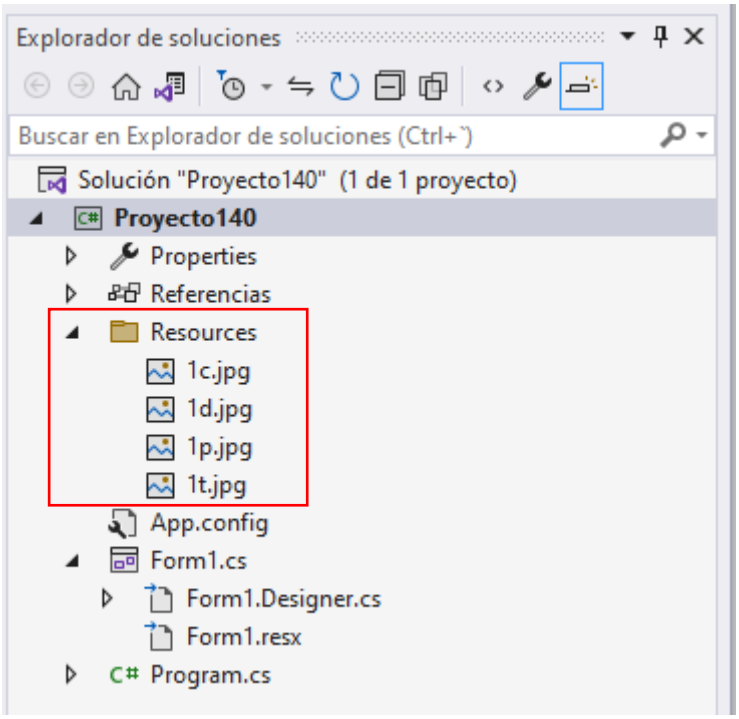


De las imágenes que hemos importado seleccionamos una seguido del botón Aceptar.



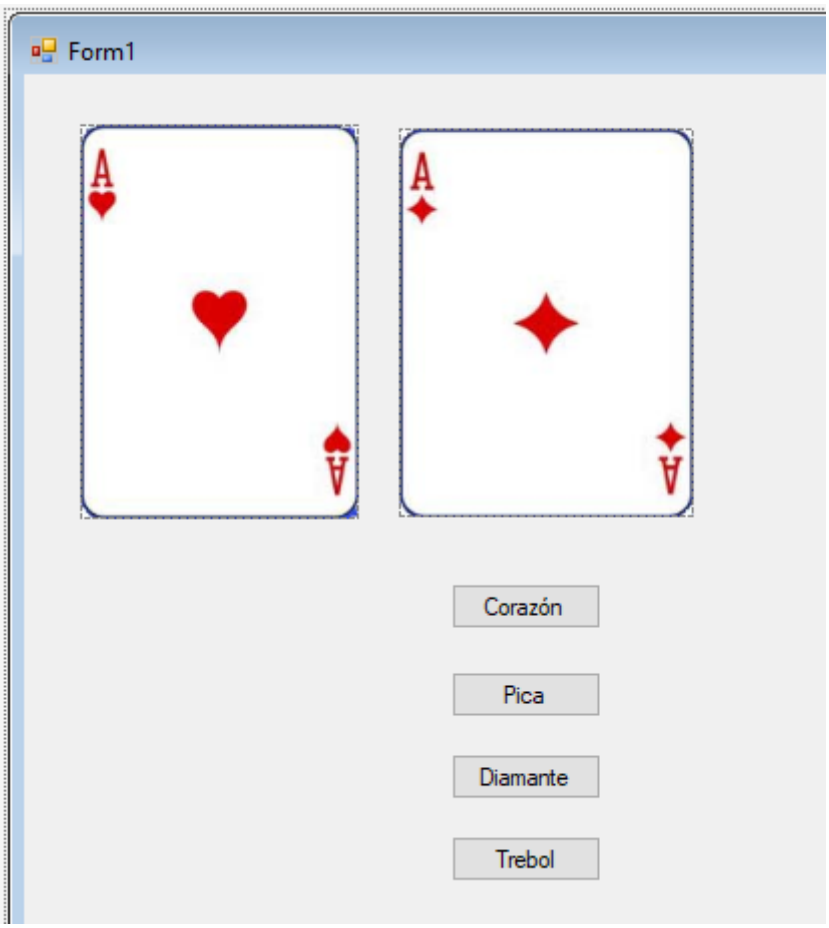
Tenemos que ajustar la imagen al tamaño del control PictureBox seleccionando de propiedades SizeMode y seleccionaremos StretchImage.





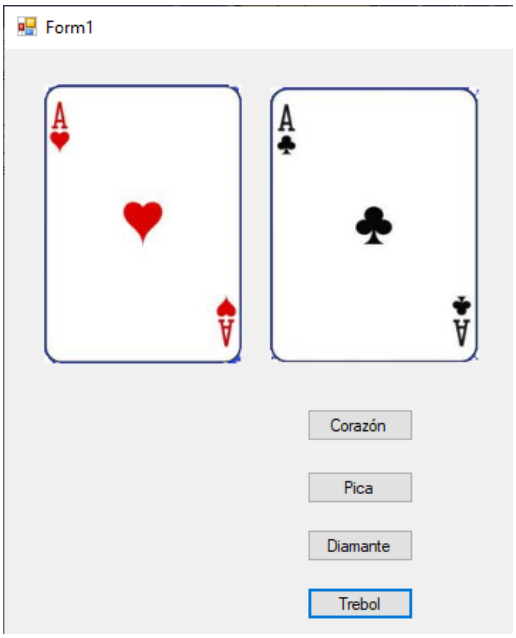
En el explorador de soluciones podemos observar como se ha creado una carpeta con las imágenes que hemos importado.

A continuación vamos a agregar 4 botones y cada botón mostrará la forma de visualizar las imágenes.



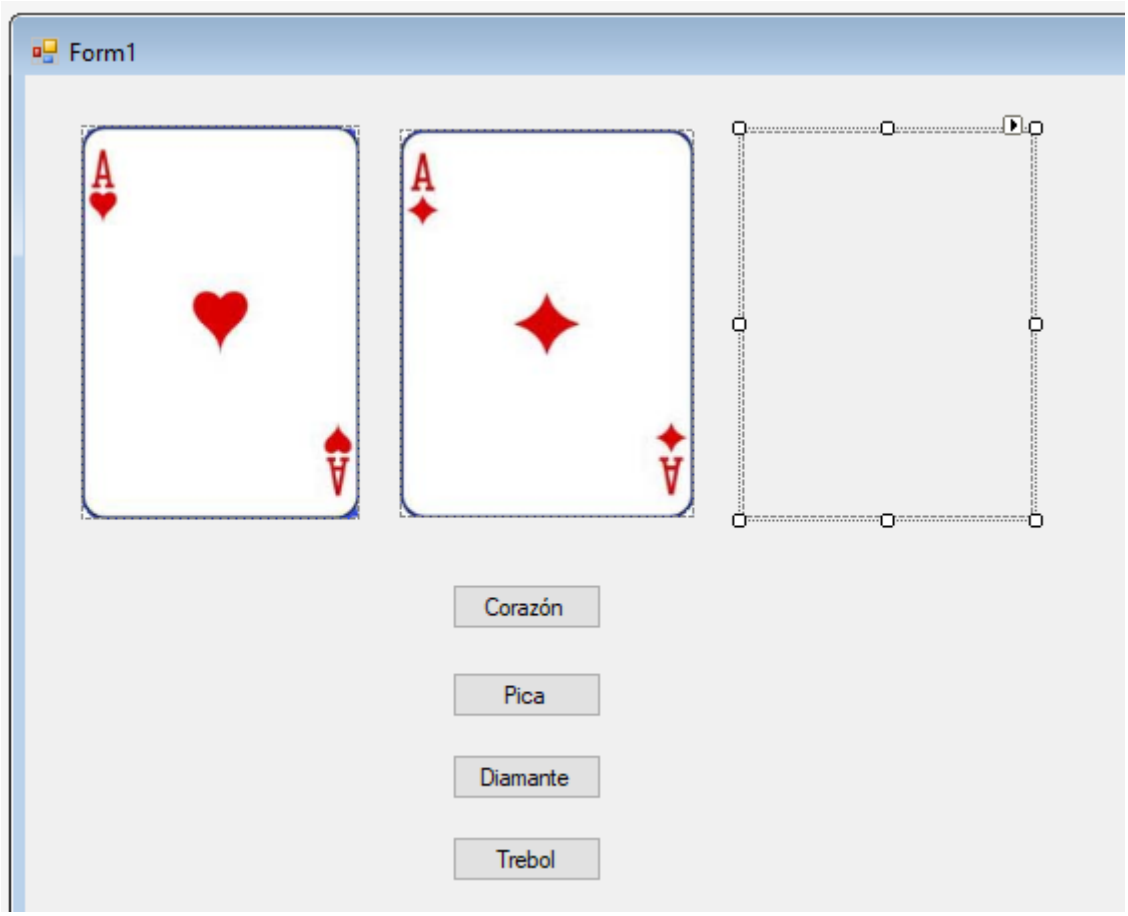
Vamos a programar los cuatro botones en el evento Click.

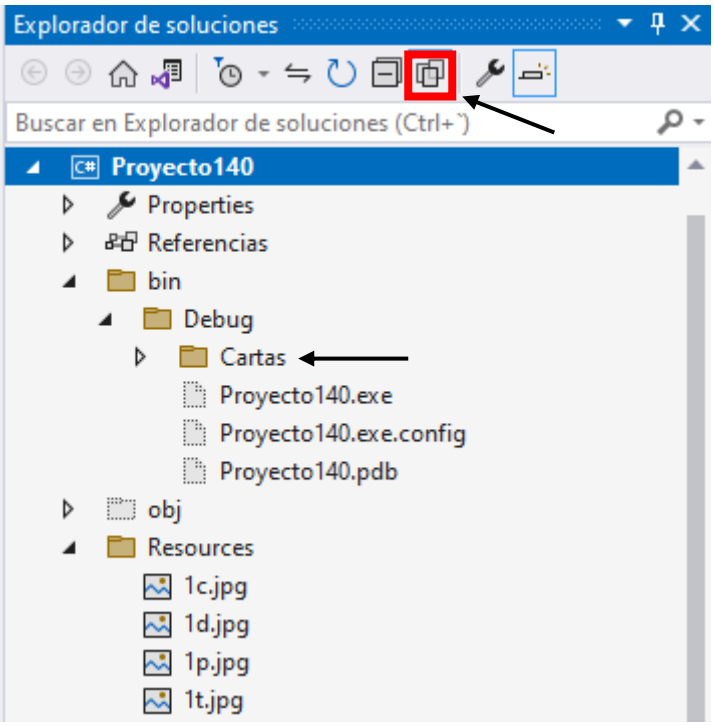
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    pictureBox2.Image = Properties.Resources._1c;
}

1 referencia
private void button2_Click(object sender, EventArgs e)
{
    pictureBox2.Image = Properties.Resources._1p;
}

1 referencia
private void button3_Click(object sender, EventArgs e)
{
    pictureBox2.Image = Properties.Resources._1d;
}

1 referencia
private void button4_Click(object sender, EventArgs e)
{
    pictureBox2.Image = Properties.Resources._1t;
}
```



Según el botón que seleccionemos mostrará una imagen u otra.

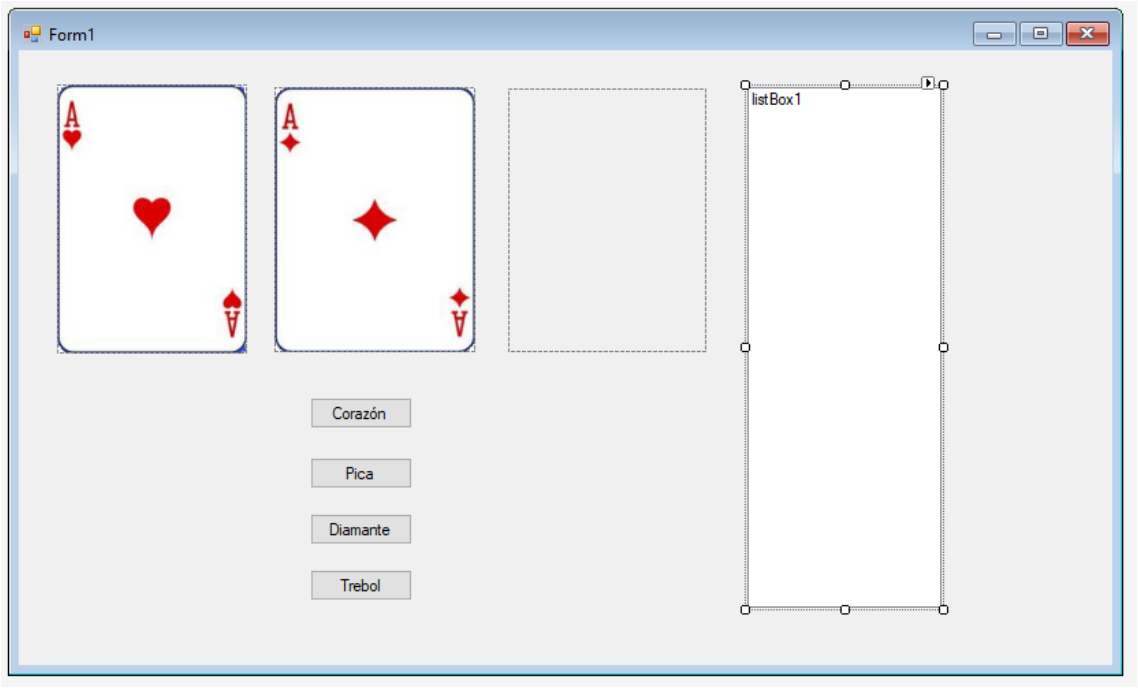
Vamos a agregar un tercer PictureBox.





Seleccionamos el botón ver todos los archivos y arrastramos la carpeta que contiene las Cartas dentro de la carpeta Debug.

A continuación vamos a cargar a nuestro formulario un control listBox.



A continuación lo que haremos será cargar todas las imágenes. Recuerda modificar la propiedad:

SizeMode	StretchImage

Para ello vamos a seleccionar el formulario y programaremos el evento Load.

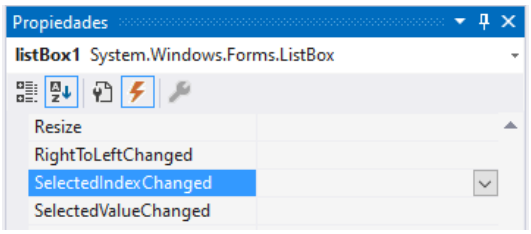
Pero antes en la parte superior vamos a importar:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
```

Ahora programamos el evento.

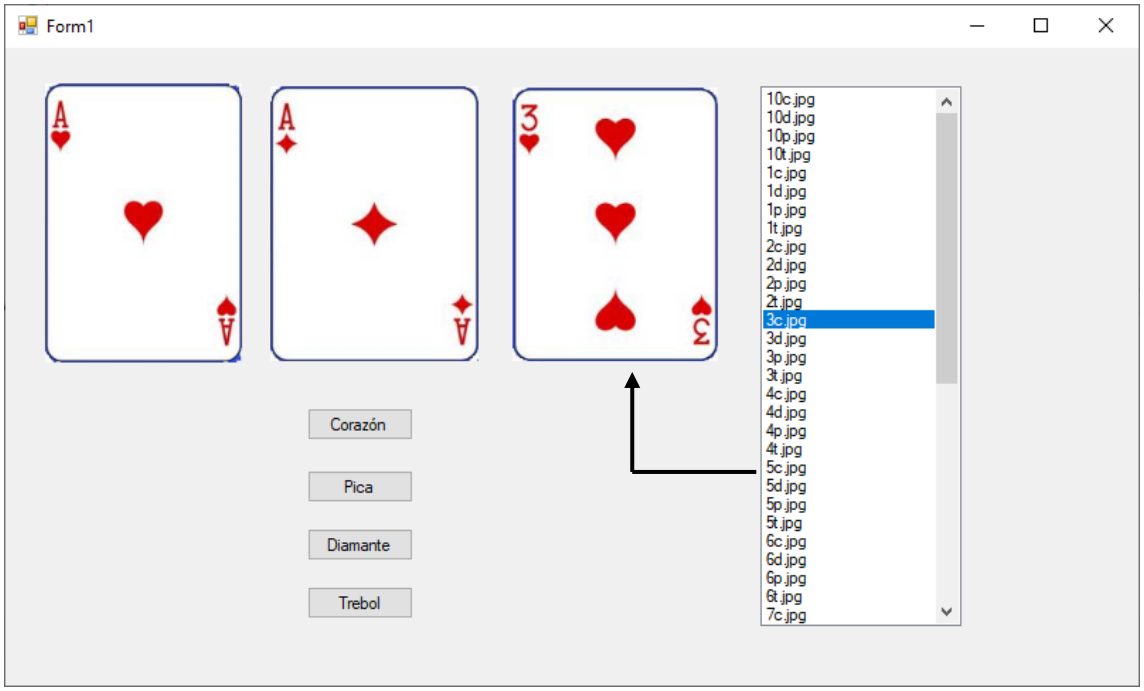
```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    DirectoryInfo d = new DirectoryInfo("cartas/");
    FileInfo[] archivos = d.GetFiles("*.jpg");
    for (int f=0; f<archivos.Length; f++)
    {
        listBox1.Items.Add(archivos[f].Name);
    }
}
```

Ahora para el evento seleccione un item del control listBox.

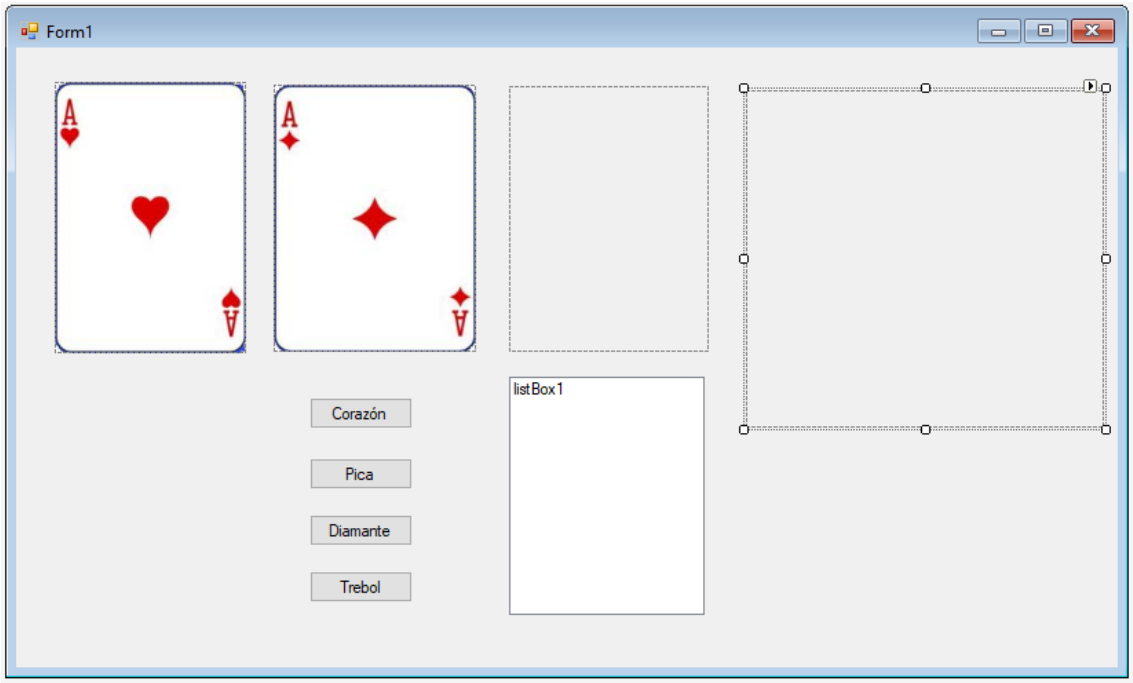



```
1 referencia
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    pictureBox3.Image = Image.FromFile("cartas\\"+listBox1.SelectedItem.ToString());
}
```

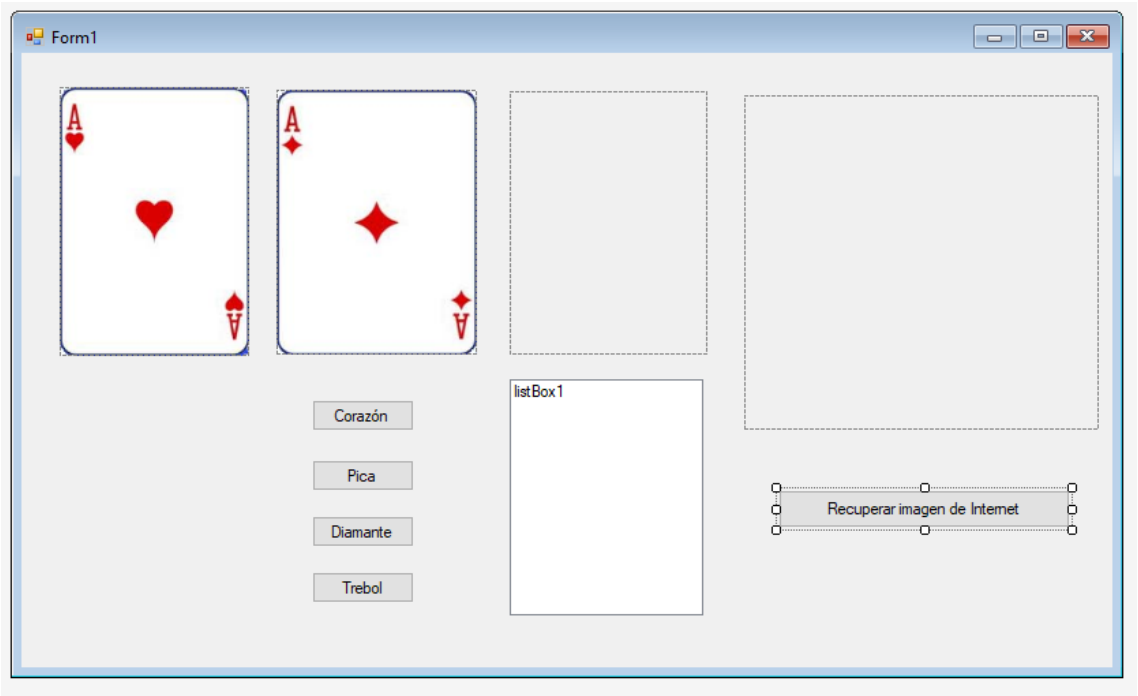
Cuando ejecutemos:



La imagen que seleccionemos se mostrará PictureBox3.



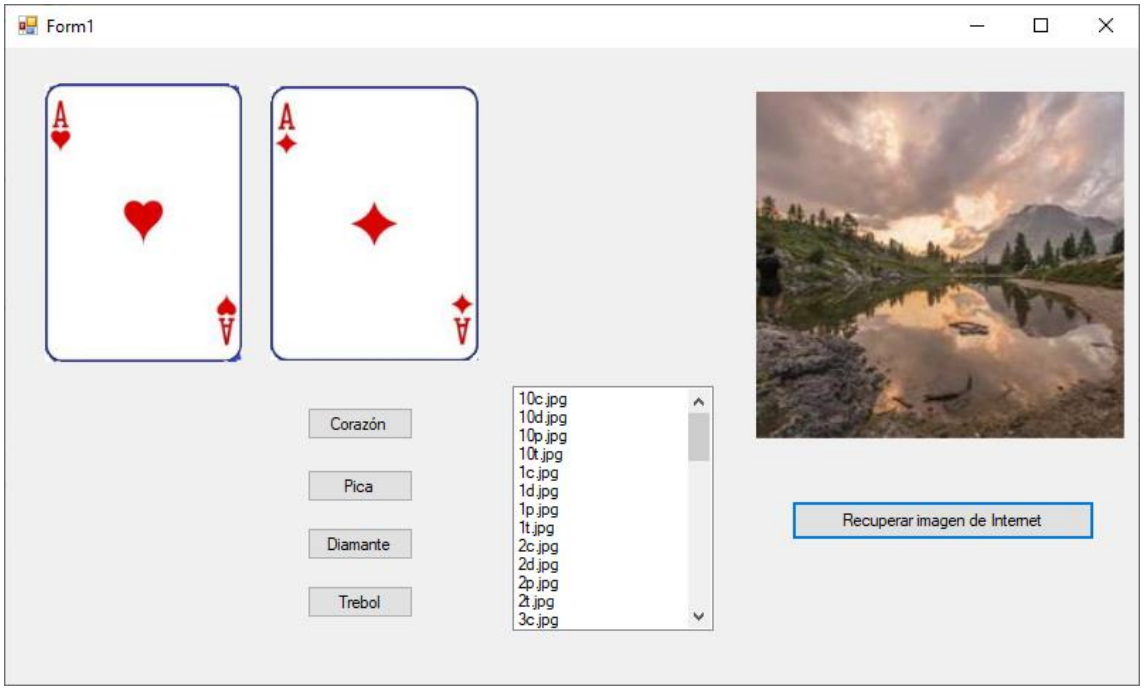
Agregamos un cuarto PictureBox para poder agregar una imagen desde internet.
Recuerda modificar la propiedad: SizeMode a StrechImage.



Agregamos un botón que ponga "Recuperar imagen de Internet".
Vamos a programa el evento Click.

```
1 referencia
private void button5_Click(object sender, EventArgs e)
{
    pictureBox4.ImageLocation = "http://picsum.photos/200";
}
```

La url que hemos puesto es donde se acceden a imágenes que van cambiando aleatoriamente.
Vamos a ejecutar y hacer click en el botón "Recuperar imagen de Internet".

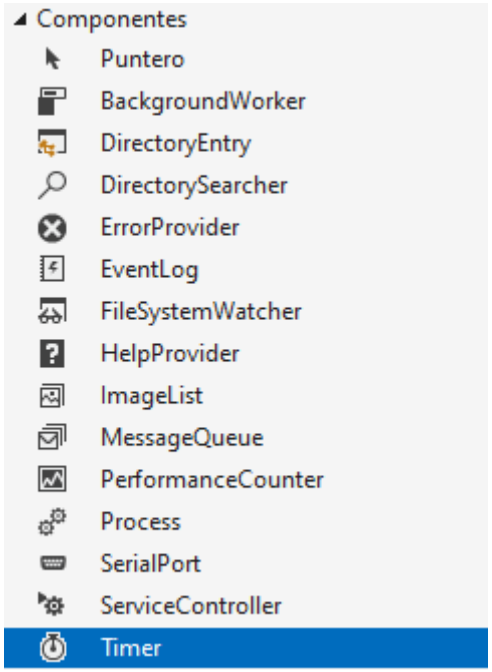


Capítulo 130.- Control Timer (Windows Forms)

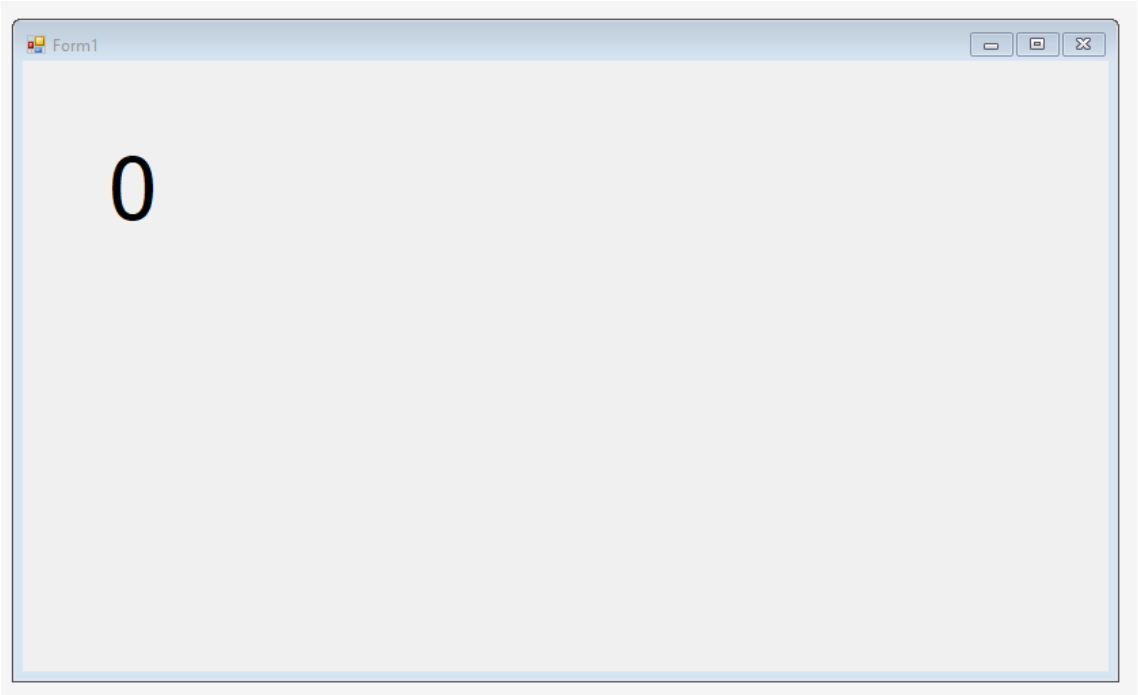
Vamos a realizar un contador de segundos, para ello vamos a agregar un Label.

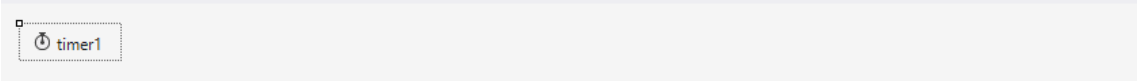
En la propiedad Text ponemos un 0.

En la propiedad Font aumentamos el tamaño de la fuente.

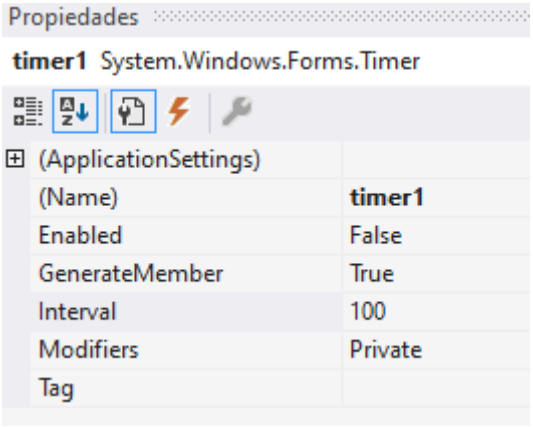


En la pestaña componentes se encuentra Timer.



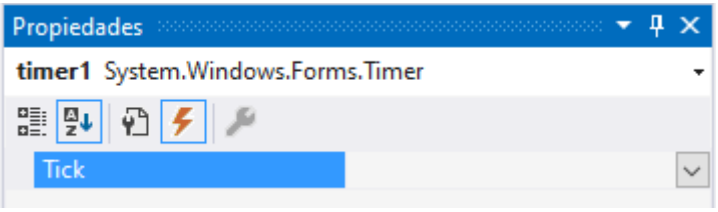


Las propiedades del objeto timer son:



En propiedades cambiamos Enabled a True e Interval a 1000 milisegundos.

Y como eventos solo tenemos el Tick.

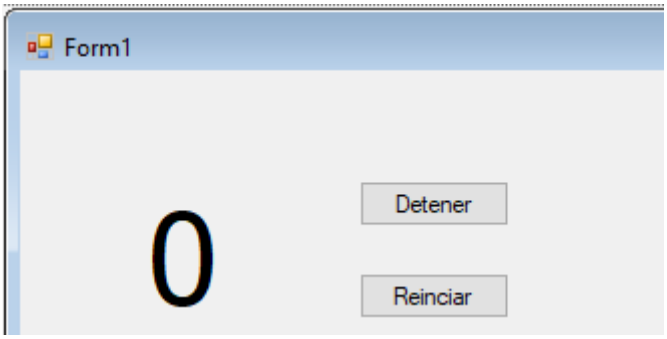


Ahora vamos a programar el evento Tick.

```
1 referencia
private void timer1_Tick(object sender, EventArgs e)
{
    int valor = int.Parse(label1.Text);
    valor++;
    label1.Text = valor.ToString();
}
```

Si ejecutamos vamos a comprobar como casa segundo el Laber se incrementa en 1.

Vamos a agregar dos botones una para detener el Timer y otro botón para reiniciar.



Vamos a programar el botón detener en el evento click.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;
}
```

Vamos a programar el botón Reiniciar en el evento click.

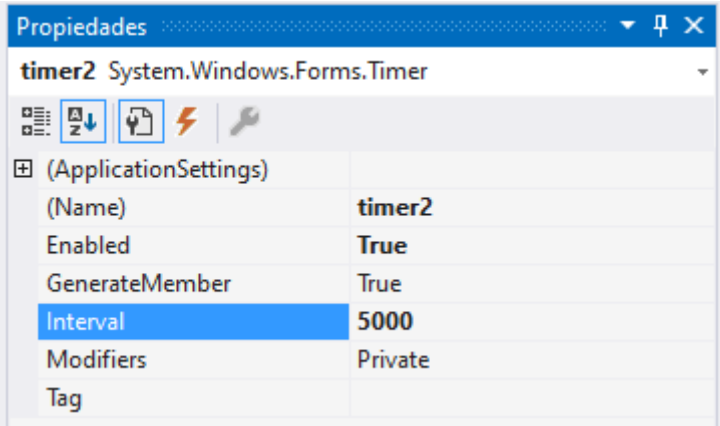
```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}
```

Ahora cuando ejecutemos podemos parar y reanudar el control timer.

Un formulario puede tener varios timer.

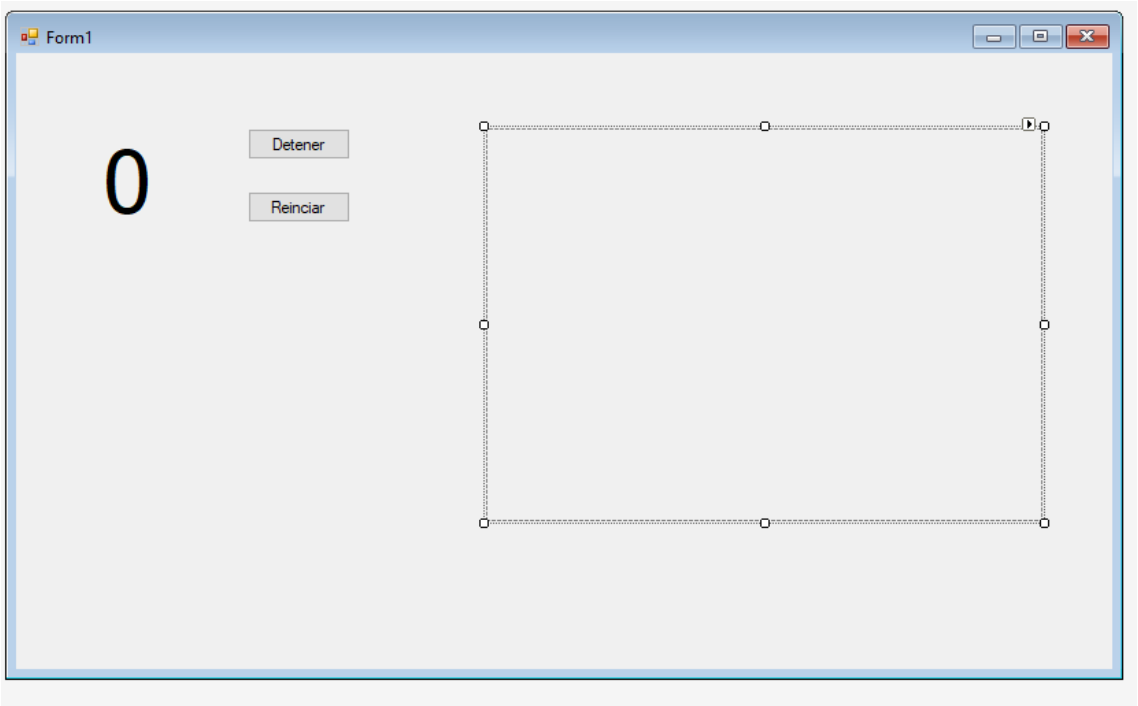
Vamos a agregar un segundo timer.

Con las siguientes propiedades:

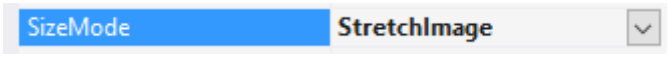


Enabled en True y que se ejecute cada 5 segundos, en lugar de 1 segundo que tiene el primer timer.

Vamos a agregar a nuestro formulario un PictureBox, para cargar una imagen desde internet.

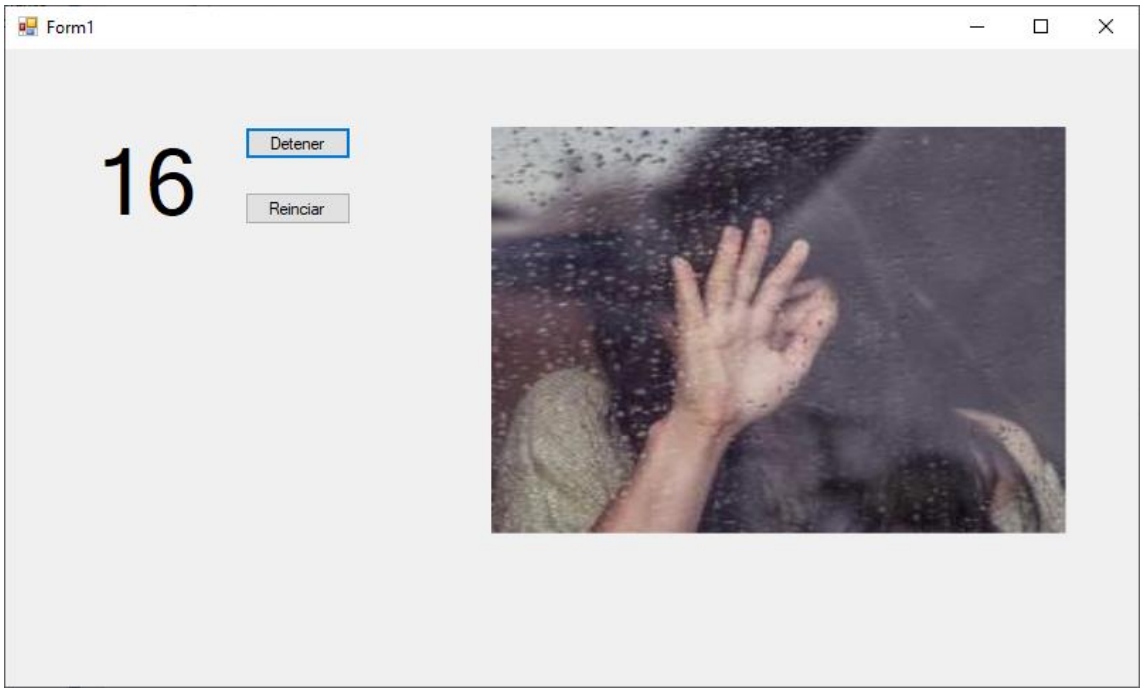


Modificamos la siguiente propiedad:



Vamos a programar el evento Tick del segundo timer.

```
1 referencia
private void timer2_Tick(object sender, EventArgs e)
{
    pictureBox1.ImageLocation = "https://picsum.photos/200";
}
```



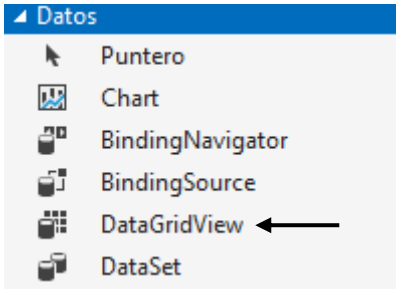
Si queremos que se cargue una imagen solo ejecutar el programa el evento Load del formulario podremos el siguiente código:

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    pictureBox1.ImageLocation = "https://picsum.photos/200";
}
```

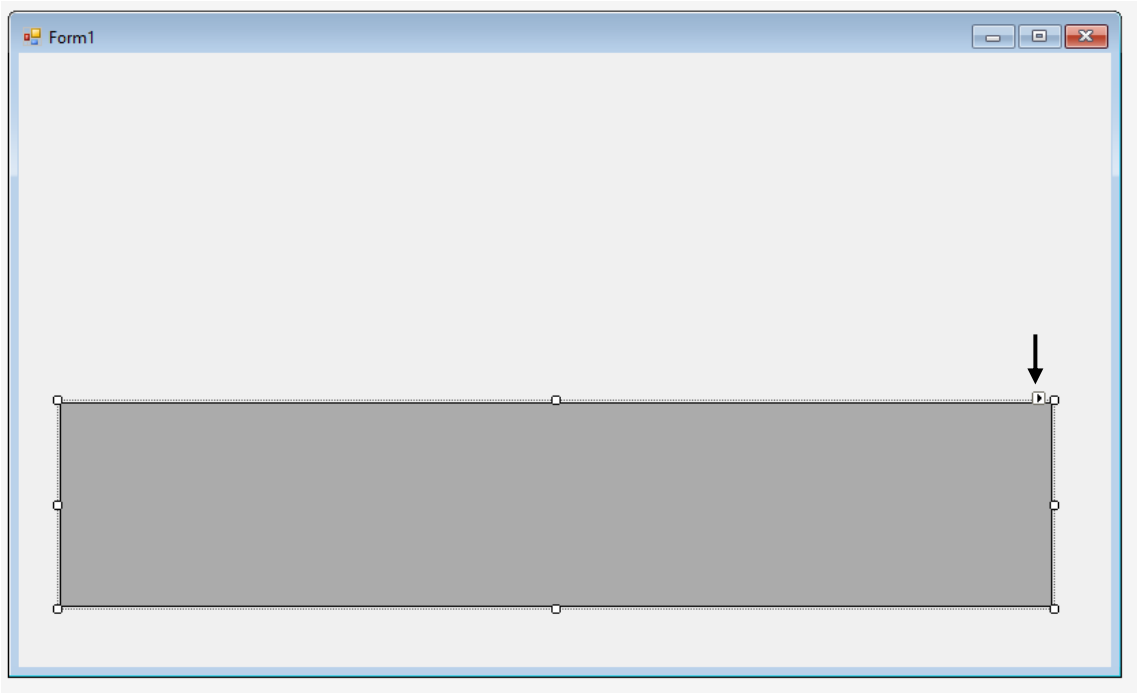
Ahora cuando ejecutemos ya aparecerá una imagen y no tendremos que esperar 5 segundos.

Capítulo 131.- Control DataGridView (Windows Forms)

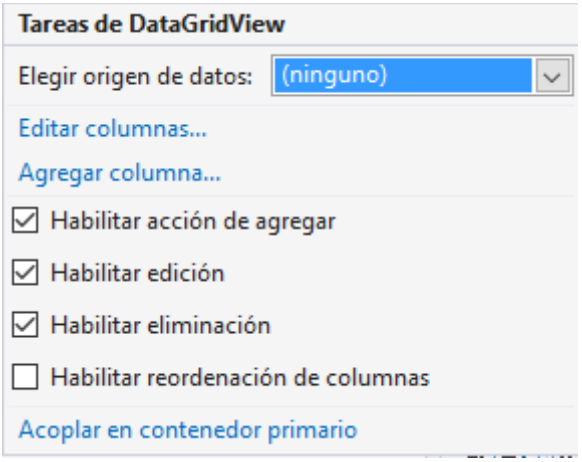
El control DataGridView proporciona una forma potente y flexible de mostrar datos en formato tabular. Puede usar el control DataGridView para mostrar vistas de solo lectura de una pequeña cantidad de datos, o puede escalarlo para mostrar vistas editables de conjunto de datos muy grandes.



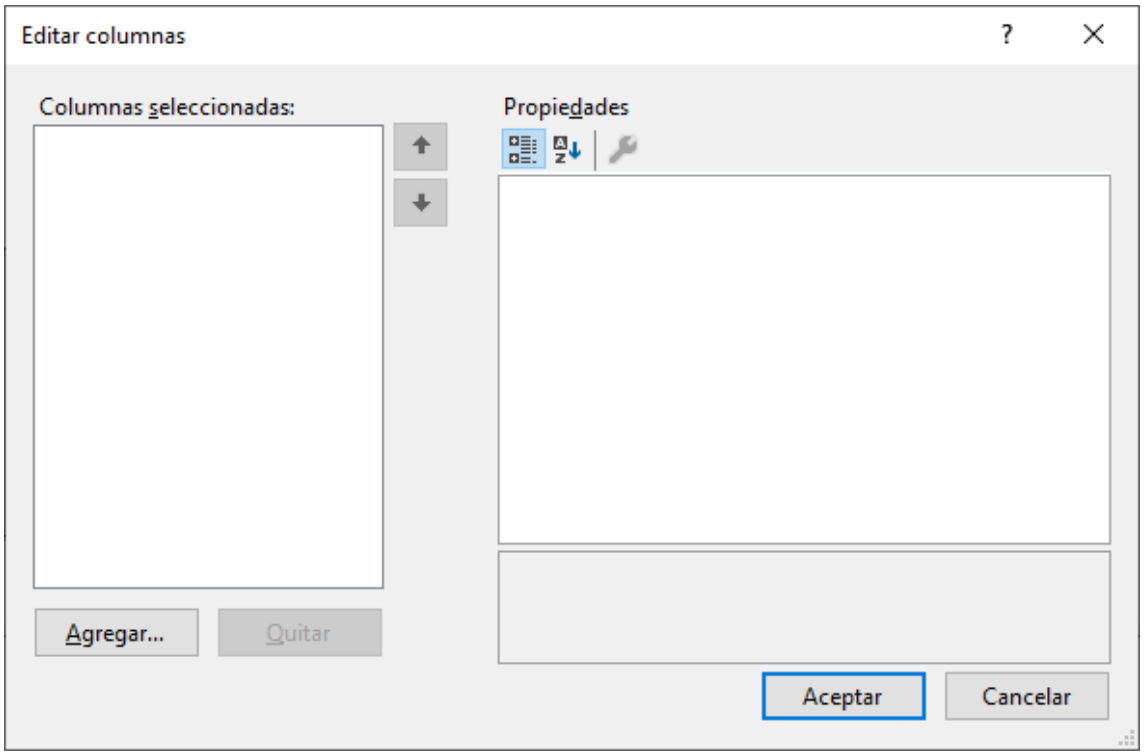
Lo encontramos en la pestaña Datos.



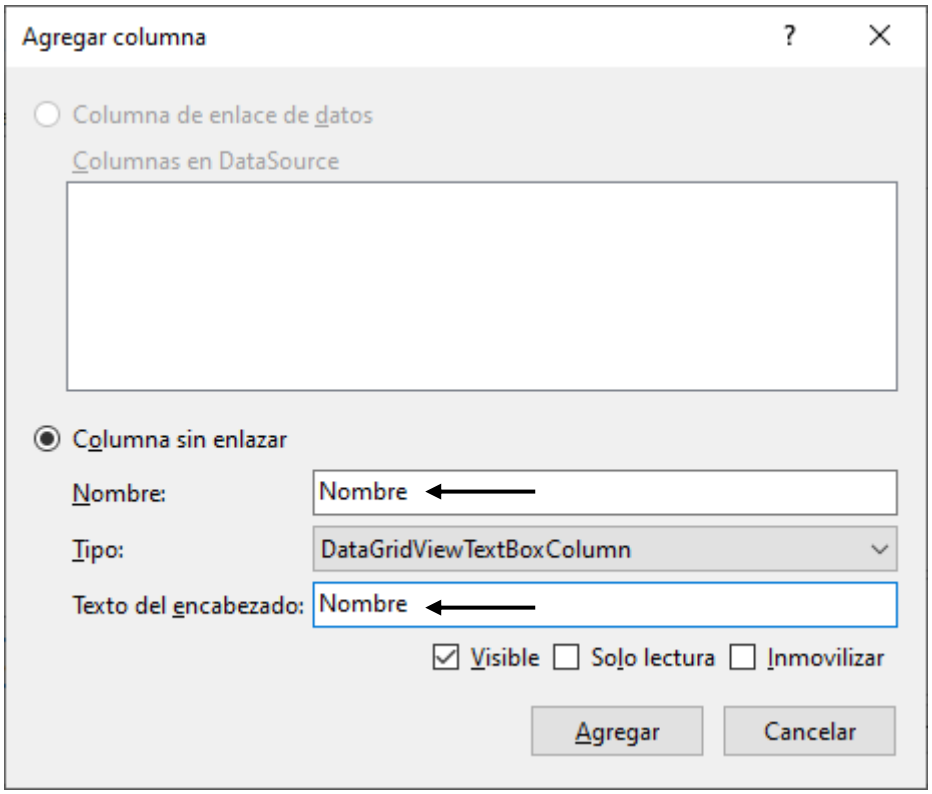
Si seleccionamos el triangulito de la derecha observaremos el siguiente menú:



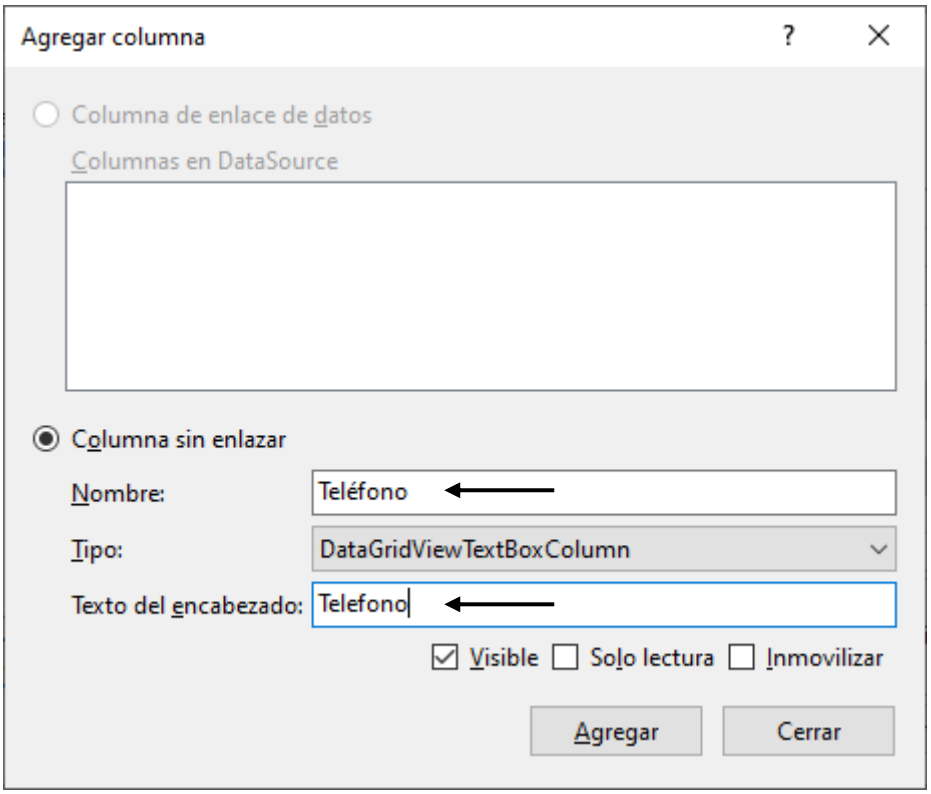
Le damos a la opción Editar columnas…

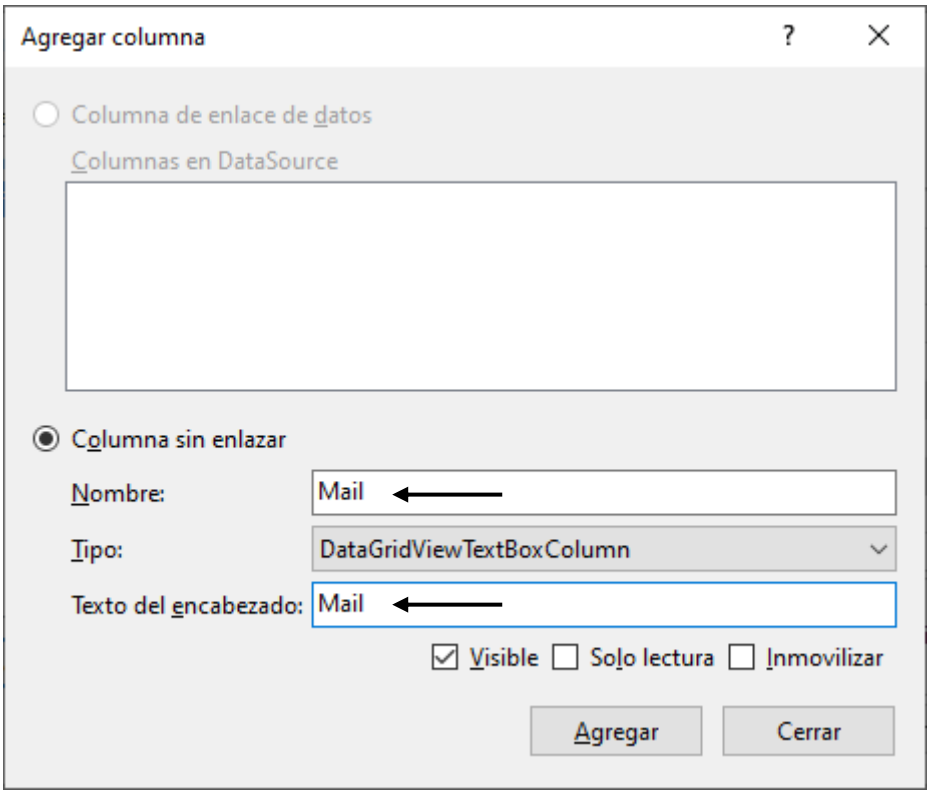


Le damos al botón Agregar…

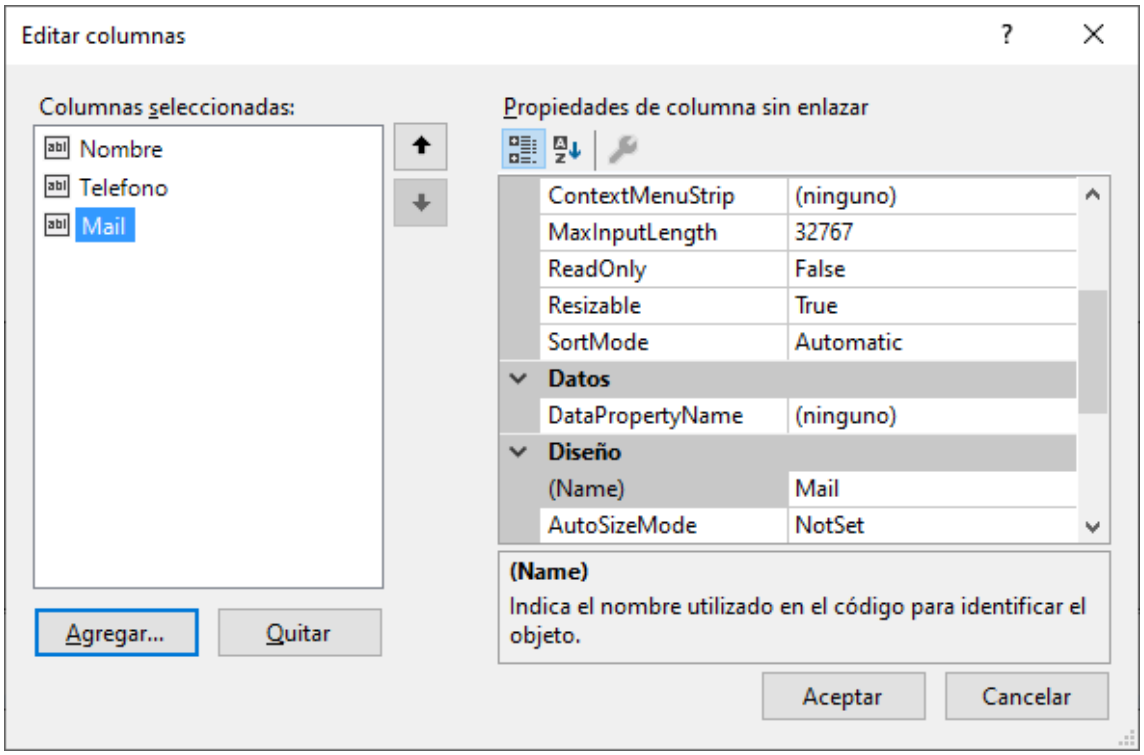


Le damos a agregar y lo repetimos con Teléfono y Mail.

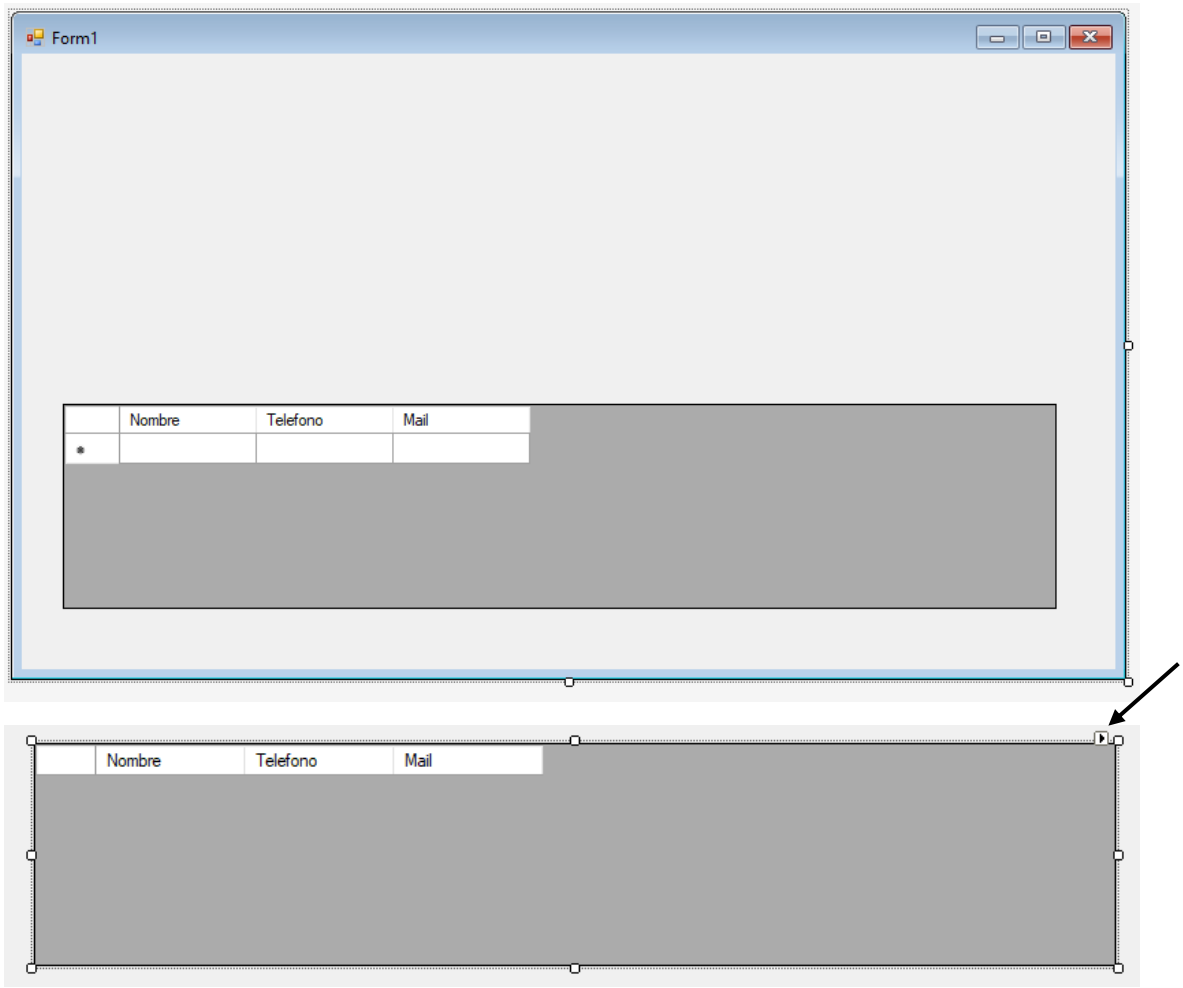


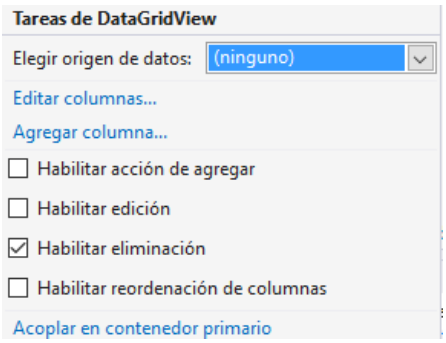


Agregar y Cerrar.

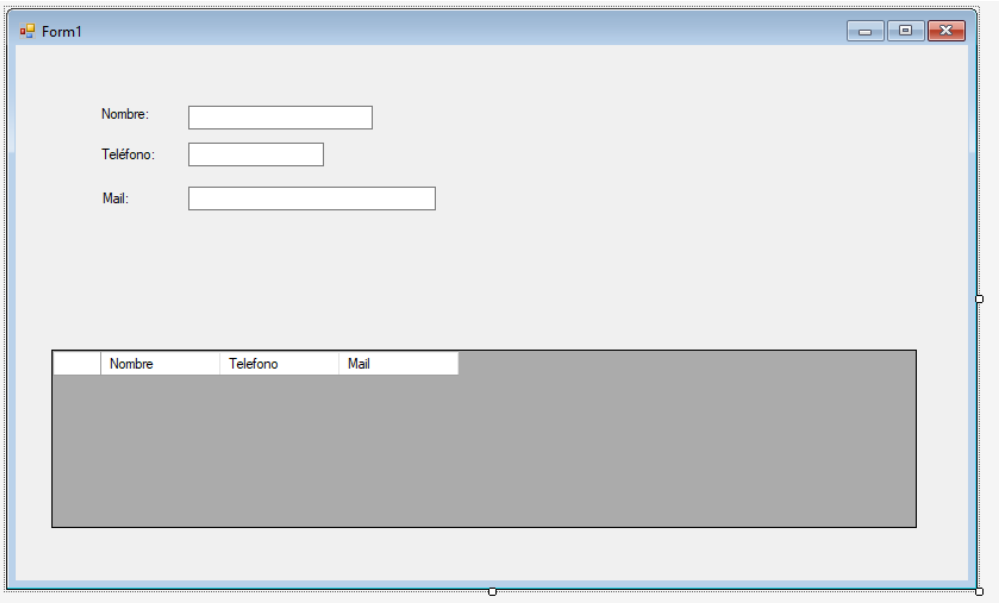


Le damos a Aceptar.

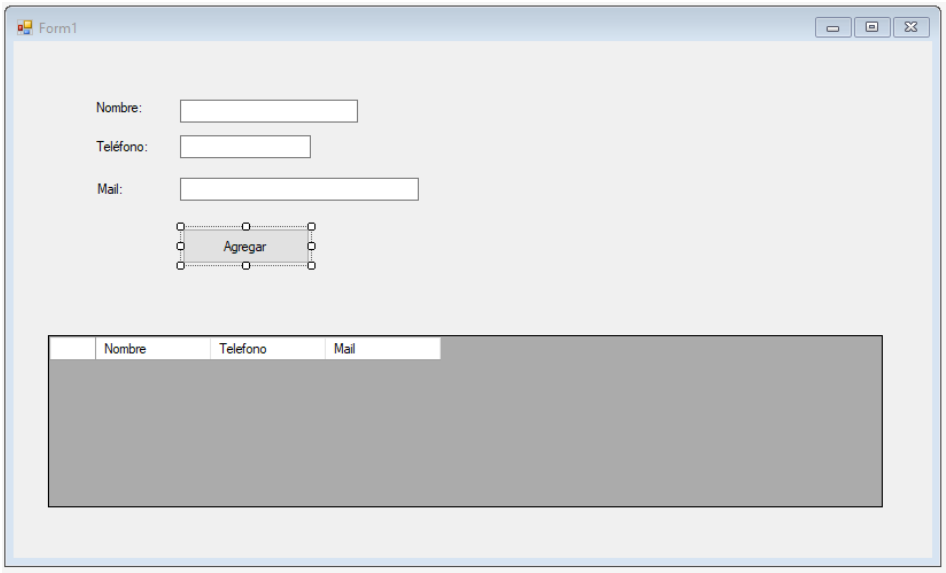




Deshabilitamos las opciones "Habilitar acción de agregar" y "Habilitar edición".
Cuando ejecutemos no podremos entrar datos
Vamos a seguir modificando el formulario.
Ahora vamos a añadir 3 label y 3 textbox.



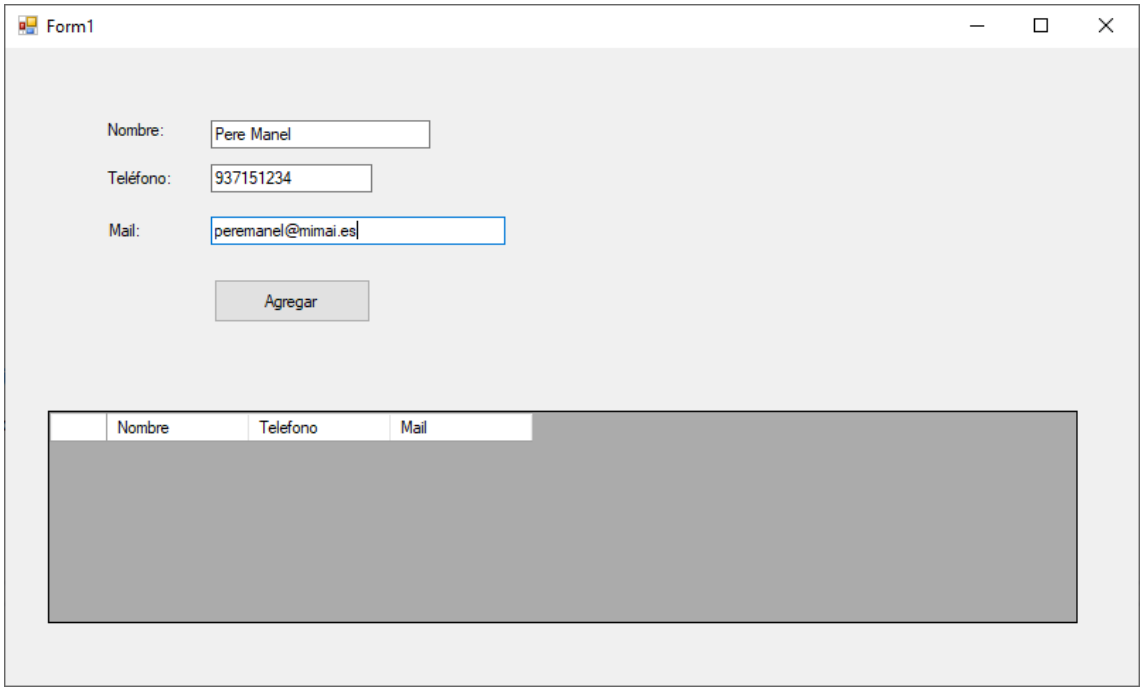
Ahora vamos a agregar un botón.



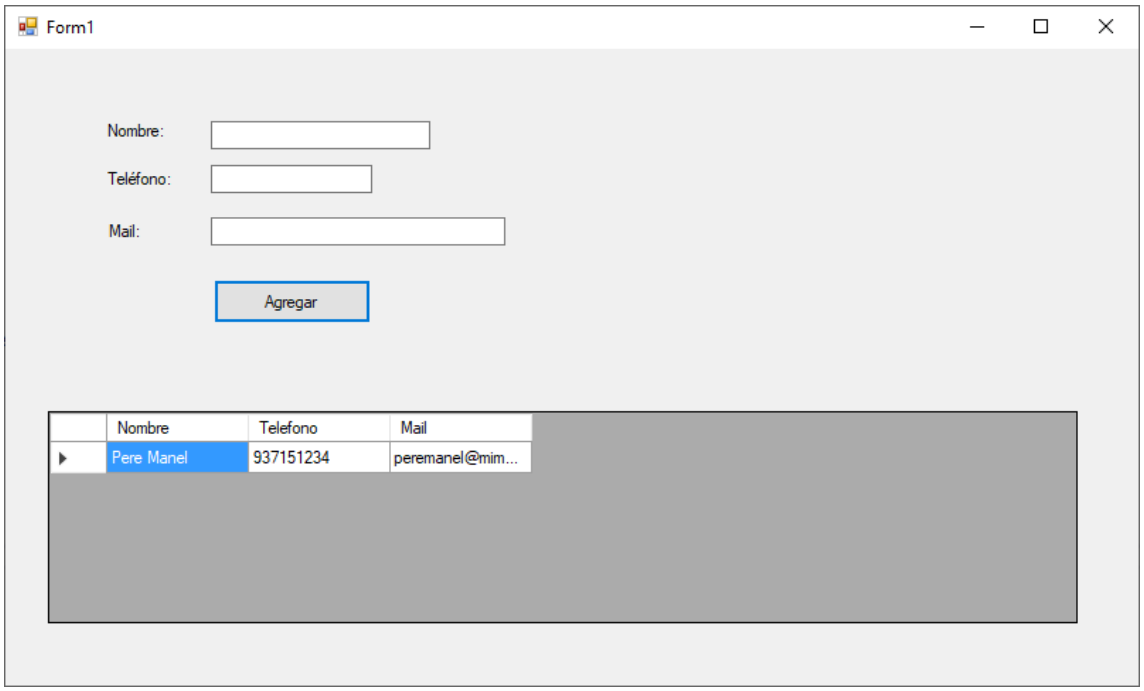
Vamos a codificar el botón Agregar en el evento click.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    dataGridView1.Rows.Add(textBox1.Text, textBox2.Text, textBox3.Text);
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";
}
```

Ejecutamos introducimos un nombre, teléfono y Mail



y hacemos click en el botón Agregar.



Cuando cerremos la aplicación estos datos se van a perder, vamos a ver como hacer que esto datos no se pierdan.
Vamos a programar en el evento Load del formulario.
Pero antes vamos a cargar la siguiente librería:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
```

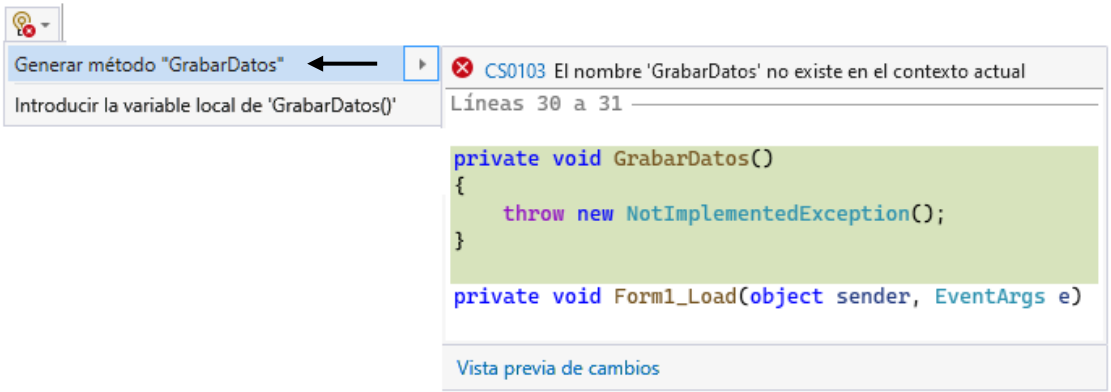
Este será el código del evento Load del formulario.

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    if (!File.Exists("agenda.txt"))
    {
        StreamWriter archivo = new StreamWriter("agenda.txt");
        archivo.Close();
    }
}
```

Cuando ejecutemos el programa si el archivo "agenda.txt" no existe lo creará.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    GrabarDatos();
    dataGridView1.Rows.Add(textBox1.Text, textBox2.Text, textBox3.Text);
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";
}
```

Modificamos el código del botón al hacer click con un método llamada GabarDatos()s, como este método no existe nos muestra un mensaje de error.
C# nos permite crear el método seleccionando el objeto que se encuentra a la izquierda del código que da error.



Seleccionaremos la opción Generar método "GrabarDatos".

```
1 referencia
private void GrabarDatos()
{
    throw new NotImplementedException();
}
```

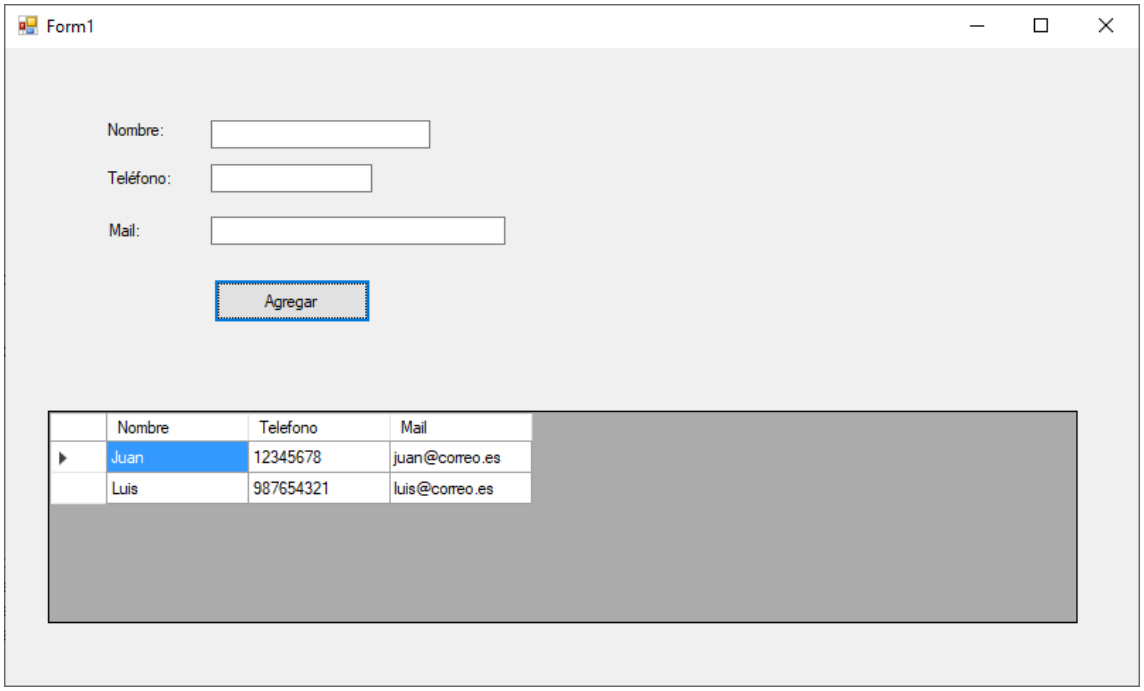
Ya lo podemos modificar.

Primero borraremos la línea que ha generado para escribir nuestro código.

```
private void GrabarDatos()
{
    StreamWriter archivo = new StreamWriter("agenda.txt", true);
    archivo.WriteLine(textBox1.Text);
    archivo.WriteLine(textBox2.Text);
    archivo.WriteLine(textBox3.Text);
    archivo.Close();
}
```

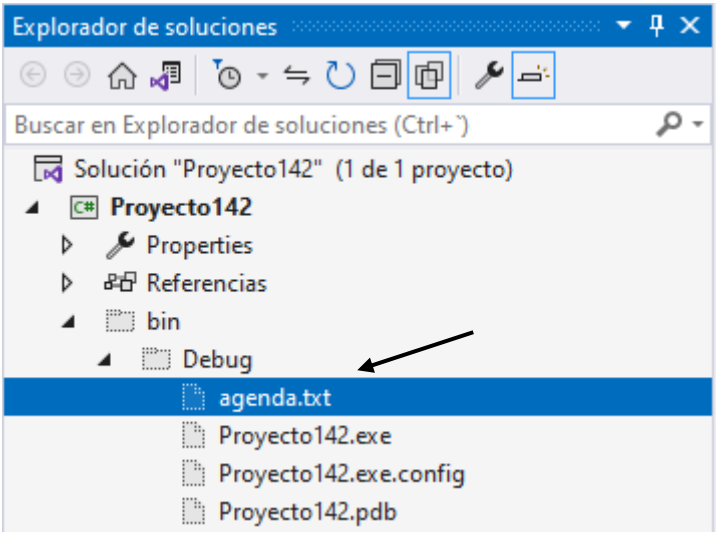
El argumento true lo utilizamos para decirlo que la información la tiene que agregar al final del contenido que tenga el archivo "agenda.txt".

Vamos a ejecutar, añadiremos los datos de dos personas y al final consultaremos el contenido del archivo "agenda.txt".

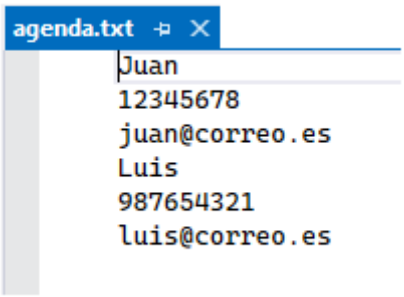


Cerramos la aplicación y vamos a ver donde a creado el archivo "agenda.txt".

En el Explorador de soluciones veremos que se ha creado el archivo "agenda.txt".



Vamos a hacer doble click sobre el para ver su contenido.

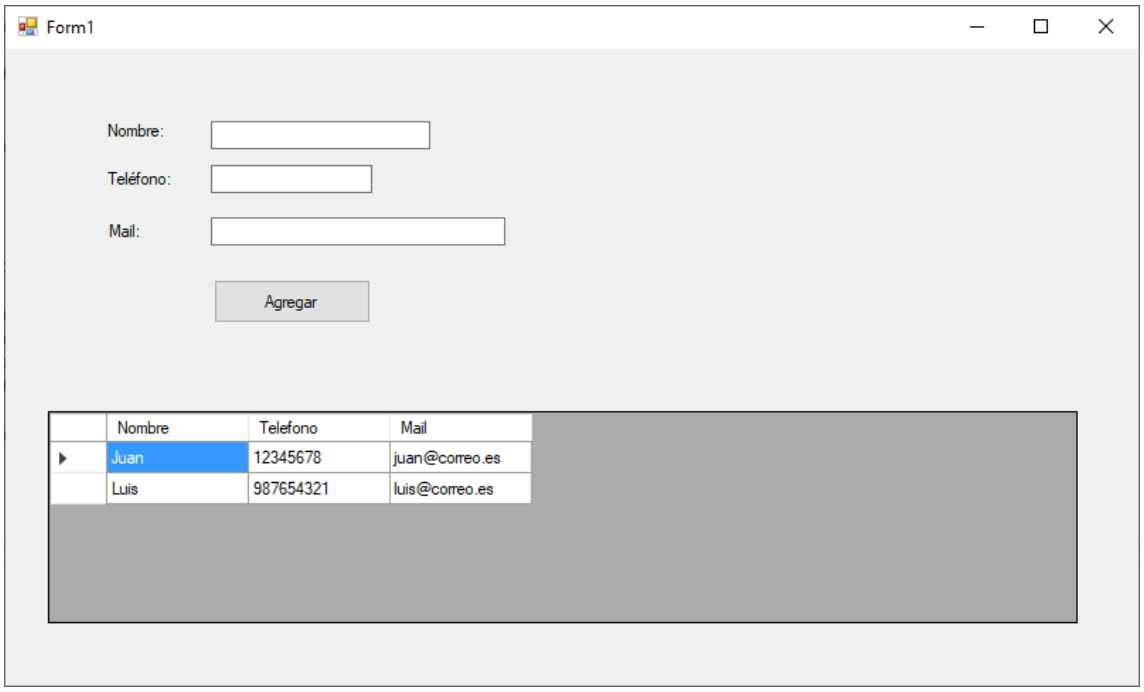


Ahora queremos que cada vez que ejecutemos el programa los cargue en control DataGridView.

Para esto vamos a modificar el código del evento Load del formulario.

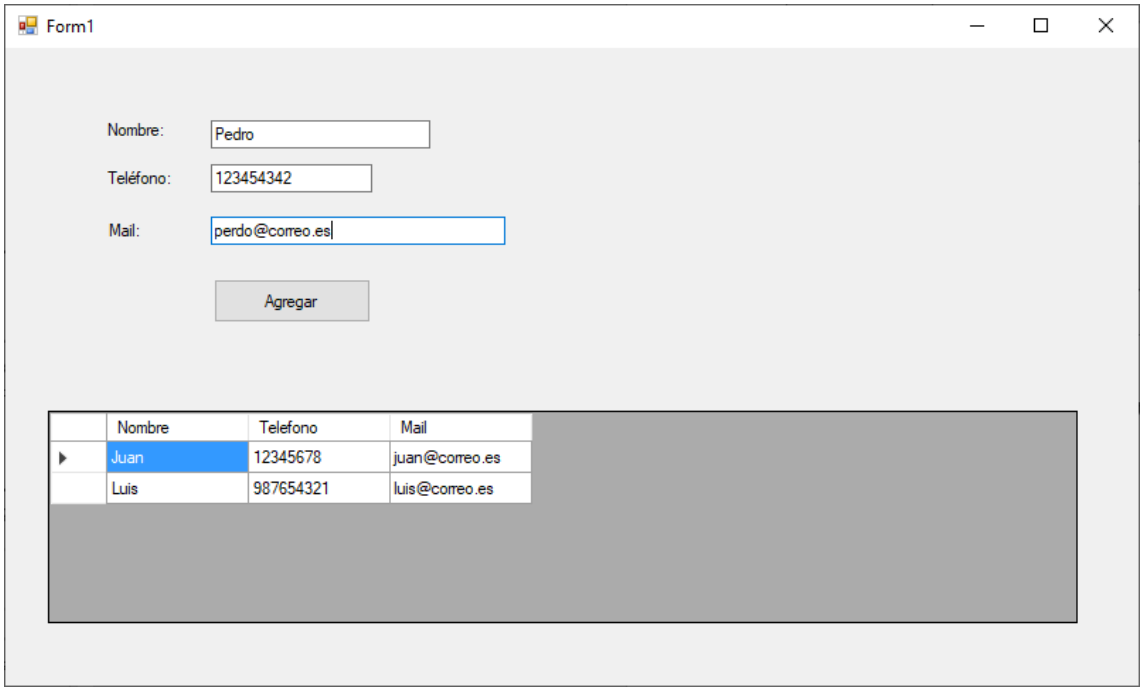
```
private void Form1_Load(object sender, EventArgs e)
{
    if (!File.Exists("agenda.txt"))
    {
        StreamWriter archivo = new StreamWriter("agenda.txt");
        archivo.Close();
    }
    else
    {
        StreamReader archivo = new StreamReader("agenda.txt");
        while (!archivo.EndOfStream)
        {
            string nombre = archivo.ReadLine();
            string telefono = archivo.ReadLine();
            string mail = archivo.ReadLine();
            dataGridView1.Rows.Add(nombre, telefono, mail);
        }
        archivo.Close();
    }
}
```

Ahora vamos a ejecutar el programa.



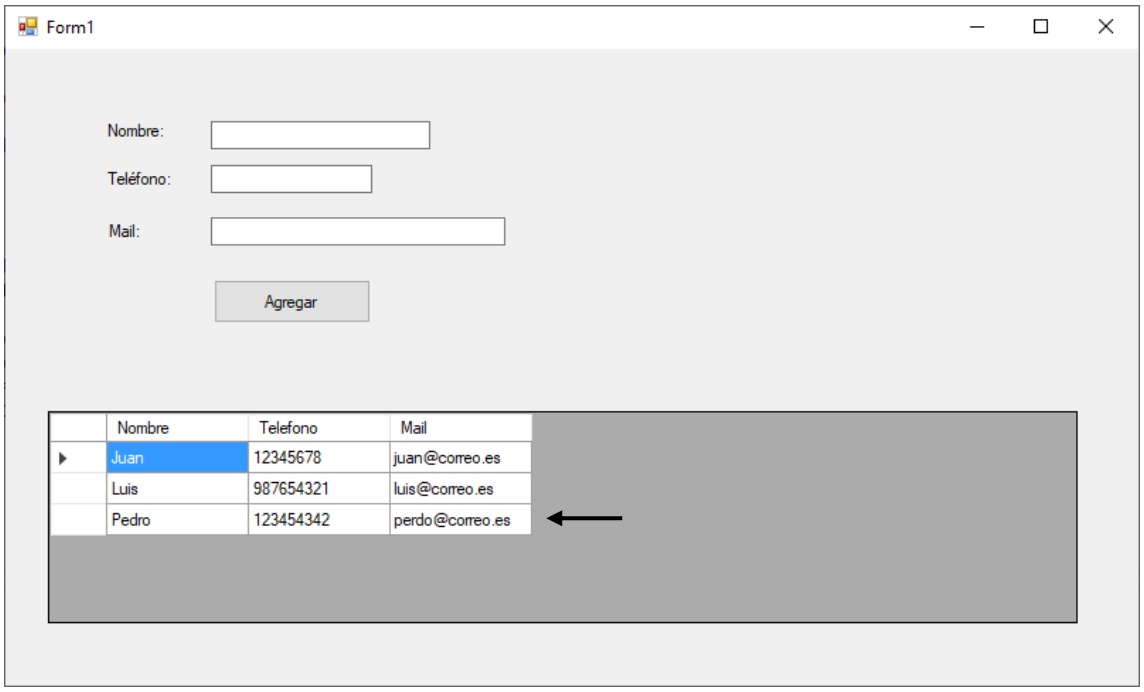
Ya se han cargado los datos.

Vamos a agregar un persona más en nuestra agenda.

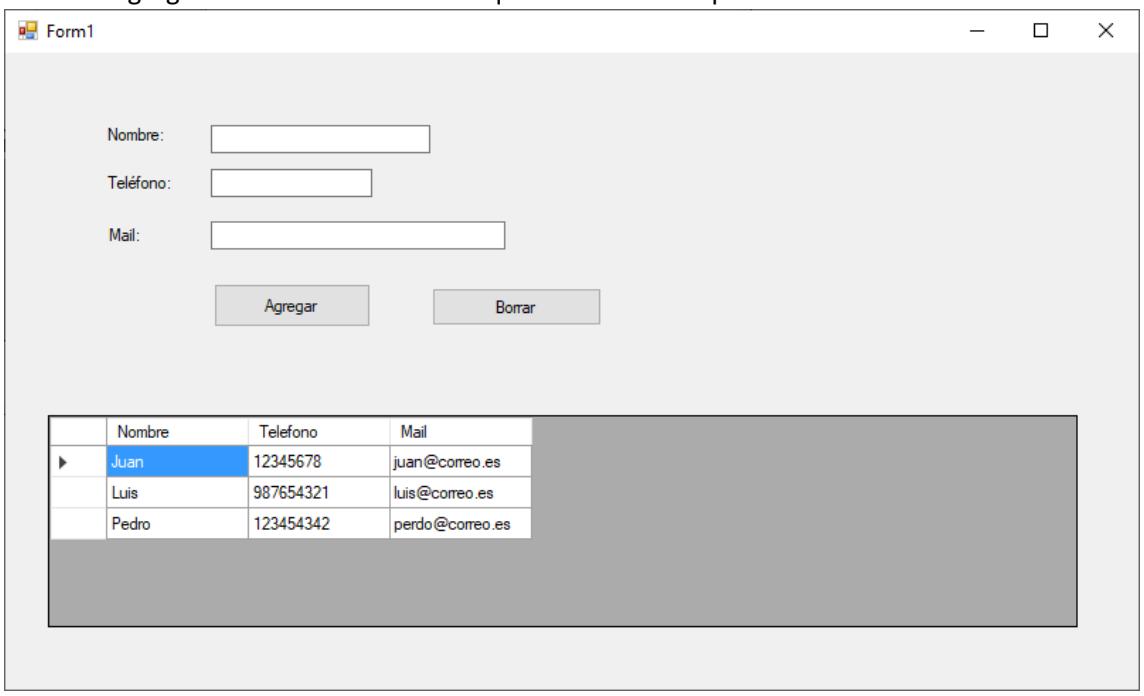


Le damos al botón agregar y a continuación cerramos el programa.

Ahora lo vamos a ejecutar de nuevo y veremos si se ha grabado la tercera persona.



Vamos a agregar una funcionalidad más que será un botón para borrar.



Vamos a programar en el evento click de este segundo botón que hemos agregado.

```
private void button2_Click(object sender, EventArgs e)
{
    for (int f=0; f<dataGridView1.Rows.Count; f++)
    {
        if (textBox1.Text == dataGridView1.Rows[f].Cells[0].Value.ToString())
        {
            dataGridView1.Rows.RemoveAt(f);
            MessageBox.Show("Se borró la persona");
        }
    }
}
```

Se borra del Control DataGridView, pero no se borra del archivo.

Si cerramos el programa y lo volvemos a ejecutar este volverá a salir.

Vamos a crear un método GrabarBorrado().

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    for (int f=0; f<dataGridView1.Rows.Count; f++)
    {
        if (textBox1.Text == dataGridView1.Rows[f].Cells[0].Value.ToString())
        {
            dataGridView1.Rows.RemoveAt(f);
            GrabarBorrado();
            MessageBox.Show("Se borró la persona");
        }
    }
}
```

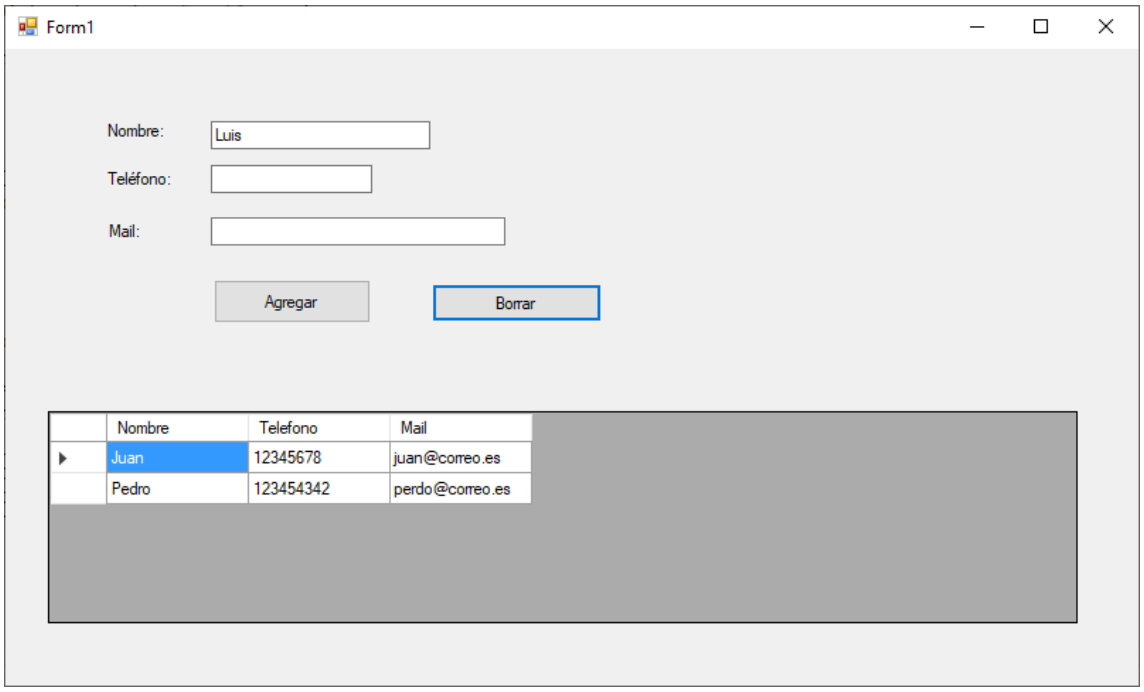
Nos colocaremos encima del método que no existe y sobre el icono que aparece le decimos que cree el método.

```
private void GrabarBorrado()
{
    throw new NotImplementedException();
}
```

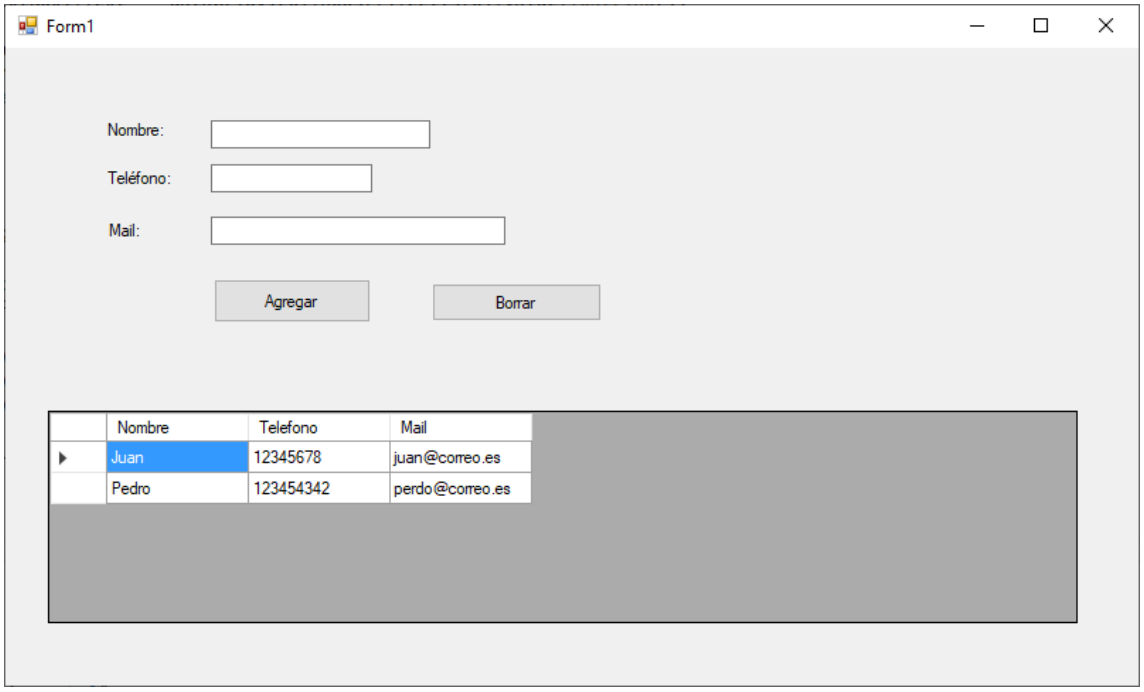
Borramos la línea que tiene este método para empezar a escribir nuestro código.

```
private void GrabarBorrado()
{
    StreamWriter archivo = new StreamWriter("agenda.txt");
    for (int f=0; f<dataGridView1.Rows.Count;f++)
    {
        archivo.WriteLine(dataGridView1.Rows[f].Cells[0].Value.ToString());
        archivo.WriteLine(dataGridView1.Rows[f].Cells[1].Value.ToString());
        archivo.WriteLine(dataGridView1.Rows[f].Cells[2].Value.ToString());
    }
    archivo.Close();
}
```

Ahora para ver si funciona vamos a borrar una persona, cerramos el programa lo abriremos de nuevo y comprobaremos si dicha persona ya no aparece.

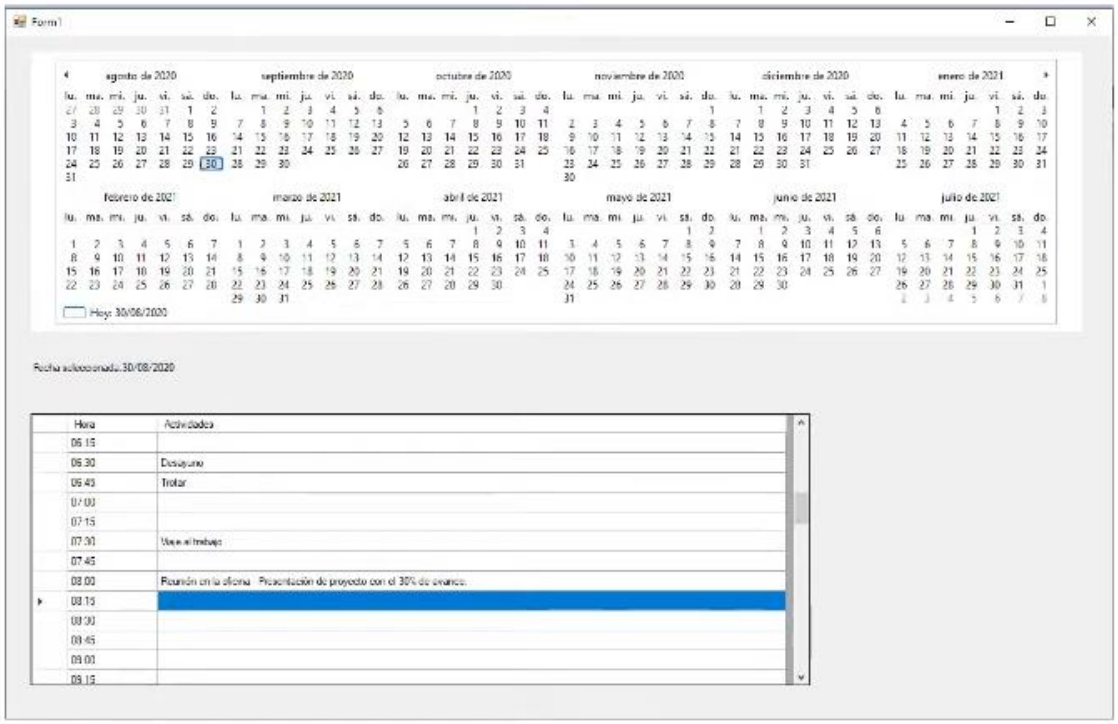


Ya he borrado una persona, voy a cerrar el programa y volverlo a ejecutar.



Ok, la persona ya no aparece.

Capítulo 132.- Control DataGridView – Calendario de actividades (Windows Forms)



Permitir seleccionar una fecha de un control MonthCalendar (Actualizar un Label).

Crear un archivo de texto cuando se selecciona una fecha que no tiene un archivo creado para dicha fecha con la hora 00:00 hasta las 23:45 (con avances de 15 minutos), además grabar una línea en blanco donde se almacenan las actividades.

Mostrar en un control DataGridView con dos columnas (Hora y Actividades) con el contenido del archivo de texto.

La columna de la Hora se debe fijar de solo lectura, en cambio la columna de Actividades el operador puede ingresar las actividades para dicha hora y fecha.

Cada vez que se termine de editar una celda del DataGridView proceder a actualizar el archivo de texto.

Vamos a la práctica:

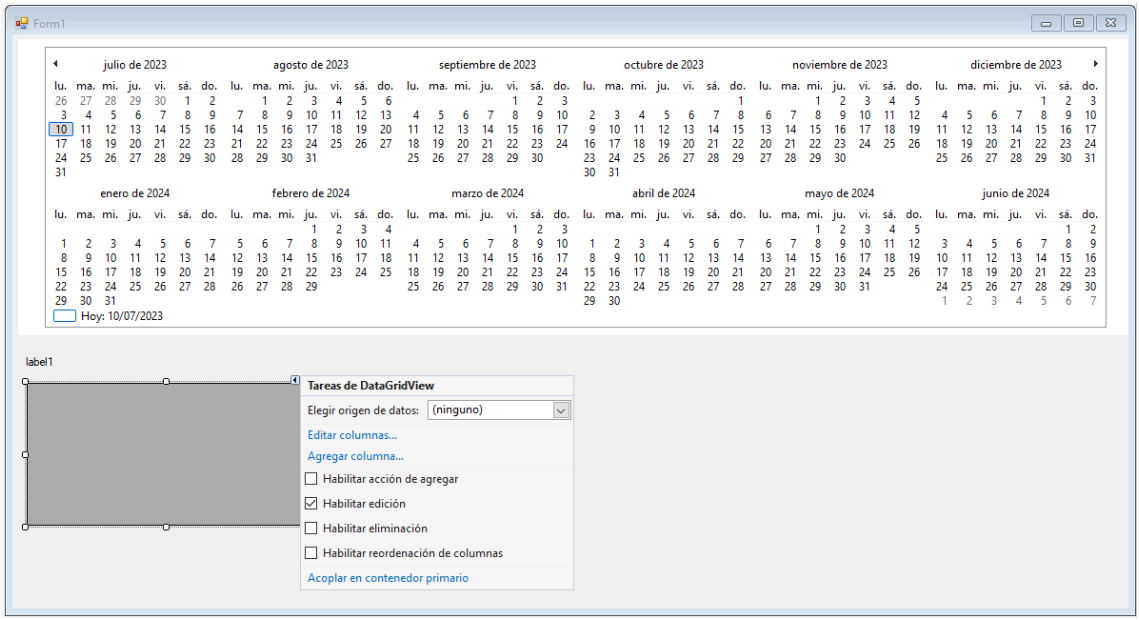
En controles comunes seleccionaremos MonthCalendar.

Lo vamos a redimensionar para que muestre 12 meses.

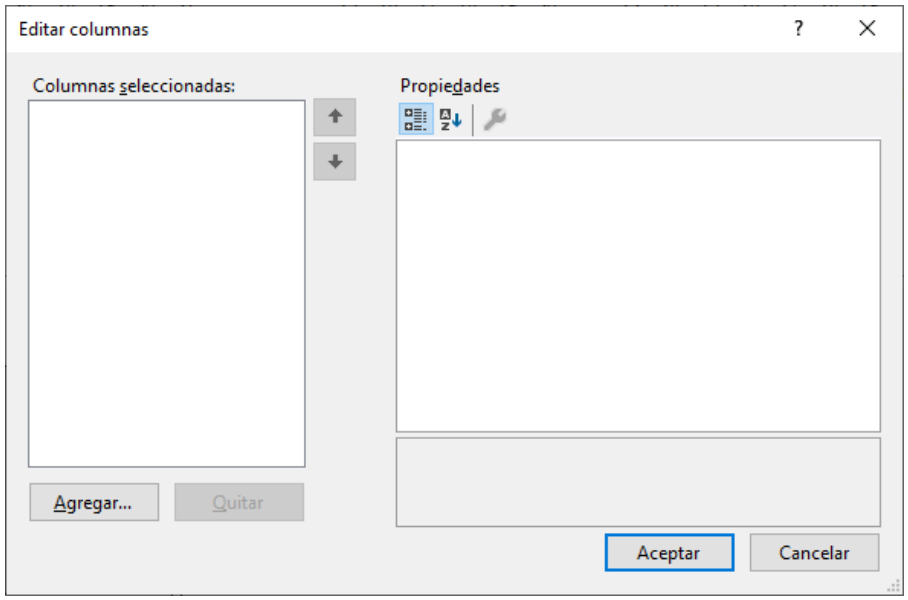
Agregamos un Label.

Del apartado Datos seleccionaremos un DataGridView.

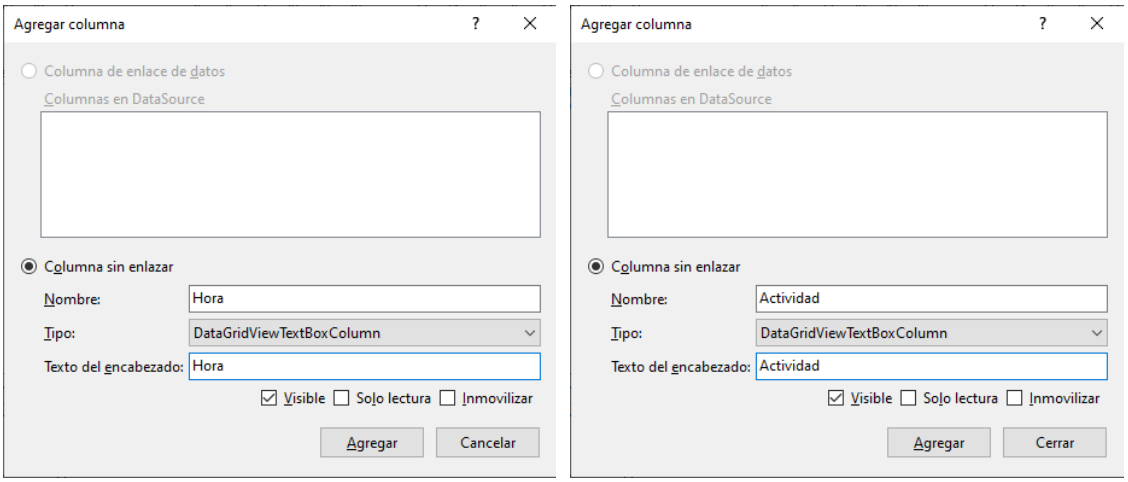
Solo dejaremos habilitada la edición.

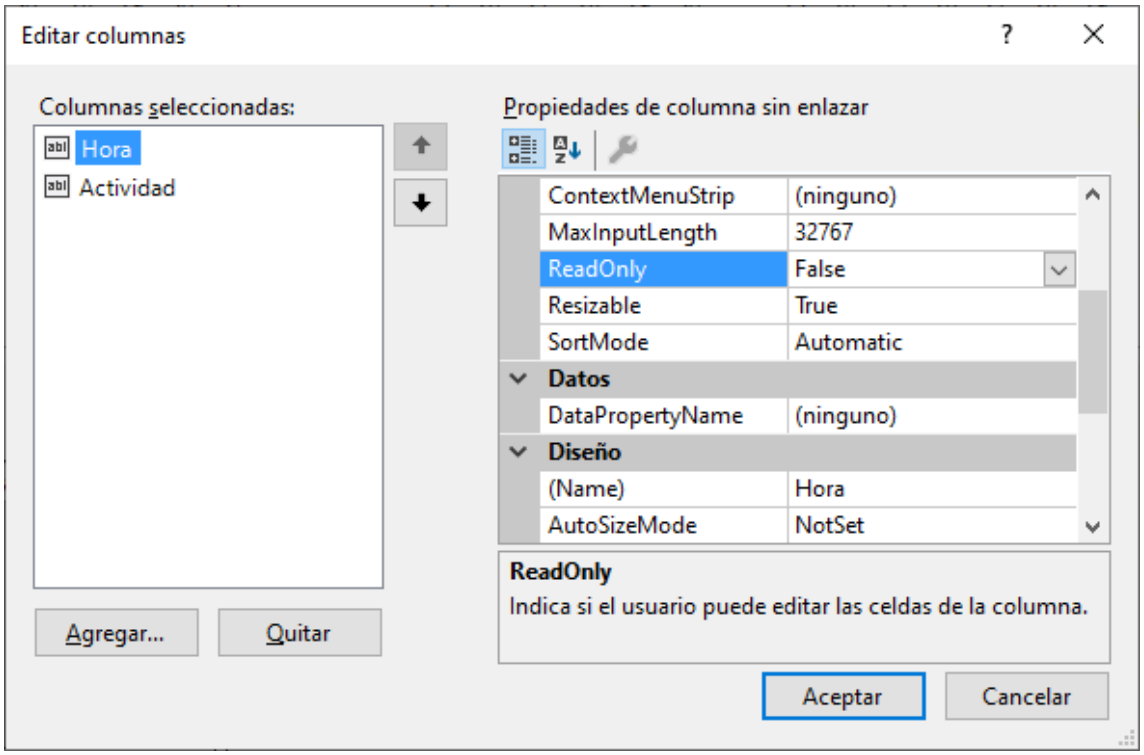


Vamos a definir las columnas seleccionaremos "Editar columnas".

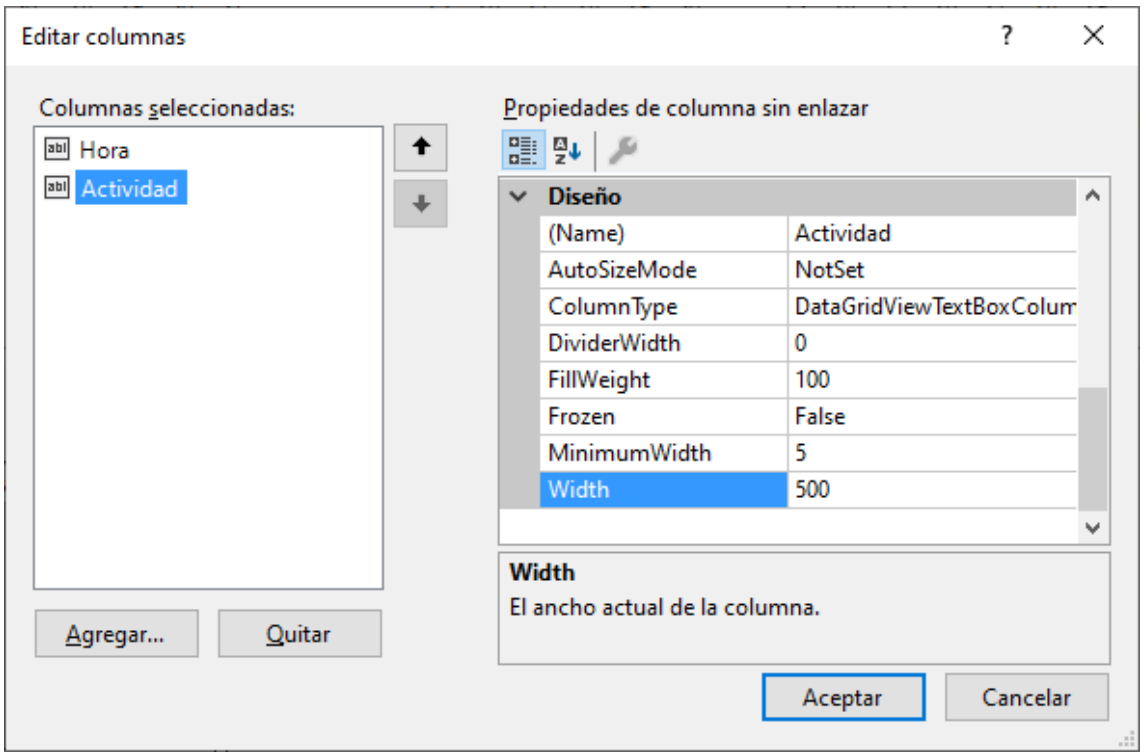


Vamos a agregar una que diga Hora y otra que diga Actividad.





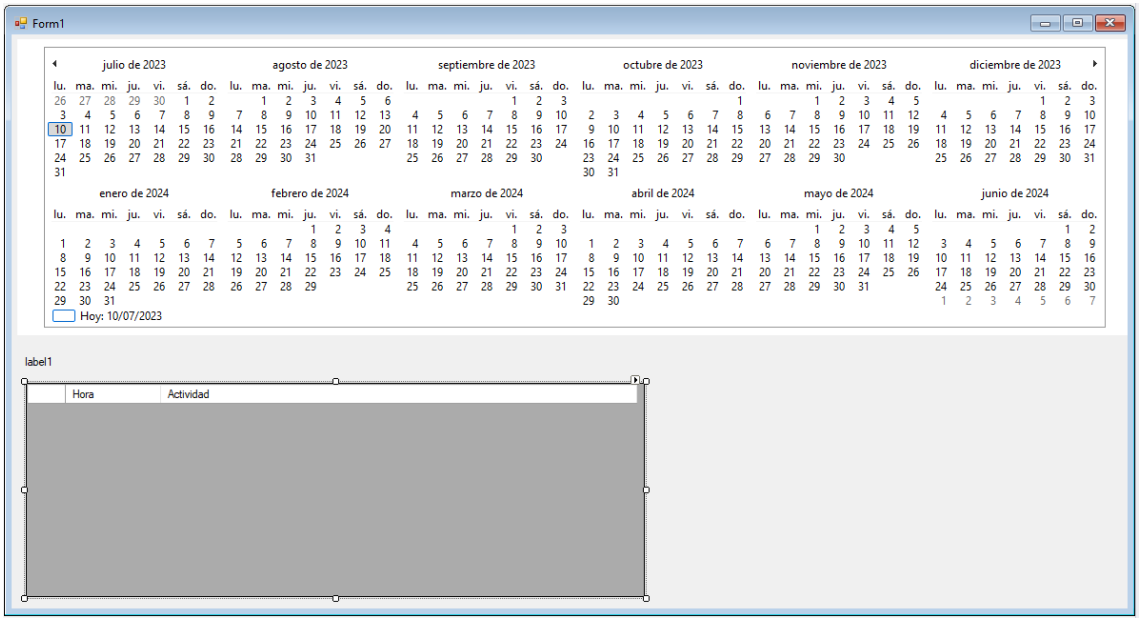
La columna Hora pondremos la propiedad ReadOnly en true.



La columna Actividad modificaremos su ancho a 500 px.

A continuación le damos a Aceptar.

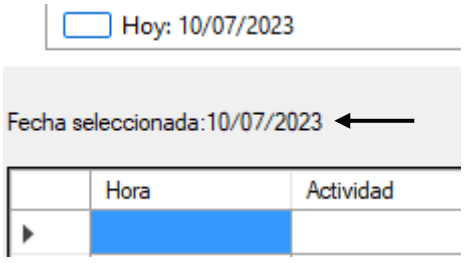
A continuación ajustamos el ancho del control DataGridView en el diseño.



El primer evento que vamos a programar será el evento Load del formulario, también podemos hacer doble click en el formulario donde no tengamos ningún objeto.

```
private void Form1_Load(object sender, EventArgs e)
{
    for (int f=1; f<=96; f++)
    {
        dataGridView1.Rows.Add();
    }
    CargarFecha();
}

1 referencia
private void CargarFecha()
{
    DateTime select = monthCalendar1.SelectionStart;
    label1.Text = "Fecha seleccionada:" + select.ToString("dd/MM/yyyy");
}
```

Si ejecutamos tiene que mostrar la fecha seleccionada:



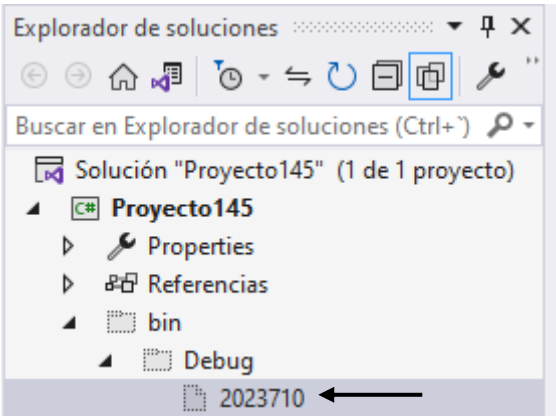
Ahora aunque cambiemos la fecha esta no se actualiza.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
```

Agregamos la librería using System.IO;

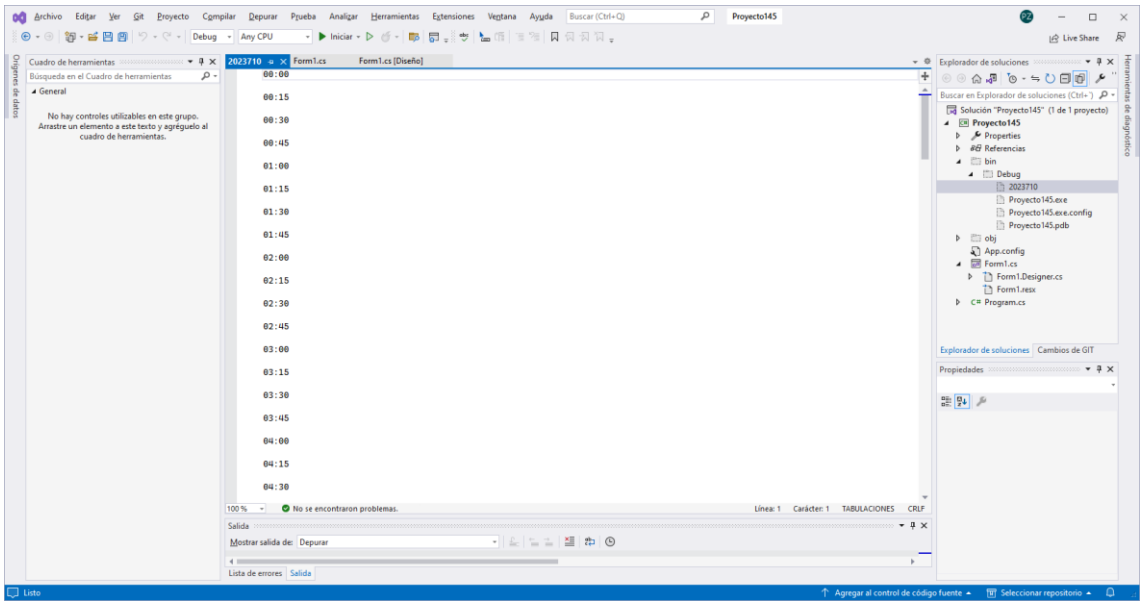
Vamos a modificar el método CargarFecha().

```
private void CargarFecha()
{
    DateTime select = monthCalendar1.SelectionStart;
    label1.Text = "Fecha seleccionada:" + select.ToString("dd/MM/yyyy");
    string fecha = select.Year.ToString()+select.Month.ToString()+select.Day.ToString();
    if (!File.Exists(fecha))
    {
        StreamWriter archivo = new StreamWriter(fecha);
        DateTime fe = DateTime.Today;
        for(int f=1; f<=96; f++)
        {
            archivo.WriteLine(fe.ToString("HH:mm"));
            archivo.WriteLine("");
            fe = fe.AddMinutes(15);
        }
        archivo.Close();
    }
    StreamReader archivo2 = new StreamReader(fecha);
    int x = 0;
    while (!archivo2.EndOfStream)
    {
        string linea1 = archivo2.ReadLine();
        string linea2 = archivo2.ReadLine();
        dataGridView1.Rows[x].Cells[0].Value = linea1;
        dataGridView1.Rows[x].Cells[1].Value = linea2;
        x++;
    }
     archivo2.Close();
}
```



Al ejecutar el programa ha creado el siguiente archivo.

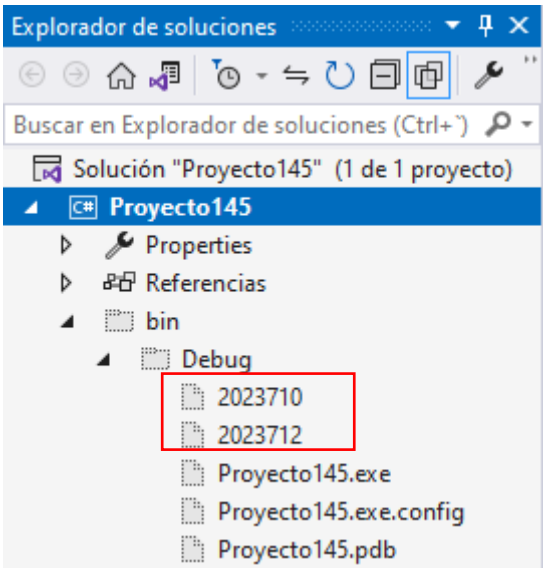
Este es su contenido:



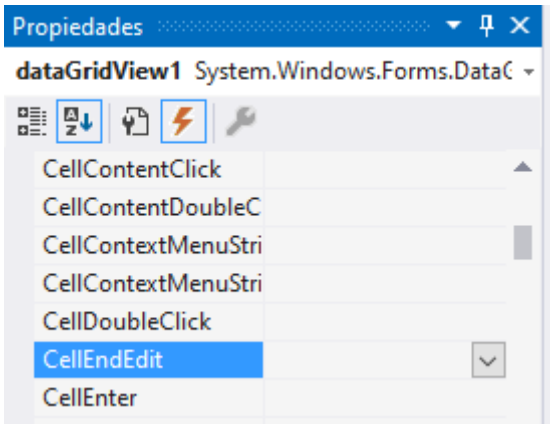
A continuación vamos a seleccionar el MothCalendar para programar el evento DateChagend.

```
private void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
{
    CargarFecha();
}
```

A partir de ahora ya podemos seleccionar otra fecha, esta se mostrara en el Label y creará otro archivo con su respectiva fecha.



A continuación vamos a seleccionar DataGridView y buscamos el evento CellEndEdit, este evento se dispara cuando termine de cargar una actividad.

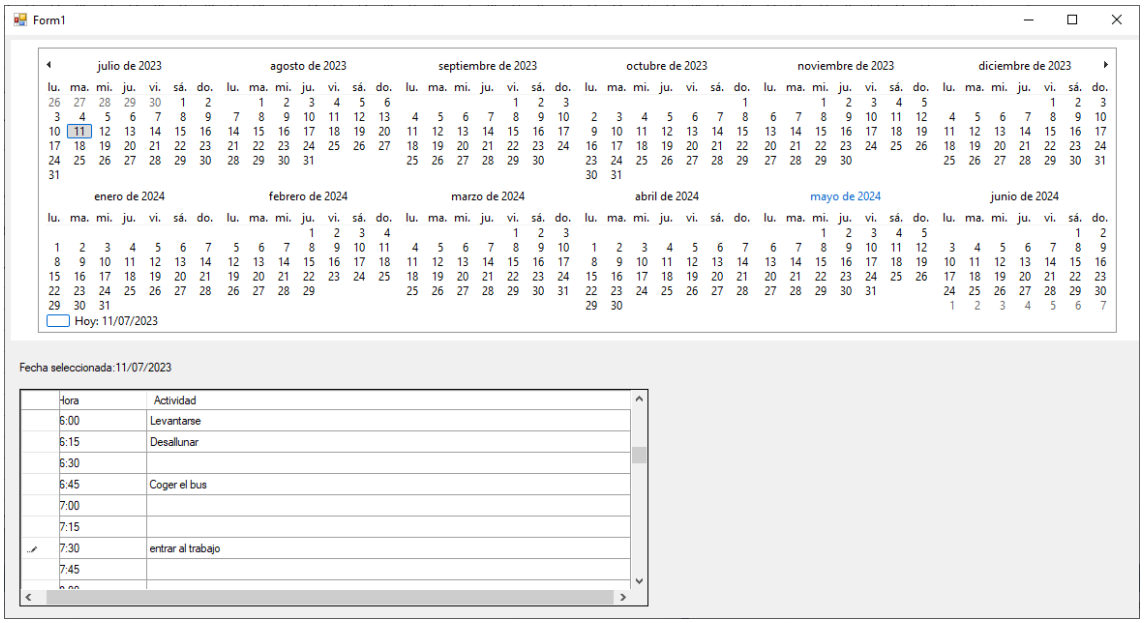


Vamos a realizar el correspondiente código:

```
private void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
{
    DateTime select = monthCalendar1.SelectionStart;
    string fecha = select.Year.ToString() + select.Month.ToString() + select.Day.ToString();
    StreamWriter archivo = new StreamWriter(fecha);
    for (int f = 1; f < dataGridView1.Rows.Count; f++)
    {
        archivo.WriteLine(dataGridView1.Rows[f].Cells[0].Value.ToString());
        if (dataGridView1.Rows[f].Cells[1].Value != null)
            archivo.WriteLine(dataGridView1.Rows[f].Cells[1].Value.ToString());
        else
            archivo.WriteLine("");
    }
    archivo.Close();
}
```

Vamos a ejecutar el programa.

Seleccionaremos una feche e introduciremos actividades, saldremos de ella y volveremos a entrar.



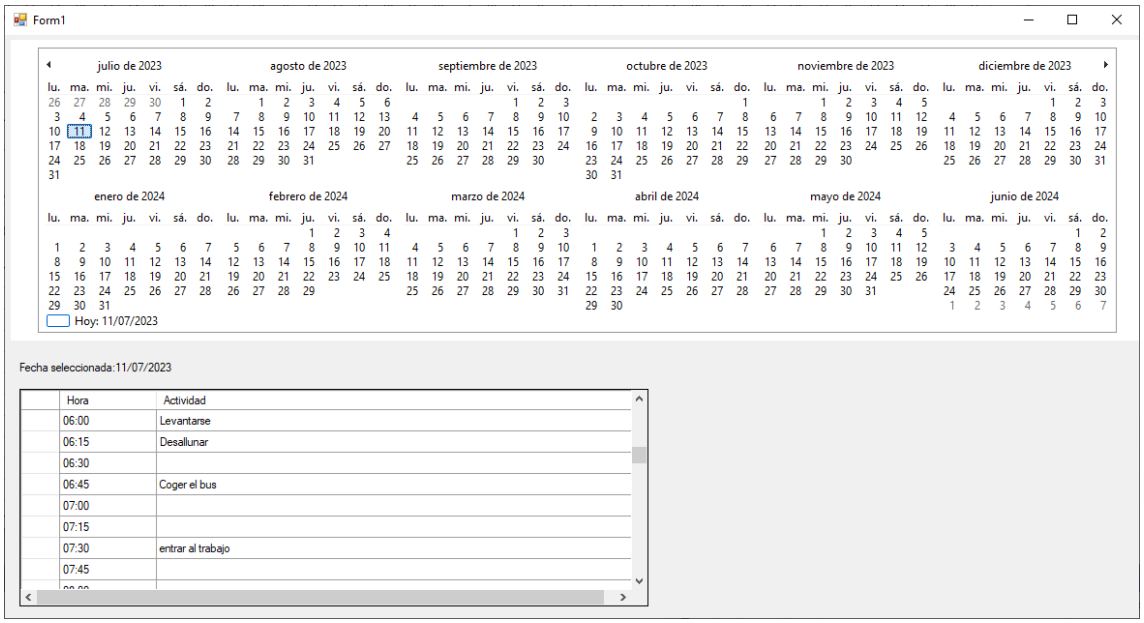
A continuación seleccionaremos la fecha siguiente y volvemos a seleccionar la fecha anterior.

Cerramos el programa.

Si abrimos el archivo de texto observaremos el siguiente contenido.

```
06:00
Levantarse
06:15
Desallunar
06:30

06:45
Coger el bus
07:00

07:15

07:30
entrar al trabajo
```

Ejecutamos de nuevo el programa.



Los datos están actualizados.

Resumen de todo el código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace Proyecto145
{
    public partial class Form1 : Form
    {
```

Para este proyecto hemos tenido que importar la siguiente librería.

```
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            for (int f=1; f<=96; f++)
            {
                dataGridView1.Rows.Add();
            }
            CargarFecha();
        }

        private void CargarFecha()
        {
            DateTime select = monthCalendar1.SelectionStart;
            label1.Text = "Fecha seleccionada:" +
select.ToString("dd/MM/yyyy");
            string fecha =
select.Year.ToString()+select.Month.ToString()+select.Day.ToString();
            if (!File.Exists(fecha))
            {
                StreamWriter archivo = new StreamWriter(fecha);
                DateTime fe = DateTime.Today;
                for(int f=1; f<=96; f++)
                {
                    archivo.WriteLine(fe.ToString("HH:mm"));
                    archivo.WriteLine("");
                    fe = fe.AddMinutes(15);
                }
                archivo.Close();
            }
            StreamReader archivo2 = new StreamReader(fecha);
            int x = 0;
            while (!archivo2.EndOfStream)
            {
                string linea1 = archivo2.ReadLine();
                string linea2 = archivo2.ReadLine();
                dataGridView1.Rows[x].Cells[0].Value = linea1;
                dataGridView1.Rows[x].Cells[1].Value = linea2;
                x++;
             }
             archivo2.Close();
        }

        private void monthCalendar1_DateChanged(object sender,
DateRangeEventArgs e)
        {
            CargarFecha();
        }

        private void dataGridView1_CellEndEdit(object sender,
DataGridViewCellEventArgs e)
        {
            DateTime select = monthCalendar1.SelectionStart;
            string fecha = select.Year.ToString() + select.Month.ToString()
+ select.Day.ToString();
            StreamWriter archivo = new StreamWriter(fecha);
            for (int f = 1; f < dataGridView1.Rows.Count; f++)
            {

archivo.WriteLine(dataGridView1.Rows[f].Cells[0].Value.ToString());
```

Cuando ejecutamos el programa queremos que añada 96 columnas al control dataGridView1.

A continuación ejecuta el método CargarFecha().

```
                if (dataGridView1.Rows[f].Cells[1].Value != null)

archivo.WriteLine(dataGridView1.Rows[f].Cells[1].Value.ToString());
                else
                    archivo.WriteLine("");
            }
            archivo.Close();
        }
    }
}
```

```
        private void CargarFecha()
        {
            DateTime select = monthCalendar1.SelectionStart;
            label1.Text = "Fecha seleccionada:" + select.ToString("dd/MM/yyyy");
```

La variable select de tipo DateTime le asignamos la fecha que está seleccionada en el control monthCalendar1.
A la etiqueta label1 le asignamos el texto "Fecha seleccionada:" más el valor de la variable select reconvertido a string con el formato "dd/MM/yyyy".

```
            string fecha =
select.Year.ToString()+select.Month.ToString()+select.Day.ToString();
```

La variable fecha de tipo string le asingamos el valor del año, mes y día por ejemplo: 20230711.

```
            if (!File.Exists(fecha))
```

Si el archive asignado al la variable fecha no existe.

```
            {
                StreamWriter archivo = new StreamWriter(fecha);
                DateTime fe = DateTime.Today;
                for(int f=1; f<=96; f++)
                {
                    archivo.WriteLine(fe.ToString("HH:mm"));
                    archivo.WriteLine("");
                    fe = fe.AddMinutes(15);
                }
                archivo.Close();
            }
```

Creamos un objeto archivo de la clase StreamWrite para escribir en un archivo con el nombre que tenga asignda la variable fecha.
La variable fe de tipo DateTime le asignamos la fecha de hoy.
En nuestra agenda queremos hacer bloques de 15 minutos para anotar nuestras actividades y multiplicamos 25 por 4 de los cuatro bloques de 15 minutos que tiene la hora nos dará un resultado de 96.
Hacemos un bucle for para que este se repita 96 veces, en el archivo escribimos en una línea la hora y minuto y en la segunda línea un espacio en blanco.
A la variable fe le incrementamos a su valor 15 minutos.
Finalizado el bucle for, cerramos el archivo.

```
            StreamReader archivo2 = new StreamReader(fecha);
            int x = 0;
            while (!archivo2.EndOfStream)
            {
                string linea1 = archivo2.ReadLine();
```

```
                string linea2 = archivo2.ReadLine();
                dataGridView1.Rows[x].Cells[0].Value = linea1;
                dataGridView1.Rows[x].Cells[1].Value = linea2;
                x++;
            }
            archivo2.Close();
        }
```

A continuación fuera de la condición creamos un objeto archivo2 de la clase StreamReader para poder leer el archivo que contiene la variable fecha.
Inicializamos y asignamos a la variable x el valor 0.
Con el While controlamos el bucle que se irá repitiendo mientras no lleguemos al final del archivo.
La linea1 lee la primera línea del archivo.
La línea2 lee la siguiente línea del archivo.
Lo asignamos al control dataGridView1 en la columna y fila correspondiente según el valor de x.
Terminado el bucle cerramos el archivo.

```
private void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
```

Cuando cambiamos de fecha se genera el evento DateChanged.

```
        {
            CargarFecha();
        }
```

Llamamos al método CargarFecha().

```
        private void dataGridView1_CellEndEdit(object sender, DataGridViewCellEventArgs e)
```

Cuando terminamos de editar en el control dataGridView1.

```
        {
            DateTime select = monthCalendar1.SelectionStart;
            string fecha = select.Year.ToString() + select.Month.ToString() +
select.Day.ToString();
            StreamWriter archivo = new StreamWriter(fecha);
```

La variable select de tipo DateTime le asignamos el valor que tiene seleccionado el control monthCalendar1.
La variable fecha de tipo string le asignamos el valor del año, mes y día.
Creamos un objeto llamado archivo de la clase StreamWrite para poder escribir en el archivo que tiene asignada la variable fecha.

```
            for (int f = 1; f < dataGridView1.Rows.Count; f++)
            {
                archivo.WriteLine(dataGridView1.Rows[f].Cells[0].Value.ToString());
                if (dataGridView1.Rows[f].Cells[1].Value != null)
                    archivo.WriteLine(dataGridView1.Rows[f].Cells[1].Value.ToString());
                else
                    archivo.WriteLine("");
            }
            archivo.Close();
        }
    }
}
```

En un for recorremos todas las columnas que tiene dataGridView1.
En el archivo escribimos lo que tiene dataGridView1 en la fila [f] y columna [0].

Comparamos si en dataGridView1 fila [f] y columna [1] es distinto a vacío, que escriba en el archivo el contenido de dataGridView1 fila [f] y columna [1].
Si no que escriba una línea en blanco.
Finalizado el bucle cerramos el archivo.

Capítulo 133.- clase Graphics – métodos más comunes.

GDI+ es un API para manipulación de gráficos en Microsoft.Net.

Al tratarse de un interfaz de programación independiente del dispositivo físico sobre el que se van a generar los gráficos, el programados gana en flexibilidad, ya que no debe preocuparse de si el gráfico generado se va a mostrar por el monitor, generación de un archivo de imagen, impresora, etc.; esta labor es resuelta por GDI+, que aísla el programa del hardware a manejar.

GDI+ divide su campo de trabajo en tres áreas principales:

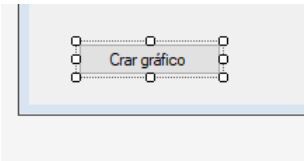
- Generación de gráficos vectoriales 2D.
- Manipulación de imágenes en los formatos gráficos más habituales.
- Visualización de texto.

Para utilizar las clases relacionadas con la manipulación de gráficos, es preciso importar este espacio de nombres System.Drawing.

System.Drawing contiene el conjunto de clases principales, aunque no es el único nemespace de GDI+; para tareas de mayor especialización con gráficos deberemos recurrir a alguno de los siguientes:

- System.Drawing.Drawing2D
- System.Drawing.Imaging
- System.Drawing.Text

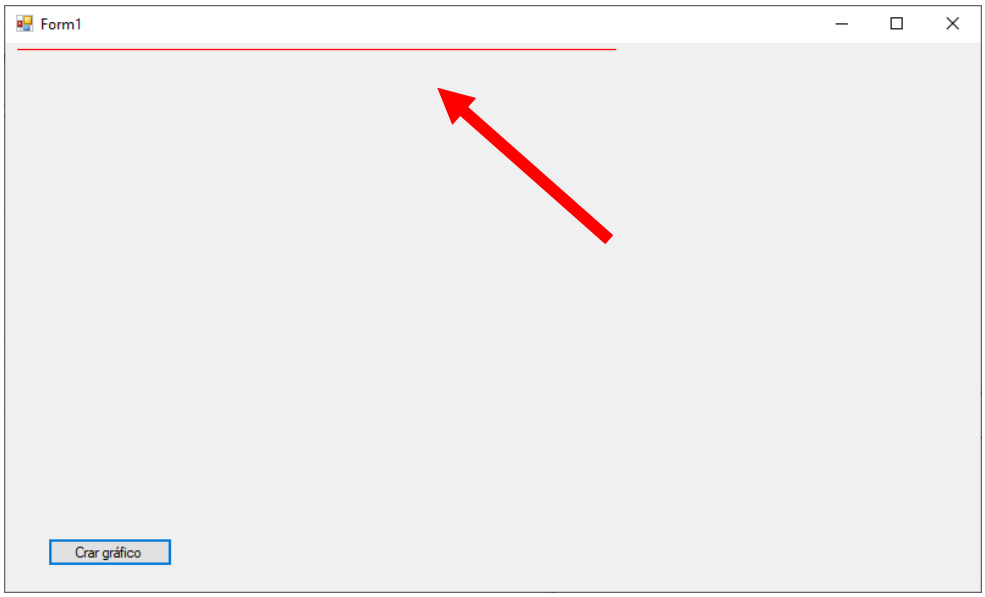
Vamos a realizar un nuevo proyecto.



En la parte inferior izquierda del formulario agregaremos un botón.

Vamos a programar el evento click del botón.

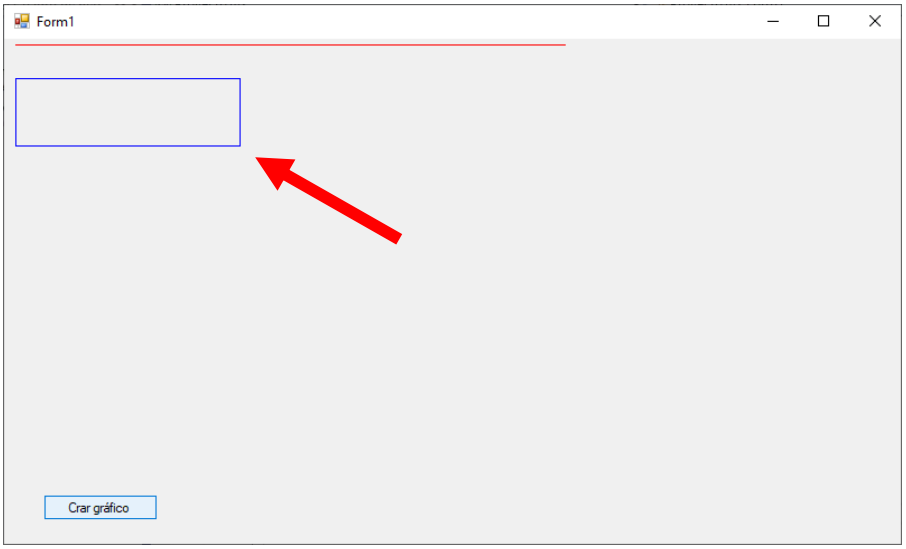
```
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red), 10, 5, 500, 5);

}
```



Hemos dibujado una línea.

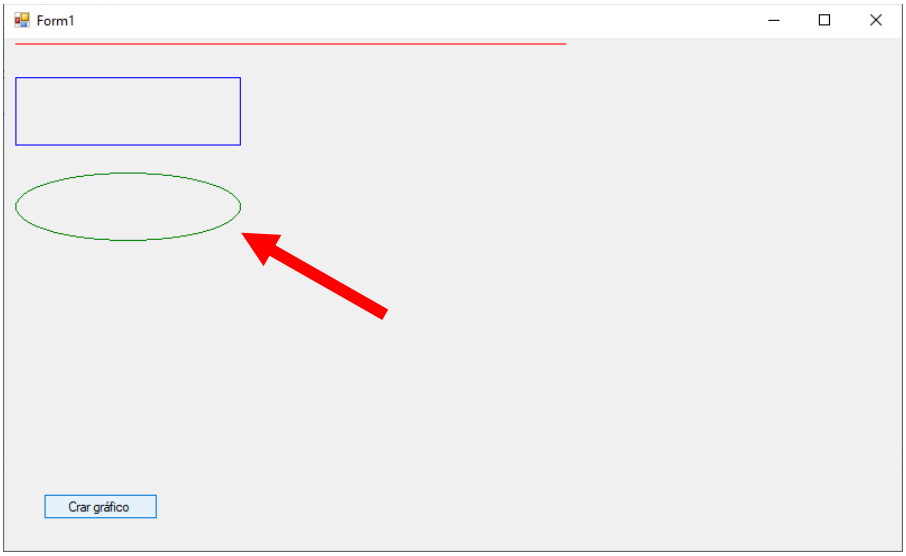
Vamos a agregar un rectángulo.

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red), 10, 5, 500, 5);
    lienzo.DrawRectangle(new Pen(Color.Blue), 10, 35, 200, 60);
}
```



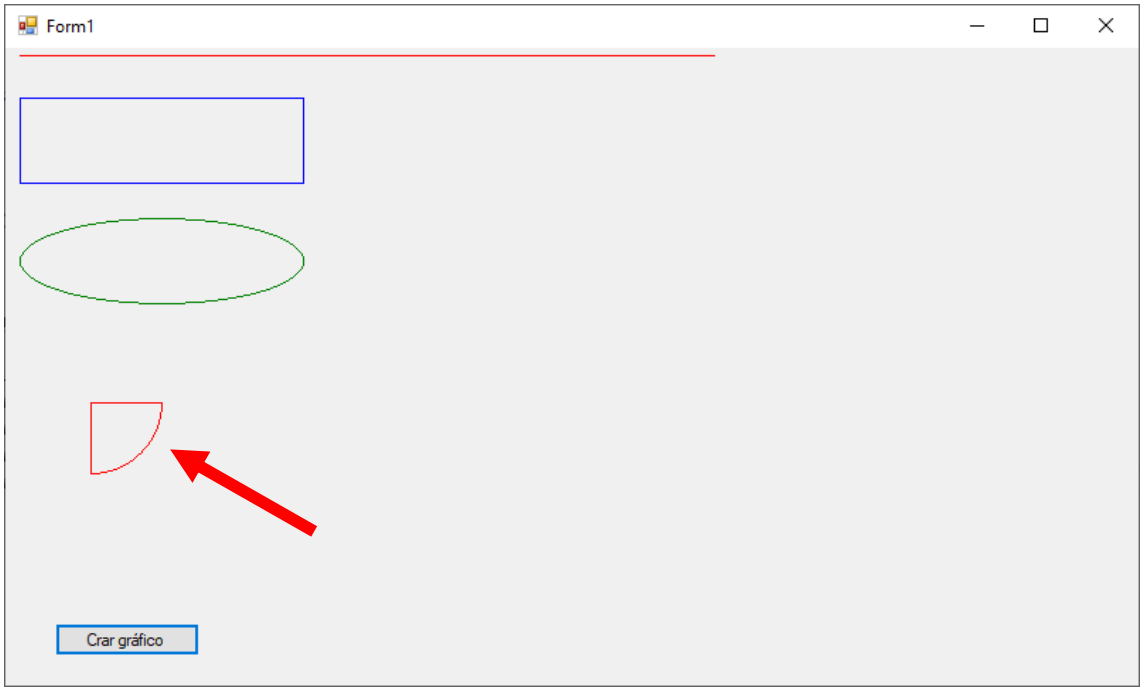
Vamos a dibujar un elipse.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red), 10, 5, 500, 5);
    lienzo.DrawRectangle(new Pen(Color.Blue), 10, 35, 200, 60);
    lienzo.DrawEllipse(new Pen(Color.Green), 10, 120, 200, 60);
}
```



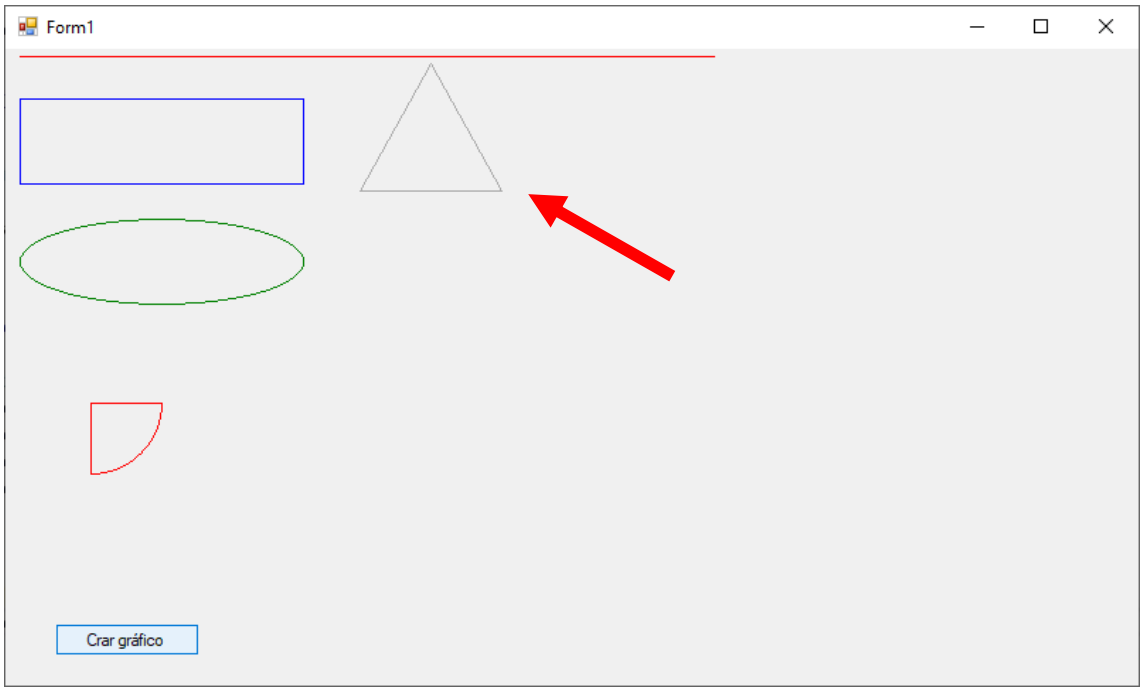
Vamos a agregar un trozo de tarta.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red), 10, 5, 500, 5);
    lienzo.DrawRectangle(new Pen(Color.Blue), 10, 35, 200, 60);
    lienzo.DrawEllipse(new Pen(Color.Green), 10, 120, 200, 60);
    lienzo.DrawPie(new Pen(Color.Red), 10, 200, 100, 100, 0, 90);
}
```



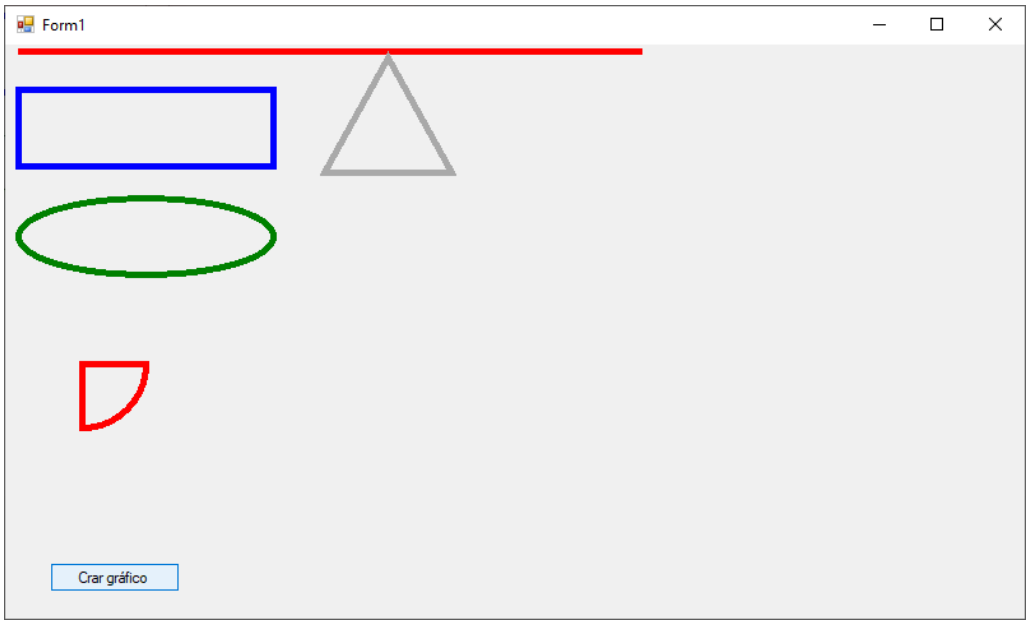
Queremos dibujar un triángulo.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red), 10, 5, 500, 5);
    lienzo.DrawRectangle(new Pen(Color.Blue), 10, 35, 200, 60);
    lienzo.DrawEllipse(new Pen(Color.Green), 10, 120, 200, 60);
    lienzo.DrawPie(new Pen(Color.Red), 10, 200, 100, 100, 0, 90);
    Point punto1 = new Point(300, 10);
    Point punto2 = new Point(350, 100);
    Point punto3 = new Point(250, 100);
    Point[] puntos = {punto1, punto2, punto3};
    lienzo.DrawPolygon(new Pen(Color.DarkGray), puntos);
}
```



Vamos a modificar los grosores de las líneas.

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red, 5), 10, 5, 500, 5);
    lienzo.DrawRectangle(new Pen(Color.Blue, 5), 10, 35, 200, 60);
    lienzo.DrawEllipse(new Pen(Color.Green, 5), 10, 120, 200, 60);
    lienzo.DrawPie(new Pen(Color.Red, 5), 10, 200, 100, 100, 0, 90);
    Point punto1 = new Point(300, 10);
    Point punto2 = new Point(350, 100);
    Point punto3 = new Point(250, 100);
    Point[] puntos = {punto1, punto2, punto3};
    lienzo.DrawPolygon(new Pen(Color.DarkGray, 5), puntos);
}
```

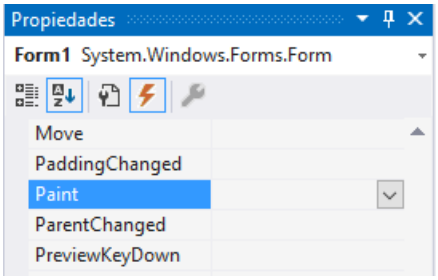


Si modificamos el tamaño de la ventana para reducir su ancho y la volvemos a dejar como estaba puede pasar esto:

Esto para algunos casos puede ser un inconveniente.

Si minimizamos y volvemos a restaurar también desaparecen las imágenes.

Para evitar que no suceda el formulario tiene un método Paint.

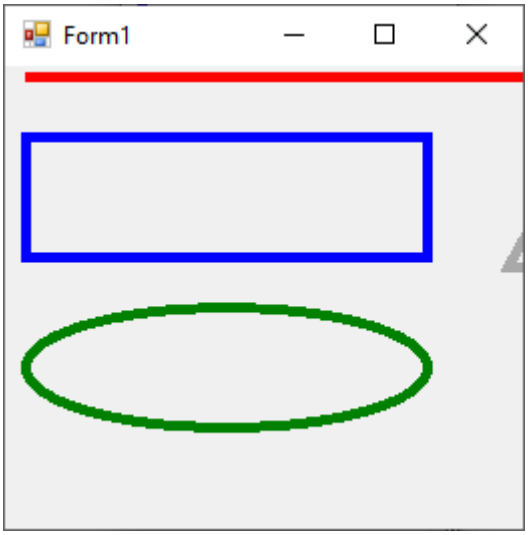


Vamos a copiar todo el código en el método Paint.

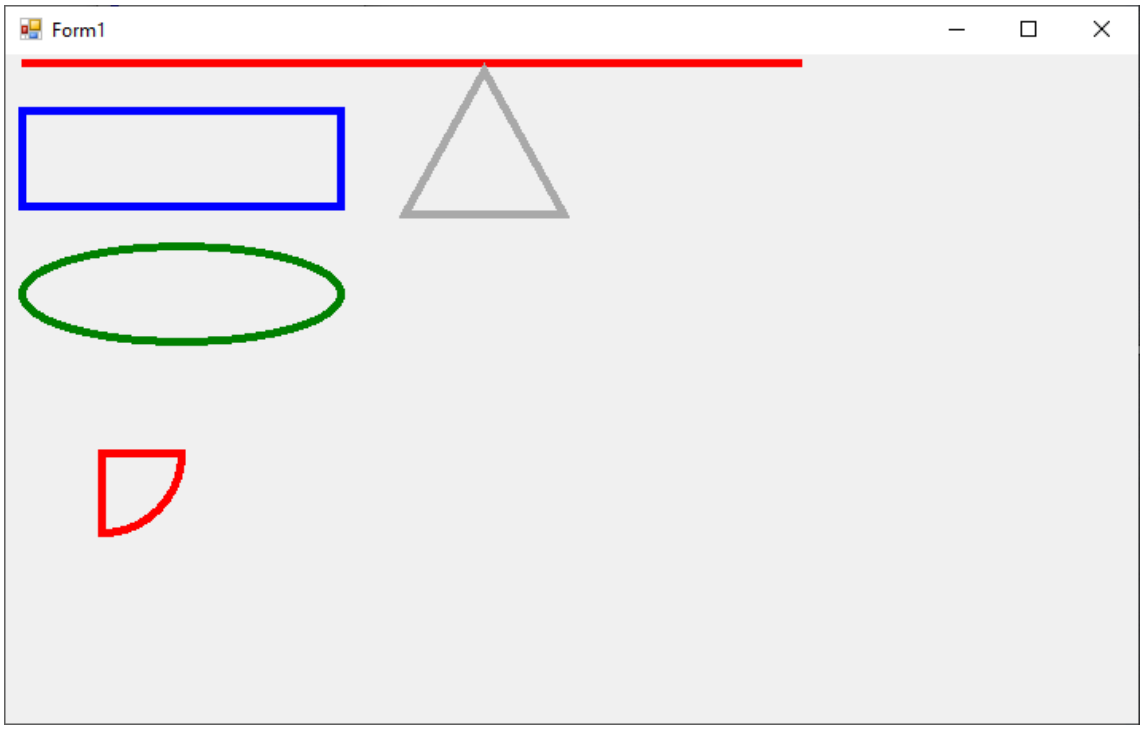
```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red, 5), 10, 5, 500, 5);
    lienzo.DrawRectangle(new Pen(Color.Blue, 5), 10, 35, 200, 60);
    lienzo.DrawEllipse(new Pen(Color.Green, 5), 10, 120, 200, 60);
    lienzo.DrawPie(new Pen(Color.Red, 5), 10, 200, 100, 100, 0, 90);
    Point punto1 = new Point(300, 10);
    Point punto2 = new Point(350, 100);
    Point punto3 = new Point(250, 100);
    Point[] puntos = { punto1, punto2, punto3 };
    lienzo.DrawPolygon(new Pen(Color.DarkGray, 5), puntos);
}
```

A continuación eliminamos el botón del formulario.

Ya podemos ejecutar de nuevo.



Podemos modificar el tamaño y cuando lo restauramos.



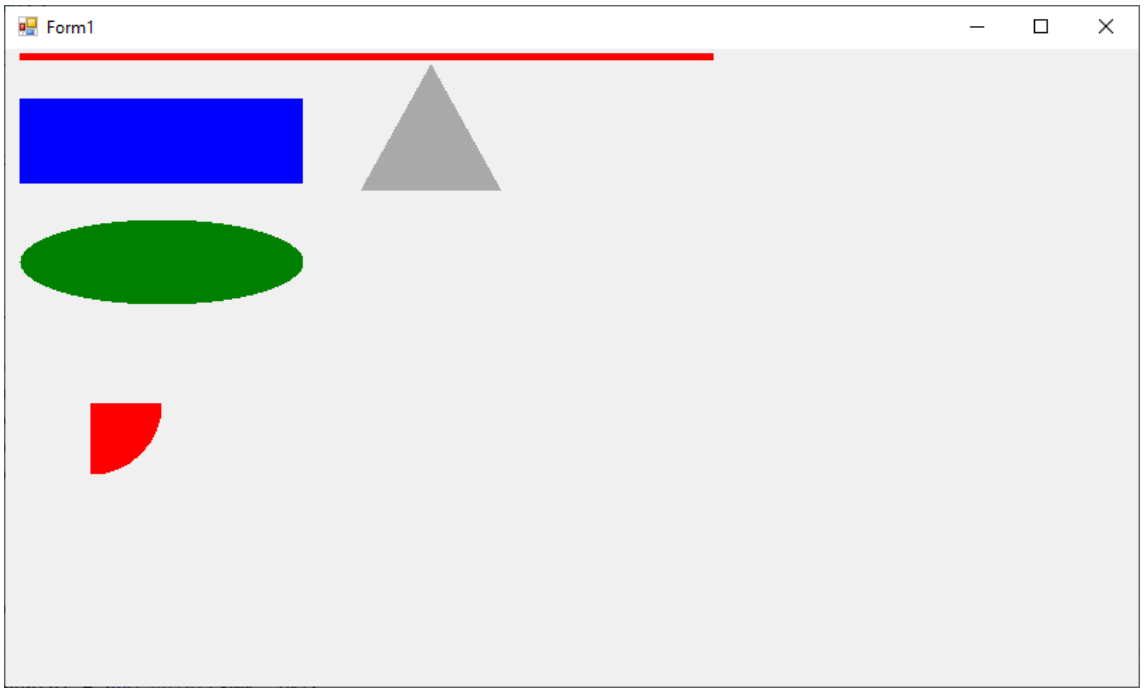
No se han borrado los dibujos.

Evento Paint dibuja de nuevo cada vez que se redimensiona el formulario.

Ahora veremos que tenemos que hacer para que aparezcan las figuras rellenas.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics lienzo = CreateGraphics();
    lienzo.DrawLine(new Pen(Color.Red, 5), 10, 5, 500, 5);
    lienzo.FillRectangle(new SolidBrush(Color.Blue), 10, 35, 200, 60);
    lienzo.FillEllipse(new SolidBrush(Color.Green), 10, 120, 200, 60);
    lienzo.FillPie(new SolidBrush(Color.Red), 10, 200, 100, 100, 0, 90);
    Point punto1 = new Point(300, 10);
    Point punto2 = new Point(350, 100);
    Point punto3 = new Point(250, 100);
    Point[] puntos = { punto1, punto2, punto3 };
    lienzo.FillPolygon(new SolidBrush(Color.DarkGray), puntos);
}
```

Cambiamos Draw por Fill y Pen por SolidBrush.

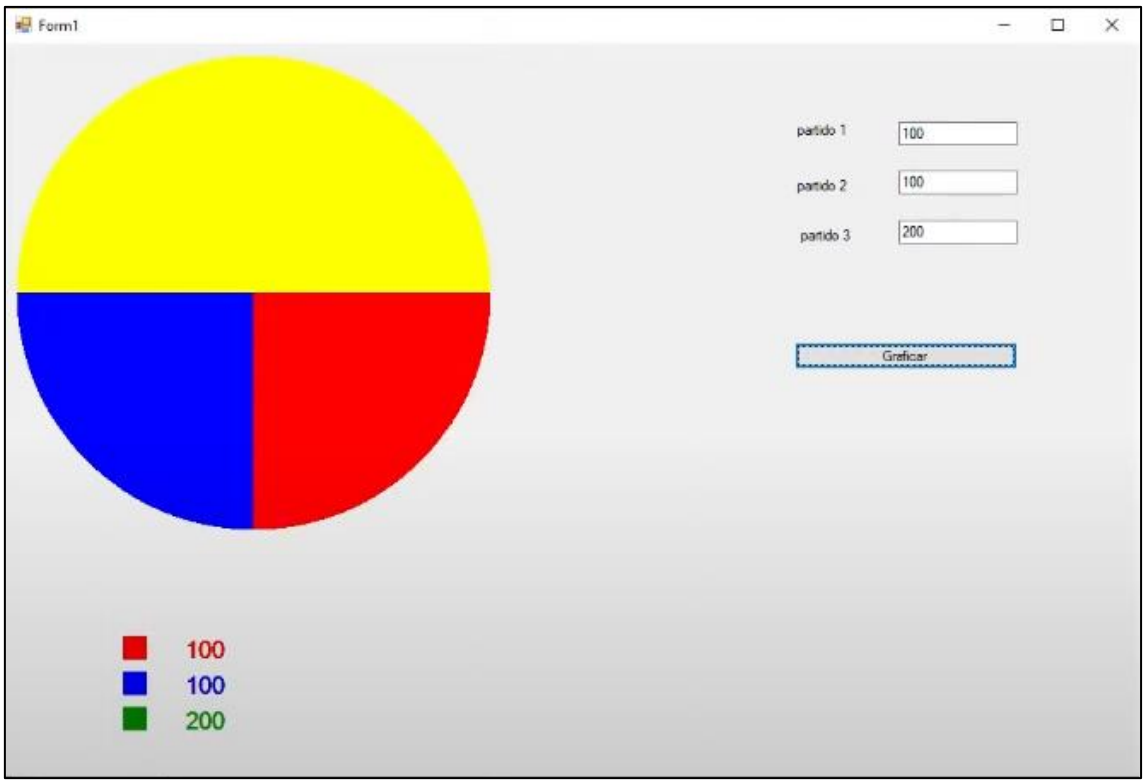


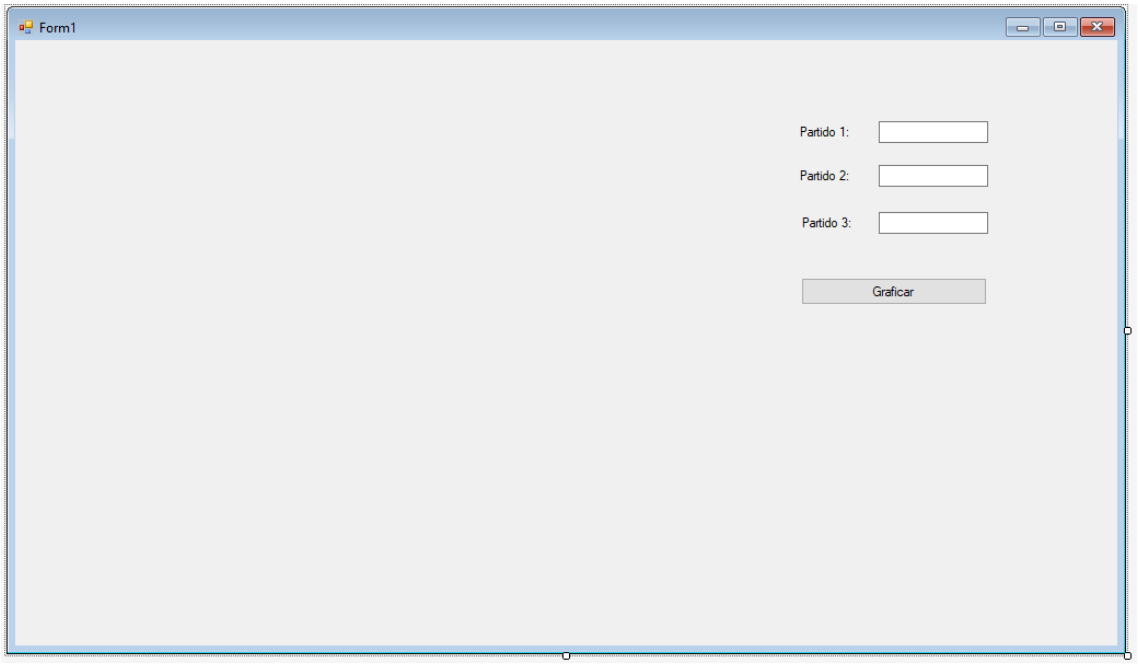
Capítulo 134.- Graphics – Ejercicio de gráfico de tarta

Gráfico de tarta

Confeccionar una aplicación que permita el ingreso de la cantidad de votos de tres partidos políticos.

Generar un gráfico de tarta al presionar un botón. Utilizar la clase Graphics.





Hemos agregado 3 Label, 3 TextBox y 1 Bitton.

Vamos a programar.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto147
{
    3 referencias
    public partial class Form1 : Form
    {
        private bool estado = false;
        private int partido1, partido2, partido3;

        1 referencia
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            if (estado)
            {
                Graphics lienzo = CreateGraphics();
                int totalVotos = partido1 + partido2 + partido3;
                int grados1 = partido1 * 360 / totalVotos;
                int grados2 = partido2 * 360 / totalVotos;
                int grados3 = partido3 * 360 / totalVotos;
                lienzo.FillPie(new SolidBrush(Color.Red), 10, 10, 400, 400, 0, grados1);
                lienzo.FillPie(new SolidBrush(Color.Blue), 10, 10, 400, 400, grados1, grados2);
                lienzo.FillPie(new SolidBrush(Color.Yellow), 10, 10, 400, 400, grados1 + grados2, grados3);
                lienzo.FillRectangle(new SolidBrush(Color.Red), 100, 450, 20, 20);
                lienzo.DrawString(partido1.ToString(), new Font("Arial", 15), new SolidBrush(Color.Red), 150, 450);
                lienzo.FillRectangle(new SolidBrush(Color.Blue), 100, 480, 20, 20);
                lienzo.DrawString(partido2.ToString(), new Font("Arial", 15), new SolidBrush(Color.Blue), 150, 480);
                lienzo.FillRectangle(new SolidBrush(Color.Yellow), 100, 510, 20, 20);
                lienzo.DrawString(partido3.ToString(), new Font("Arial", 15), new SolidBrush(Color.Yellow), 150, 510);
            }
        }

        1 referencia
        public Form1()
        {
            InitializeComponent();
        }

        1 referencia
        private void button1_Click(object sender, EventArgs e)
        {
            partido1 = int.Parse(textBox1.Text);
            partido2 = int.Parse(textBox2.Text);
            partido3 = int.Parse(textBox3.Text);
            estado = true;
            Refresh();
        }
    }
}
```

Vamos a comentar el código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto147
{
    public partial class Form1 : Form
    {
        private bool estado = false;
        private int partido1, partido2, partido3;
```

Definimos las variables globales estado de tipo boolean y partido1, partido2 y partido3 de tipo int.

El evento Paint del formulario repinta cada vez que modificamos el tamaño de la ventana.

```
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            if (estado)
```

Si estado vale true.

```
            {
                Graphics lienzo = CreateGraphics();
```

Creamos un objeto llamado lienzo de la clase Graphics.

```
                int totalVotos = partido1 + partido2 + partido3;
                int grados1 = partido1 * 360 / totalVotos;
                int grados2 = partido2 * 360 / totalVotos;
                int grados3 = partido3 * 360 / totalVotos;
```

La variable totalVotal suma el total de votos de los tres partidos.
Las variables grados1, grados2 y grados3 se le asignan los grados que tendrán con respecto al 360 grados de totalVotos.

```
                lienzo.FillPie(new SolidBrush(Color.Red), 10, 10, 400, 400, 0, grados1);
                lienzo.FillPie(new SolidBrush(Color.Blue), 10, 10, 400, 400, grados1,
grados2);
                lienzo.FillPie(new SolidBrush(Color.Yellow), 10, 10, 400, 400, grados1 +
grados2, grados3);
```

Dibujamos una tarta con cada porcenaje de las variables grados1, grados2 y grados3.

```
                lienzo.FillRectangle(new SolidBrush(Color.Red), 100, 450, 20, 20);
                lienzo.DrawString(partido1.ToString(), new Font("Arial", 15), new
SolidBrush(Color.Red), 150, 450);
                lienzo.FillRectangle(new SolidBrush(Color.Blue), 100, 480, 20, 20);
                lienzo.DrawString(partido2.ToString(), new Font("Arial", 15), new
SolidBrush(Color.Blue), 150, 480);
                lienzo.FillRectangle(new SolidBrush(Color.Yellow), 100, 510, 20, 20);
                lienzo.DrawString(partido3.ToString(), new Font("Arial", 15), new
SolidBrush(Color.Yellow), 150, 510);
```

Dibujamos los cuadrados y los valores de cada cuadrado.

```
            }
        }

        public Form1()
        {
            InitializeComponent();
        }
```

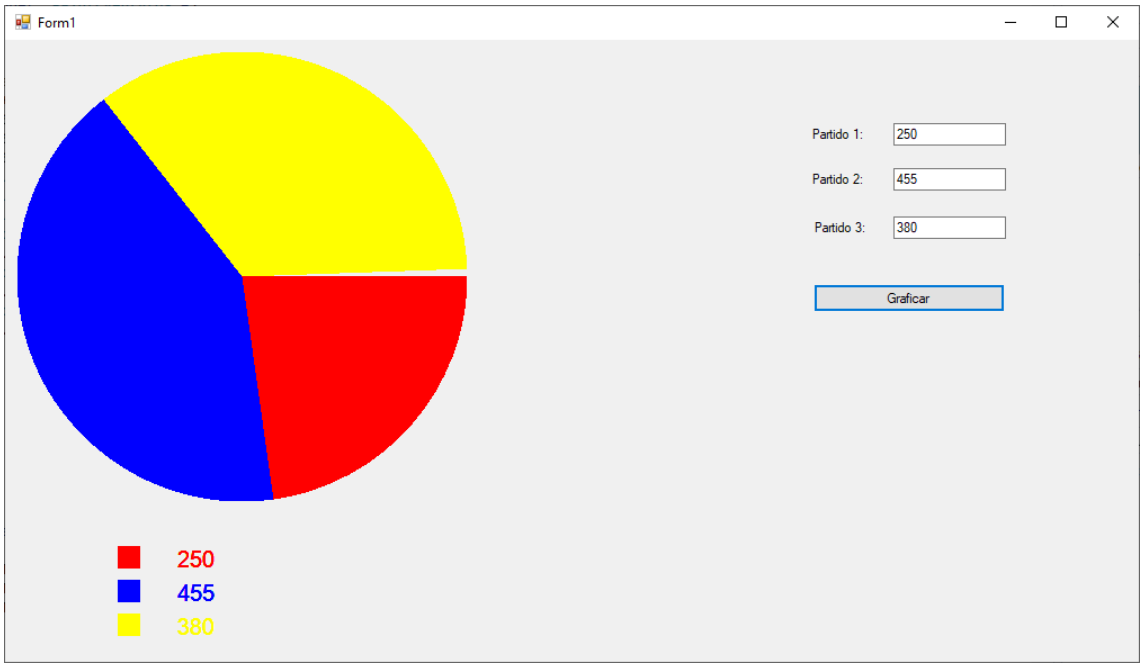
El evento click del botón realiza lo siguiente:

```
        private void button1_Click(object sender, EventArgs e)
        {
            partido1 = int.Parse(textBox1.Text);
            partido2 = int.Parse(textBox2.Text);
            partido3 = int.Parse(textBox3.Text);
            estado = true;
            Refresh();
        }
```

A las variables partido1, partido2 y partido2 le asignamos los valores de los textBox1, textBox2 y textBox3 como valor entero.
La variable estado pasa a valer true.

```
    }
}
```

Vamos a ejecutar introduciendo los valor 250, 455 y 380.



Capítulo 135.- Clase Graphics – Ejercicio de gráfico de barras

Gráfico de barras

Confeccionar una aplicación que permita el ingreso de la cantidad de votos de tres partidos políticos.

Generar un gráfico de barras al presionar un botón.

Utilizar la clase Graphics.

Este será el código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto149
{
    public partial class Form1 : Form
    {
        private bool estado = false;
        private int partidoA, partidoB, partidoC;

        private void button1_Click(object sender, EventArgs e)
        {
            partidoA = int.Parse(textBox1.Text);
            partidoB = int.Parse(textBox2.Text);
            partidoC = int.Parse(textBox3.Text);
            estado = true;
            Refresh();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            if (estado)
            {
                Graphics lienzo = CreateGraphics();
```

```
                int totalVotos = partidoA + partidoB + partidoC;
                int porcentaje1 = partidoA * 100 / totalVotos;
                int porcentaje2 = partidoB * 100 / totalVotos;
                int porcentaje3 = partidoC * 100 / totalVotos;
                lienzo.FillRectangle(new SolidBrush(Color.Red), 10, 10,
700*porcentaje1/100, 80);
                lienzo.DrawString("Partido A: "+ partidoA.ToString(), new Font("Arial",
15), new SolidBrush(Color.Black), 20, 40);
                lienzo.FillRectangle(new SolidBrush(Color.DarkCyan), 10, 110,
700*porcentaje2/100, 80);
                lienzo.DrawString("Partido B: " + partidoB.ToString(), new Font("Arial",
15), new SolidBrush(Color.Black), 20, 140);
                lienzo.FillRectangle(new SolidBrush(Color.Green), 10, 210,
700*porcentaje3/100, 80);
                lienzo.DrawString("Partido C: " + partidoC.ToString(), new Font("Arial",
15), new SolidBrush(Color.Black), 20, 240);
            }
        }

        public Form1()
        {
            InitializeComponent();
        }

    }
}
```

Vamos a ver otra forma de realizarlo.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto149
{
    public partial class Form1 : Form
    {
        private bool estado = false;
        private int partidoA, partidoB, partidoC;

        private void button1_Click(object sender, EventArgs e)
        {
            partidoA = int.Parse(textBox1.Text);
            partidoB = int.Parse(textBox2.Text);
            partidoC = int.Parse(textBox3.Text);
            estado = true;
            Refresh();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            if (estado)
            {
                int mayor;
                if (partidoA>partidoB && partidoA>partidoC)
                {
                    mayor = partidoA;
                }
                else
                {
                    if (partidoB>partidoC)
                    {
                        mayor = partidoB;
                    }
                    else
                    {
                        mayor = partidoC;
```

```
                    }
                }
                int largo1 = partidoA * 400 / mayor;
                int largo2 = partidoB * 400 / mayor;
                int largo3 = partidoC * 400 / mayor;
                Graphics lienzo = CreateGraphics();
                lienzo.FillRectangle(new SolidBrush(Color.Red), 10, 10, largo1, 80);
                lienzo.DrawString("Partido A: "+ partidoA.ToString(), new Font("Arial",
15), new SolidBrush(Color.Black), 20, 40);
                lienzo.FillRectangle(new SolidBrush(Color.DarkCyan), 10, 110, largo2, 80);
                lienzo.DrawString("Partido B: " + partidoB.ToString(), new Font("Arial",
15), new SolidBrush(Color.Black), 20, 140);
                lienzo.FillRectangle(new SolidBrush(Color.Green), 10, 210, largo3, 80);
                lienzo.DrawString("Partido C: " + partidoC.ToString(), new Font("Arial",
15), new SolidBrush(Color.Black), 20, 240);
            }
        }

        public Form1()
        {
            InitializeComponent();
        }

    }
}
```

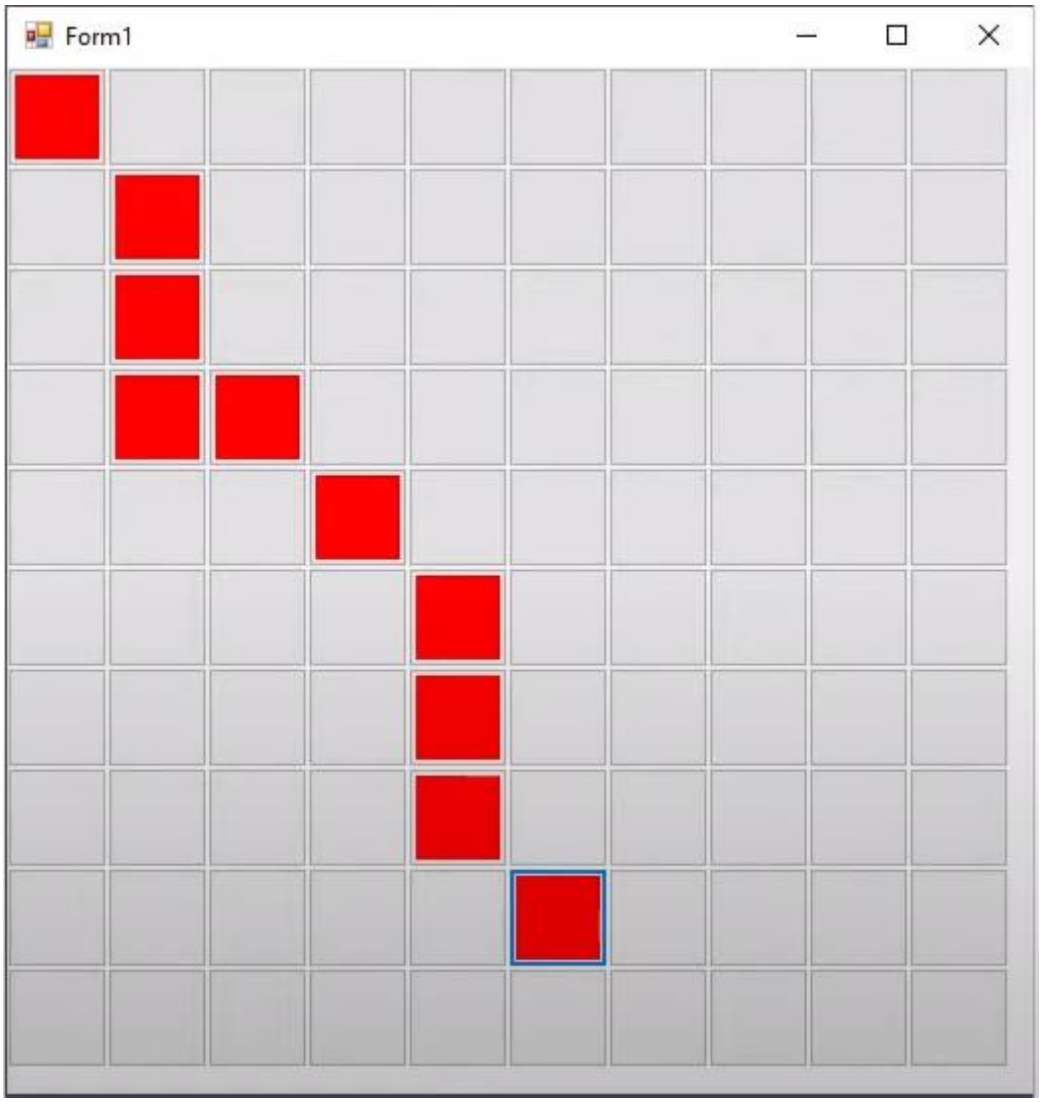
Las diferencias se encuentran en el método Form1_Paint.

En este ejemplo las barras se adaptan al valor mayor.

Capítulo 136.- Creación de controles en forma dinámica (Windows Form)

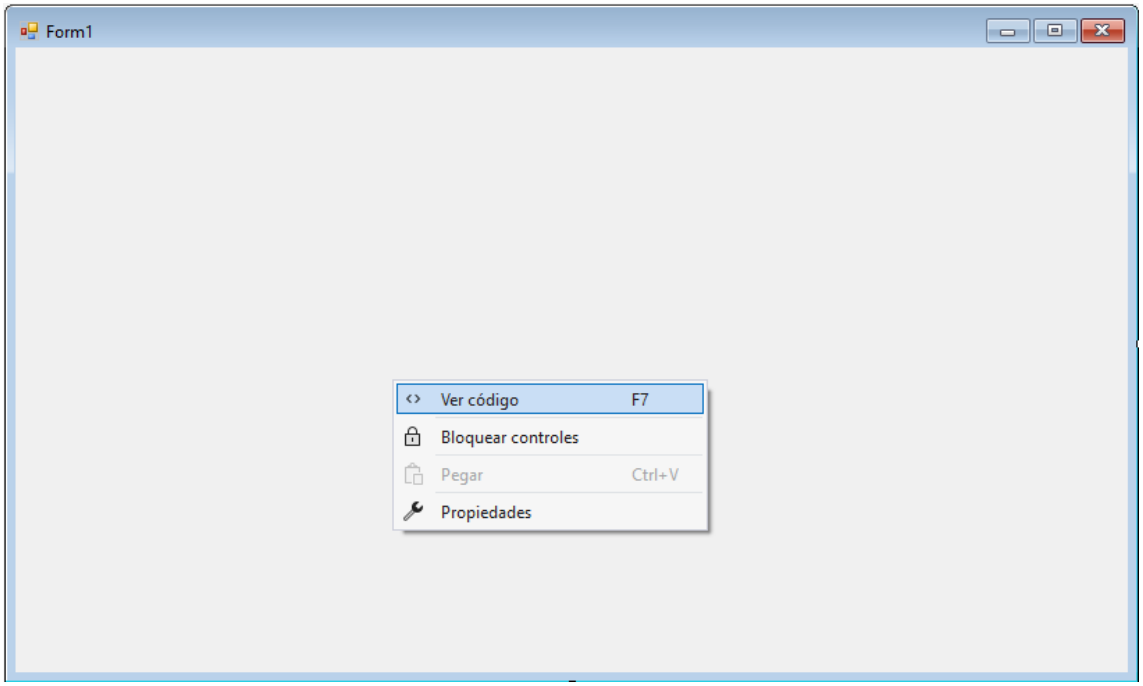
Problema propuesto

Crear una matriz en pantalla de 10*10 elementos de tipo Button. Cuando se presiona el mismo cambia el color de dicho botón.



En este ejercicio lo que no es viable es insertar los botones uno a uno.

Vamos a crear los botones de forma dinámica.



Botón derecho del mouse sobre el formulario y del menú seleccionaremos "Ver código".

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto150
{
    3 referencias
    public partial class Form1 : Form
    {
        private Button[,] botones = new Button[10,10];

        1 referencia
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Creamos una matiz de 10 x 10 de tipo Button.

Cerramos este código para agregar un nuevo código al evento Load del control Formulario.

Este será el código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto150
{
    public partial class Form1 : Form
    {
        private Button[,] botones = new Button[10, 10];

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            for (int f = 0; f < botones.GetLength(0); f++)
            {
                for (int c = 0; c < botones.GetLength(1); c++)
                {
                    botones[f, c] = new Button();
                    botones[f, c].Location = new Point(f * 50, c * 50);
                    botones[f, c].Size = new Size(50, 50);
                    botones[f, c].Click += Presion_Boton;
                    Controls.Add(botones[f, c]);
                }
            }
        }

        private void Presion_Boton(object sender, EventArgs e)
        {
            for (int f = 0; f < botones.GetLength(0); f++)
            {
                for (int c = 0; c < botones.GetLength(1); c++)
                {
                    if (sender == botones[f, c])
                    {
                        botones[f,c].BackColor = Color.Red;
                    }
                }

            }

        }
    }
}
```

Presione TAB para insertar, el nombre que sale lo podemos cambiar.

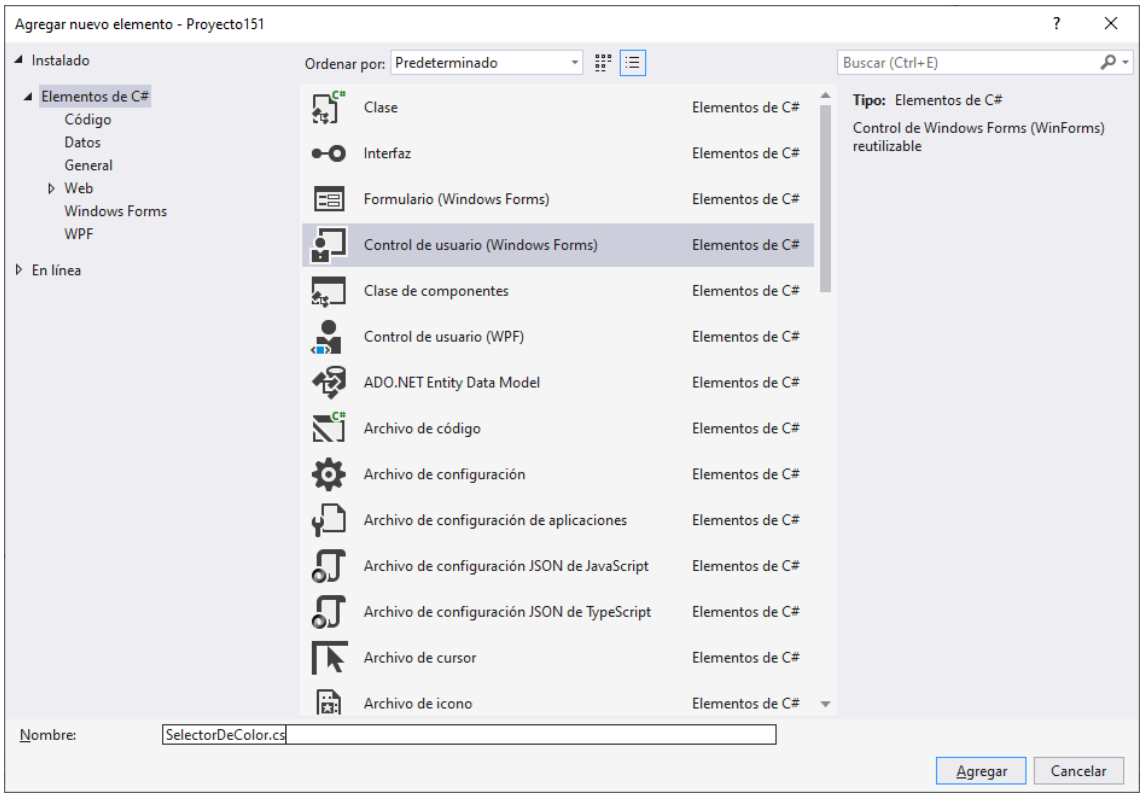
Nos crea el método.

Comparamos el botón que hemos presionado.

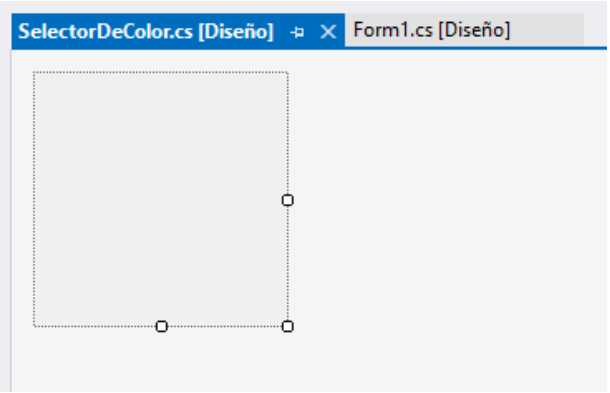
Capítulo 137.- Creación de controles de usuario – 1

Vamos a crear un selector de color.

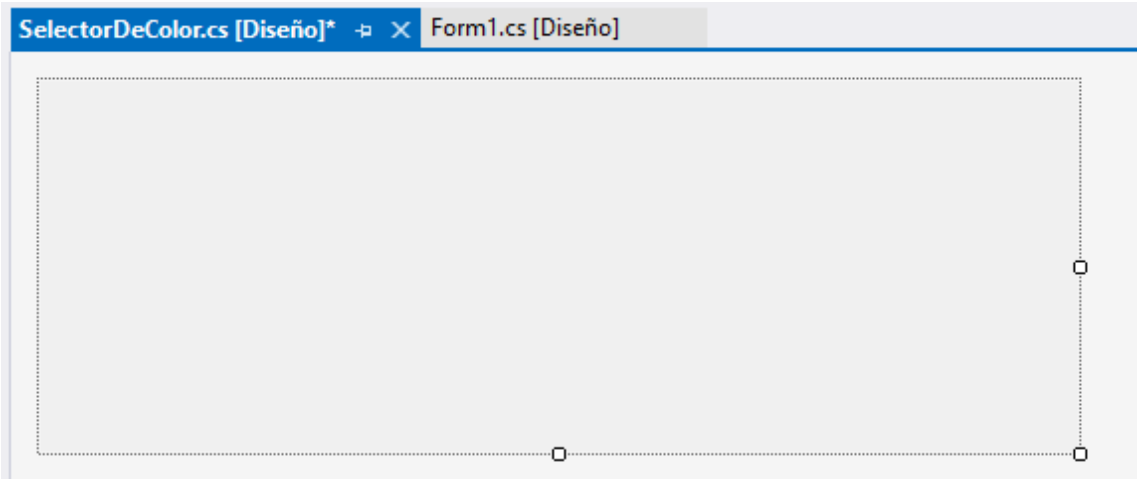
Para la creación de controles botón derecho sobre el nombre del proyecto en explorador de soluciones, seguido de agregar y por último Control de usuario (Windows Forms)…



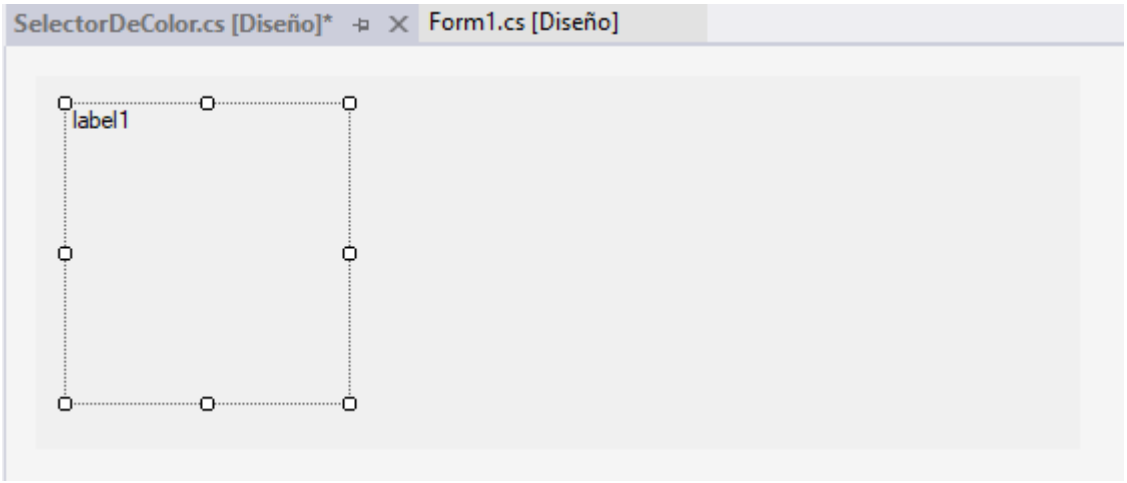
Seleccionaremos Control de usuario (Windows Forms) como nombre pondremos SelectorDeColor.cs seguido del botón Agregar.



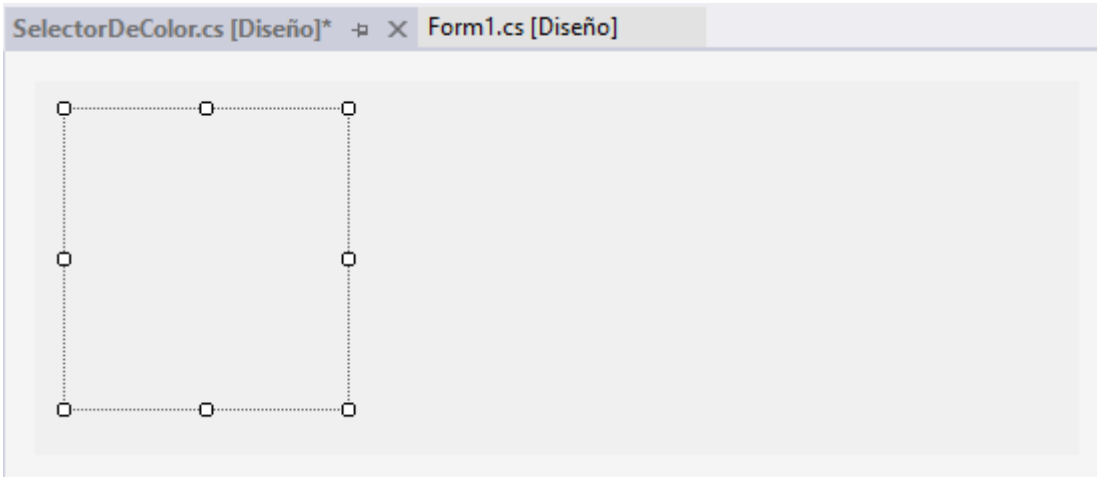
También la podemos redimensionar.



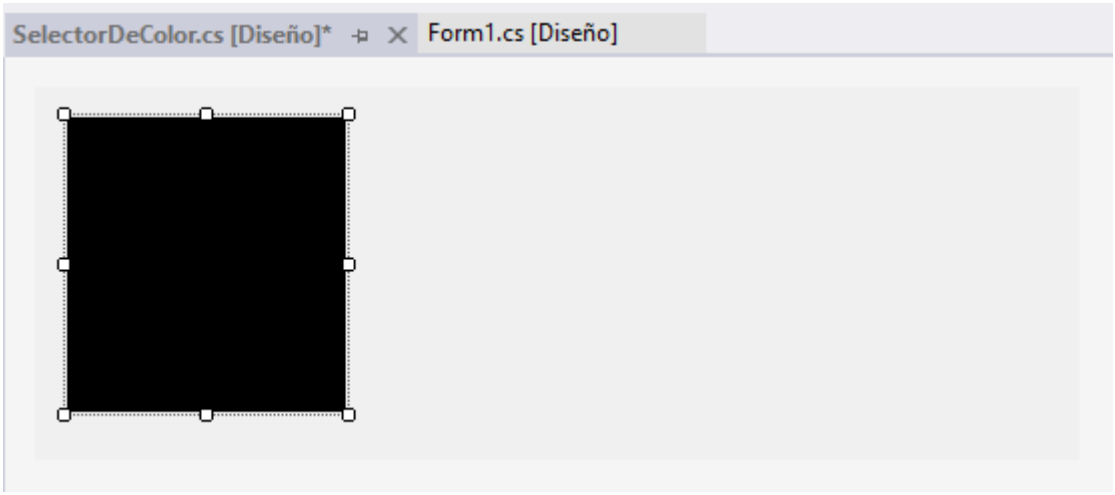
En la parte izquierda vamos a agregar un label, modificaremos la propiedad AutoSize como false y así nos permitirá redimensionarla.



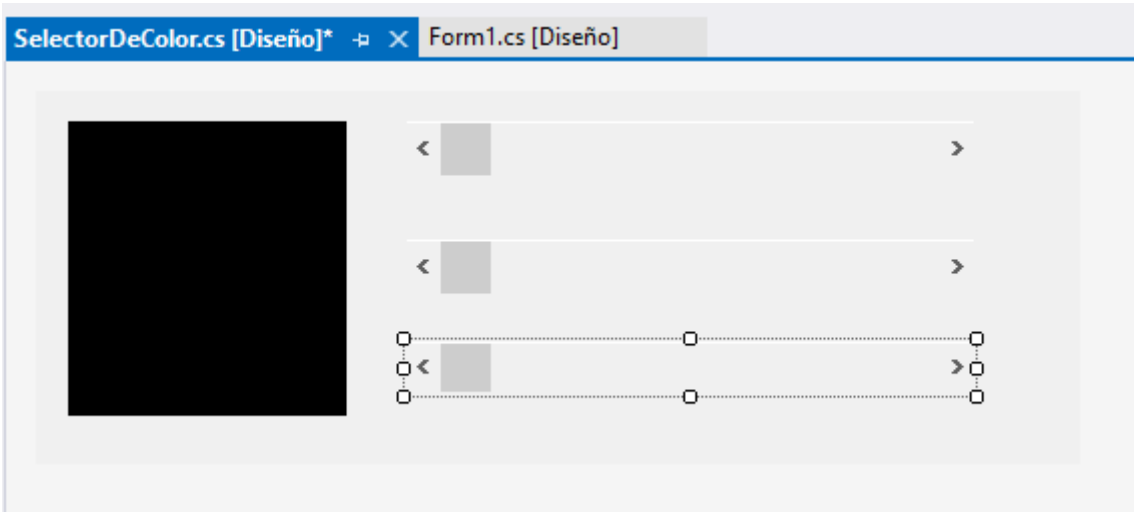
En la propiedad Text quitamos el texto.



Cambiaremos la propiedad BackColor a color negro.

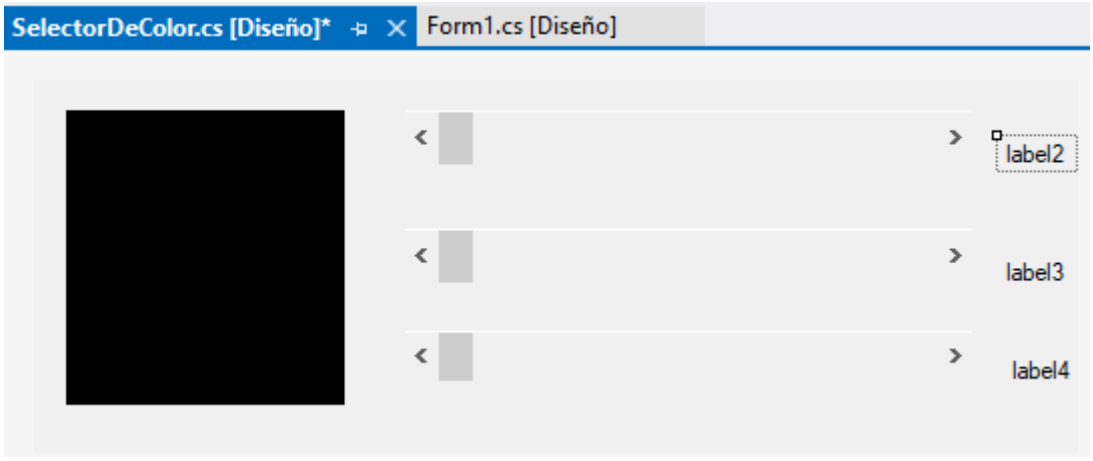


Ahora seleccionaremos del Cuadro de herramientas Todo Windows Form y arrastraremos tres HsScrollBar, que a continuación modificaremos sus dimensiones.

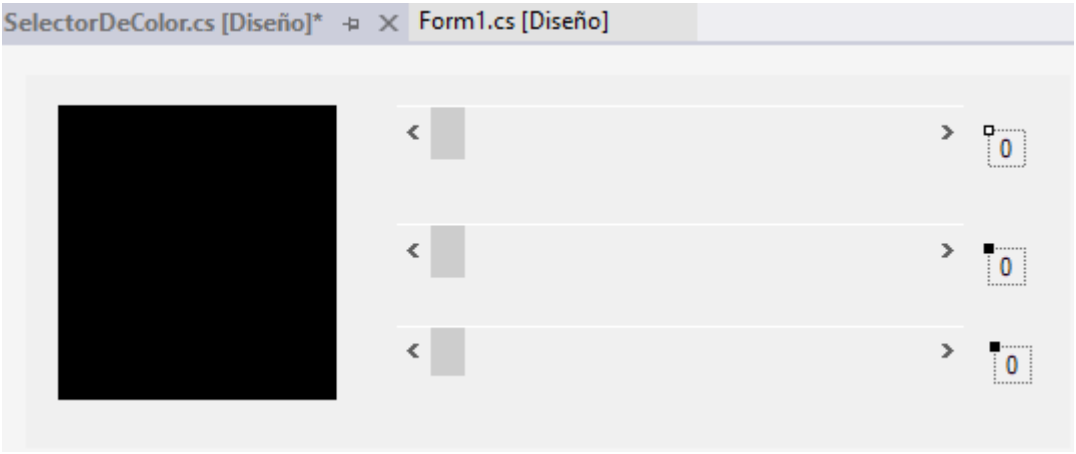


Con ayuda de la tecla Ctrol seleccionaremos los tres HsScrollBar y modificaremos las propiedades Minimun con 0, Maximoun con 255 y LargeChange en 1.

En la parte derecha de cada HsCrollBar vamos a agregar un label.



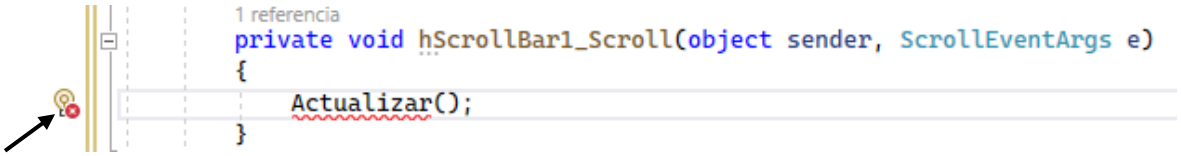
Seleccionamos las 3 label con ayuda de la tecla control y cambiamos la propiedad Text a 0.



A continuación botón derecho sobre el formulario y del menú seleccionaremos Ver código y agregaremos el siguiente código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto151
{
    2 referencias
    public partial class SelectorDeColor : UserControl
    {
        private Color colorSeleccionado = Color.Black;
        0 referencias
        public Color ColorSeleccionado
        {
            get
            {
                return colorSeleccionado;
            }
            set
            {
                colorSeleccionado = value;
            }
        }

        0 referencias
        public SelectorDeColor()
        {
            InitializeComponent();
        }
    }
}
```

Volvemos al diseño y haremos doble clic sobre el primer HsScrollBar.



Llamamos al método Actualizar() como este método no existe utilizaremos el icono de la izquierda para que nos lo genere.

```
1 referencia
private void Actualizar()
{
    throw new NotImplementedException();
}
```

Borramos la línea seleccionada y vamos a escribir el código.

```
1 referencia
private void Actualizar()
{
    ColorSeleccionado = Color.FromArgb(hScrollBar1.Value, hScrollBar2.Value, hScrollBar3.Value);
    label1.BackColor = ColorSeleccionado;
    label2.Text = hScrollBar1.Value.ToString();
    label3.Text = hScrollBar2.Value.ToString();
    label4.Text = hScrollBar3.Value.ToString();
}
```

A continuación desde el hSCrollBar2 y hSCrollBar3 vamos a llamar al método Actualizar.

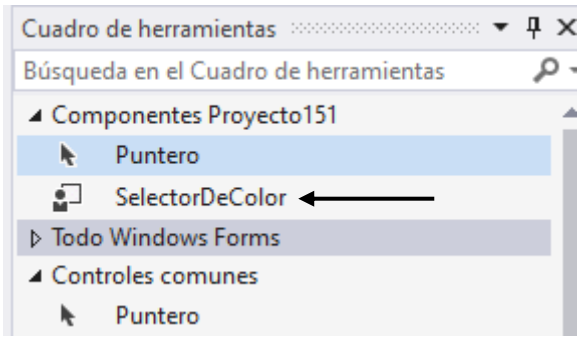
```
1 referencia
private void hScrollBar2_Scroll(object sender, ScrollEventArgs e)
{
    Actualizar();
}
```

```
1 referencia
private void hScrollBar3_Scroll(object sender, ScrollEventArgs e)
{
    Actualizar();
}
```

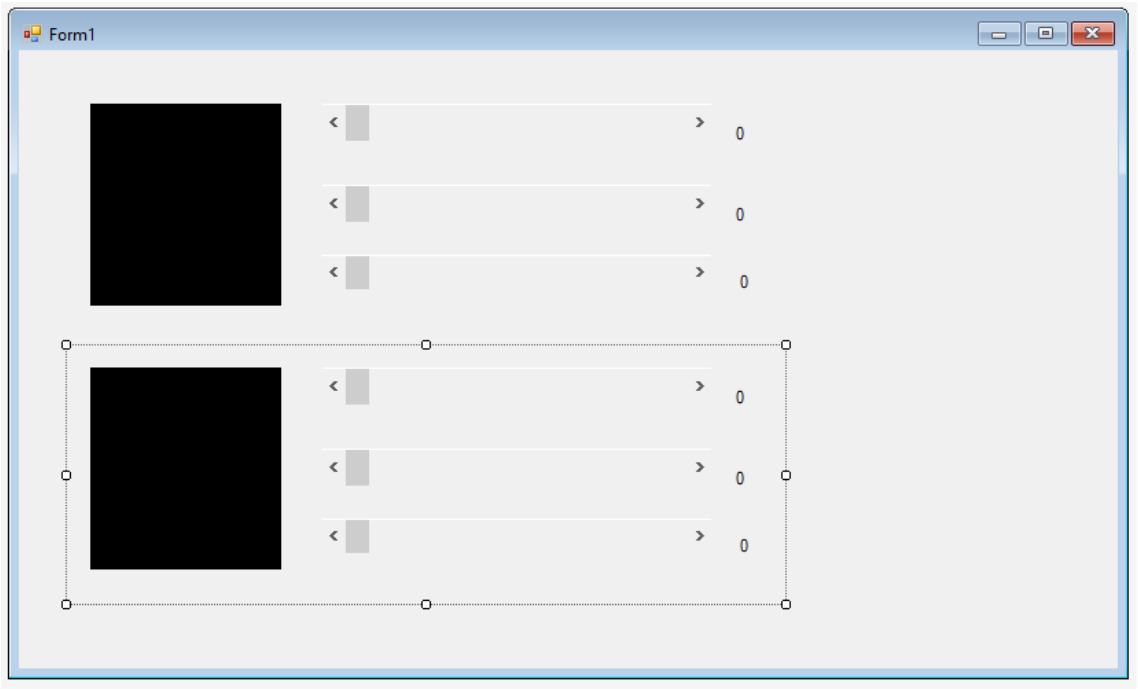
El control que acabamos de crear no lo podemos ejecutar directamente.

Para ello del menú Compilar seleccionaremos Compilar la solución.

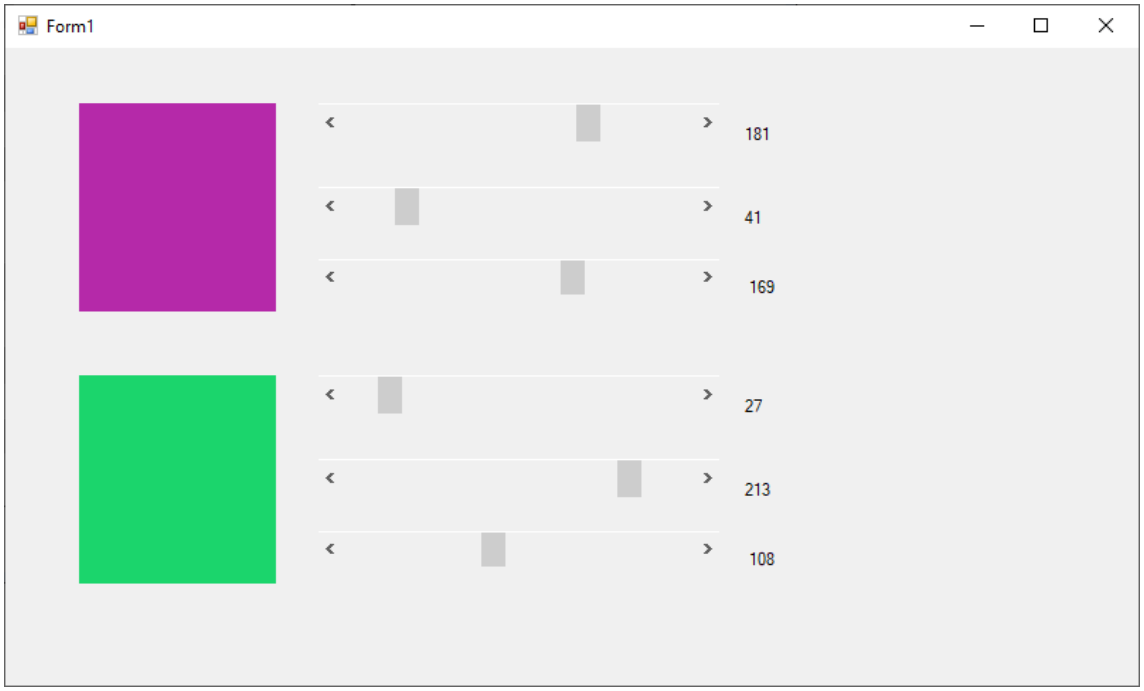
Seleccionaremos el formulario principal.



En el cuadro de herramientas parece otro grupo llamado "Componentes Proyecto..." y encontramos nuestro selector de color que lo arrastraremos al formulario dos veces.

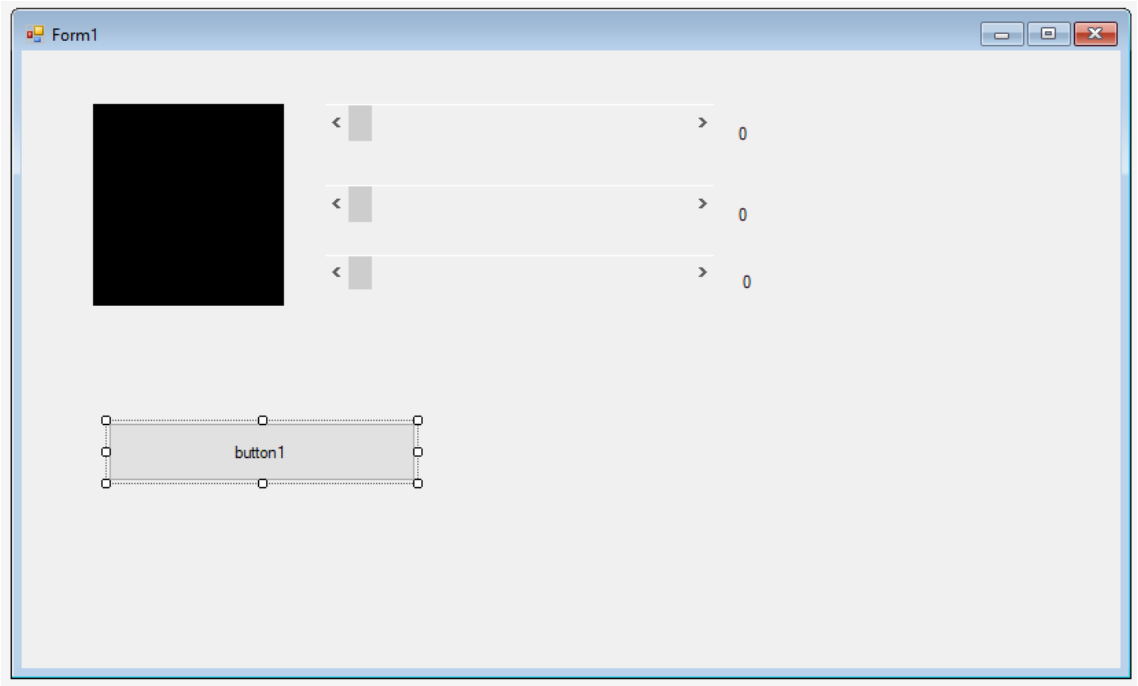


Vamos a ejecutar.



Este nuevo control lo podemos utilizar dentro de nuestra aplicación.

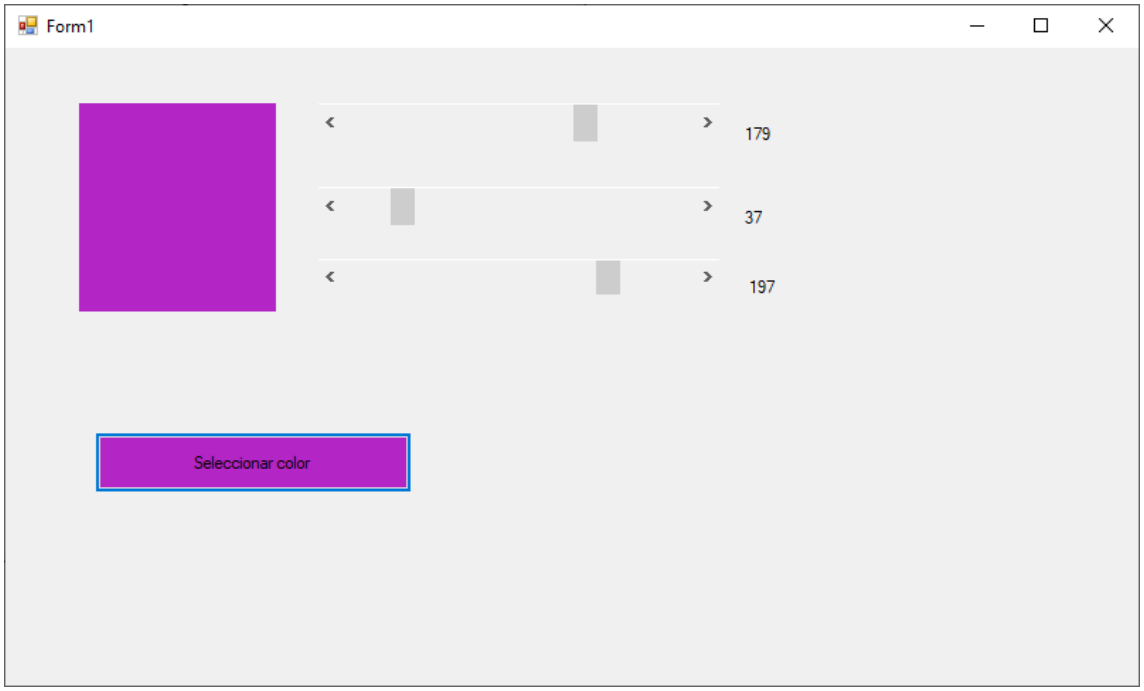
Vamos a modificar el formulario, eliminamos un selector de color y vamos a agregar un botón.



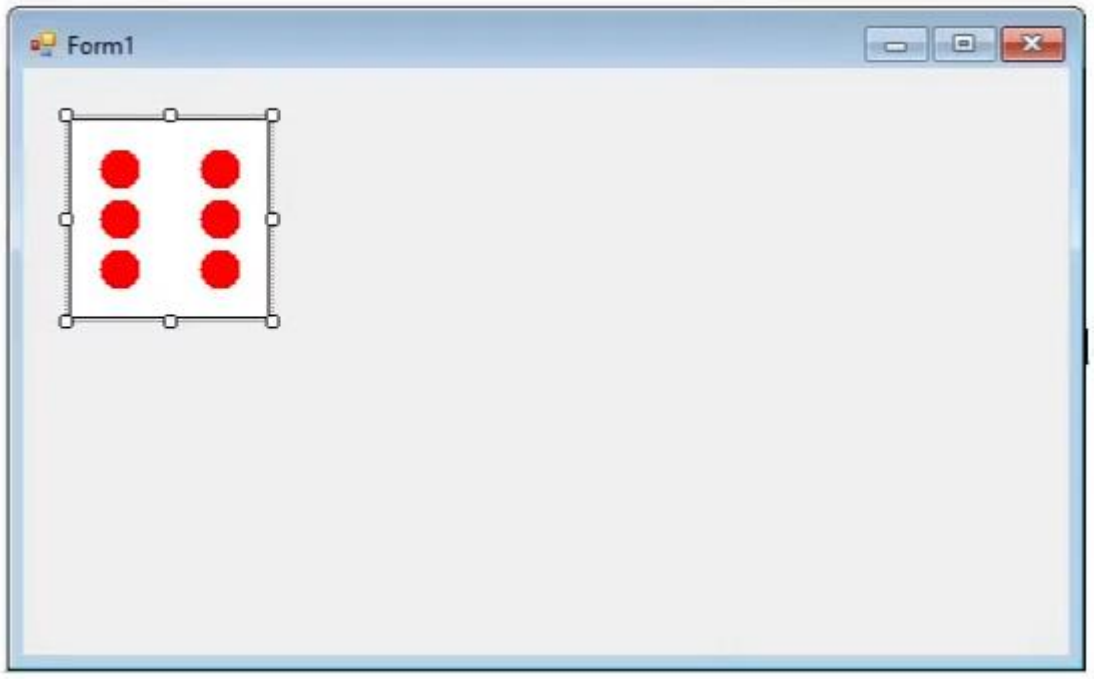
Hacemos click sobre él y escribiremos el siguiente código:

```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    button1.BackColor = selectorDeColor1.ColorSeleccionado;
}
```

Ahora cuando ejecutemos y seleccionemos un determinado color presionaremos el botón para que se le asigne el mismo color.

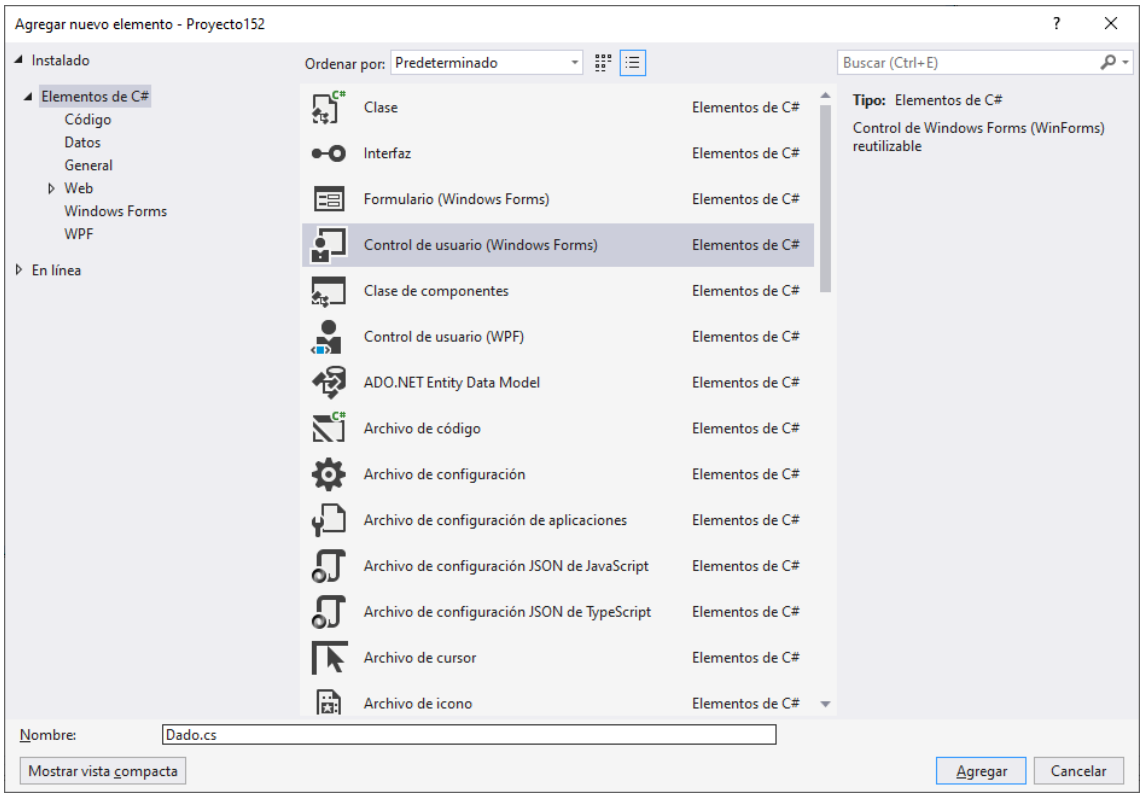


Capítulo 138.- Creación de controles de usuario – 2

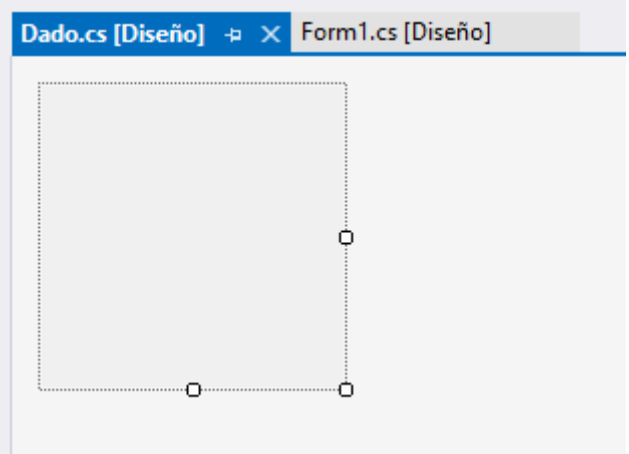


Vamos a realizar un componente que sea un dado.

Para realizar este componente botón derecho sobre el nombre del proyecto del explorador de soluciones, seleccionaremos Agregar y de este nuevo elemento.



Seleccionaremos Control de usuario (Windows Forms) y como nombre le daremos Dado.cs, seguido del botón Agregar.

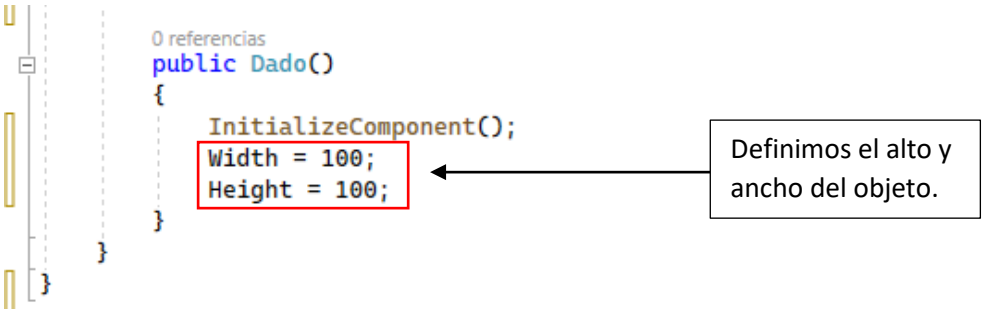


Seleccionaremos este objeto con el botón derecho del mouse y seleccionaremos "Ver código".

Vamos a escribir el siguiente código:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto152
{
    2 referencias
    public partial class Dado : UserControl
    {
        private int valor = 1;
        0 referencias
        public int Valor
        {
            get
            {
                return valor;
            }
            set
            {
                if (value>=1 && value <=6)
                {
                    valor = value;
                    Invalidate();
                }
                else
                {
                    MessageBox.Show("Debe ingresar un valor entre 1 y 6");
                }
            }
        }
```

Invalida toda la superficie del control y hace que se vuelva a dibujar el control.



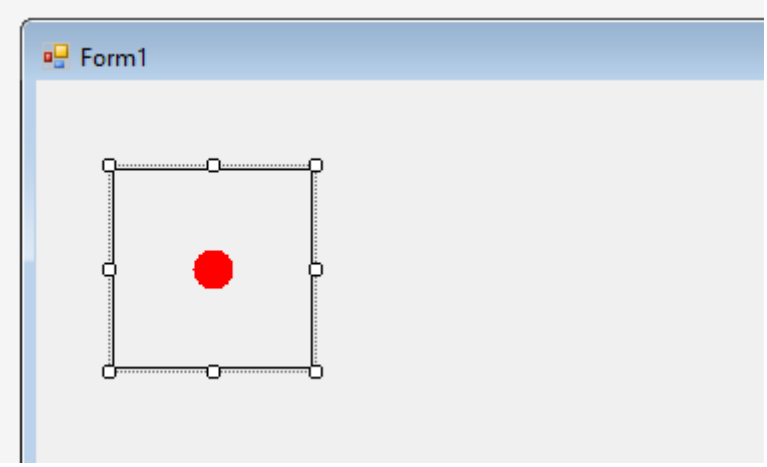
En modo edición seleccionamos el objeto y seleccionamos el evento Paint.

Vamos a escribir el siguiente código:

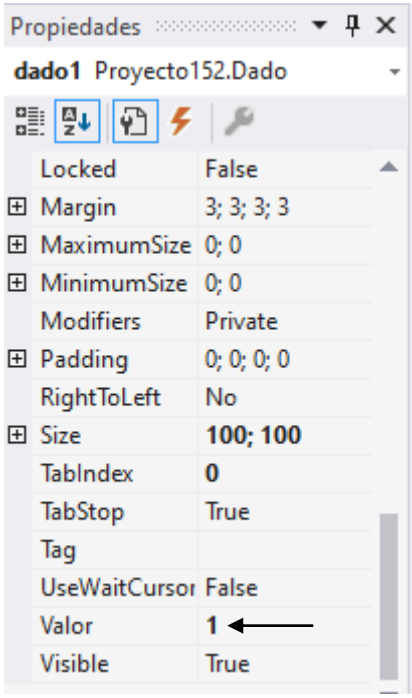
```
private void Dado_Paint(object sender, PaintEventArgs e)
        {
            Graphics lienzo = CreateGraphics();
            lienzo.DrawRectangle(new Pen(Color.Black), 0,0, Width-1, Height-1);
            int diametro = Width / 5;
            if (Valor==1)
            {
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.5f) - (diametro /
2), (Height * 0.5f) - (diametro / 2), diametro, diametro);
            }
            if (Valor==2)
            {
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
            }
            if (Valor == 3)
            {
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.50f) - (diametro
/ 2), (Height * 0.50f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
            }
            if (Valor == 4)
            {
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
            }
            if (Valor == 5)
            {
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.5f) - (diametro /
2), (Height * 0.5f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
            }
            if (Valor == 6)
            {
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.25f) - (diametro / 2), diametro, diametro);
```

```
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.50f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.50f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.25f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);
                lienzo.FillEllipse(new SolidBrush(Color.Red), (Width * 0.75f) - (diametro
/ 2), (Height * 0.75f) - (diametro / 2), diametro, diametro);

            }
        }
```

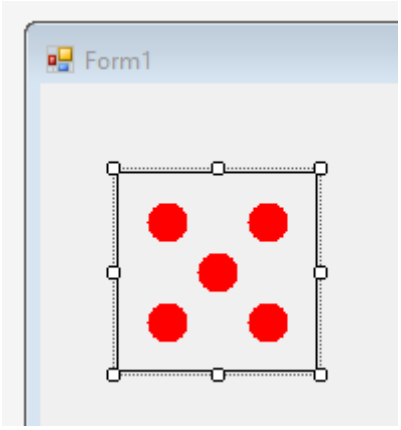
Del menú compilar seleccionaremos Compilar solución.



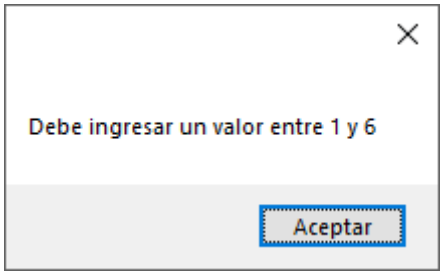
Nos vamos al diseño del formulario principal y en propiedades vamos a buscar la propiedad valor.



Po defecto pone el valor 1 lo puedes ir cambiando entre valores del 1 al 6 vamos a cambiar por el número 5.



Ahora vamos a poner un valor que no esté entre el 1 y el 6 pondremos un 7.

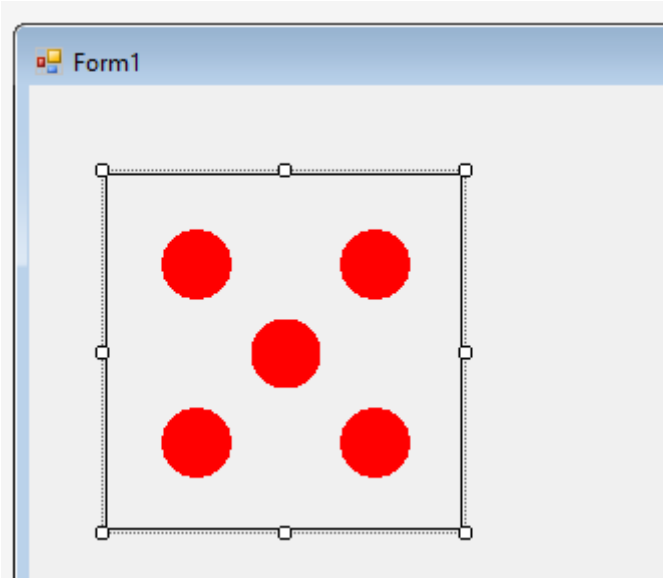


Aparece un mensaje indicando que le valor tiene que ser entre 1 y 6.

Ahora para hacer que siempre sea un cuadrado del control Dado vamos a codificar el evento Resize.

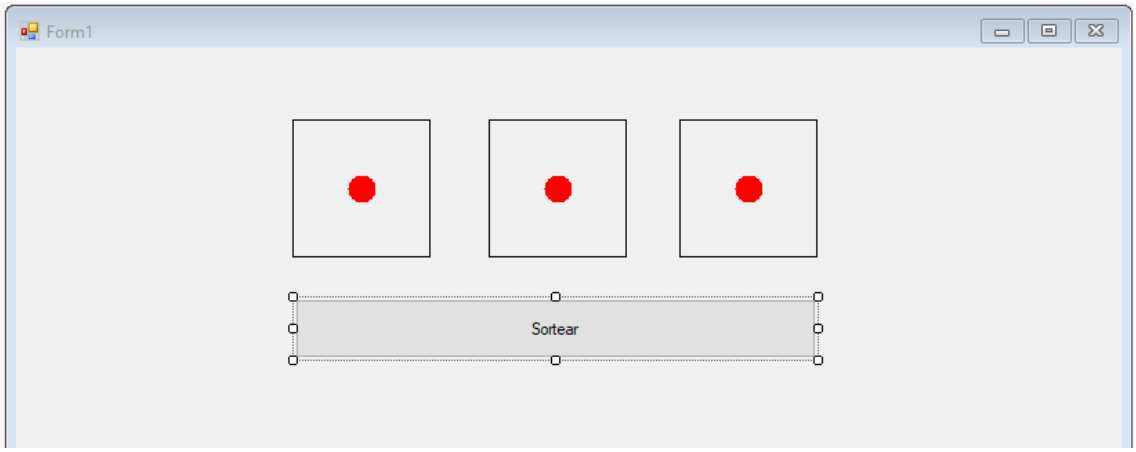
```
1 referencia
private void Dado_Resize(object sender, EventArgs e)
{
    Width = Height;
}
```

Compilamos de nuevo.



Podemos modificar el tamaño pero siempre será cuadrado.

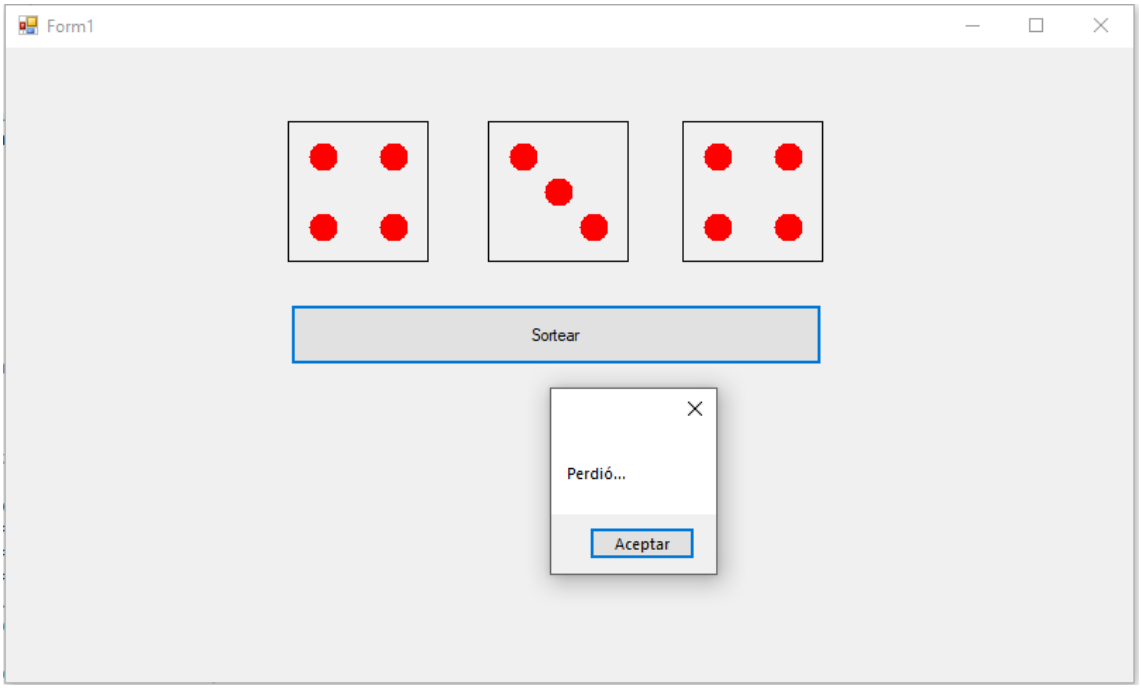
Desde el formulario principal vamos a realizar el siguiente diseño.



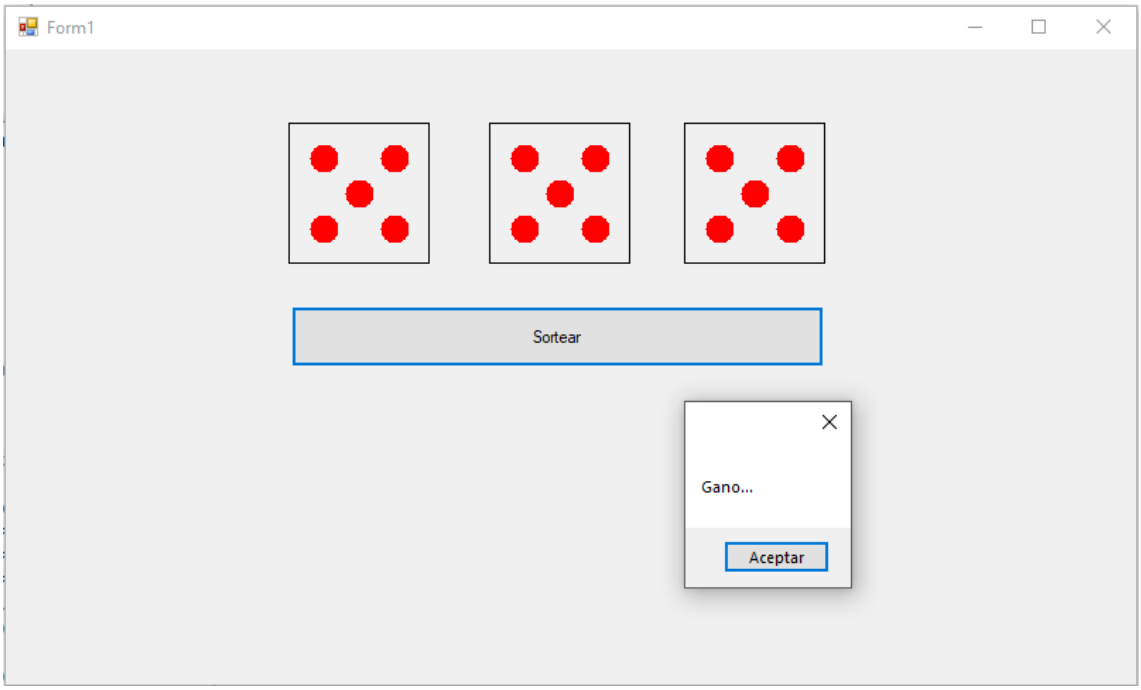
Vamos a codificar el botón Sortear en el evento click.

```
private void button1_Click(object sender, EventArgs e)
{
    Random aleatorio = new Random();
    dado1.Valor = aleatorio.Next(1, 7);
    dado2.Valor = aleatorio.Next(1, 7);
    dado3.Valor = aleatorio.Next(1, 7);
    if (dado1.Valor == dado2.Valor && dado1.Valor==dado3.Valor)
        MessageBox.Show("Gano...");
    else
        MessageBox.Show("Perdió...");
}
```

Vamos a ejecutar:



Le damos de nuevo al botón.



He tenido suerte.

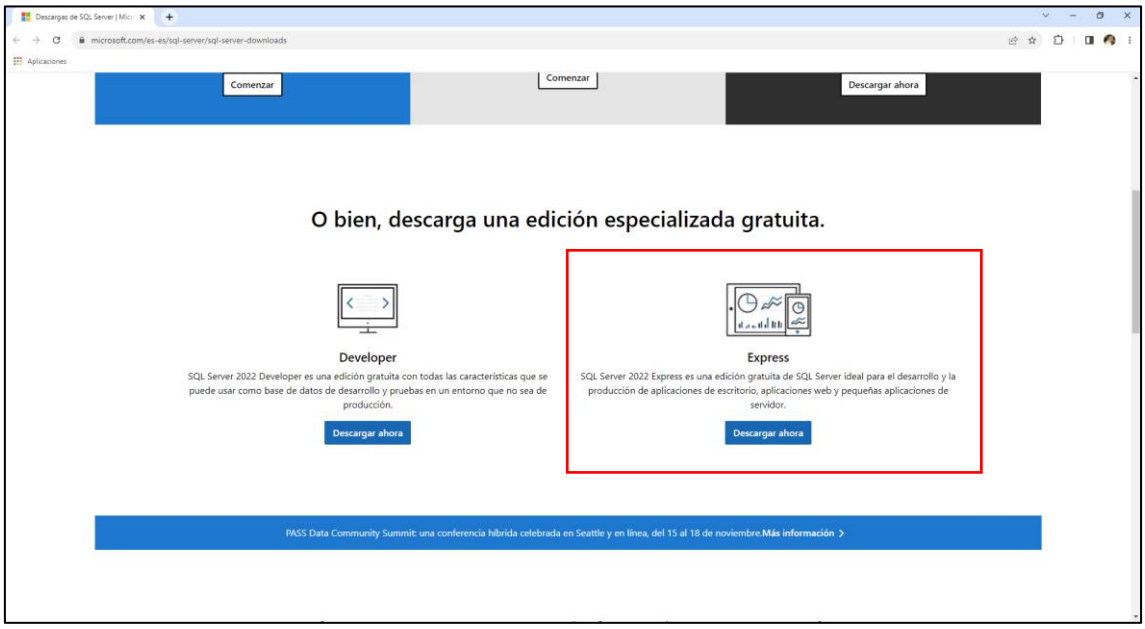
Capítulo 139.- Insertar filas en una tabla empleando solo la clase SqlConnetion y SqlCommand

SqlConection: Esta clase permite configurar la conexión a una base de datos de un servidor de base de datos SQL Server.

SqlCommand: Nos permite configurar un comando SQL (insert, delete, update, select, etc.) y su posterior comunicación con el servidor SQL Server configurando previamente mediante un objeto de la clase SqlConnection.

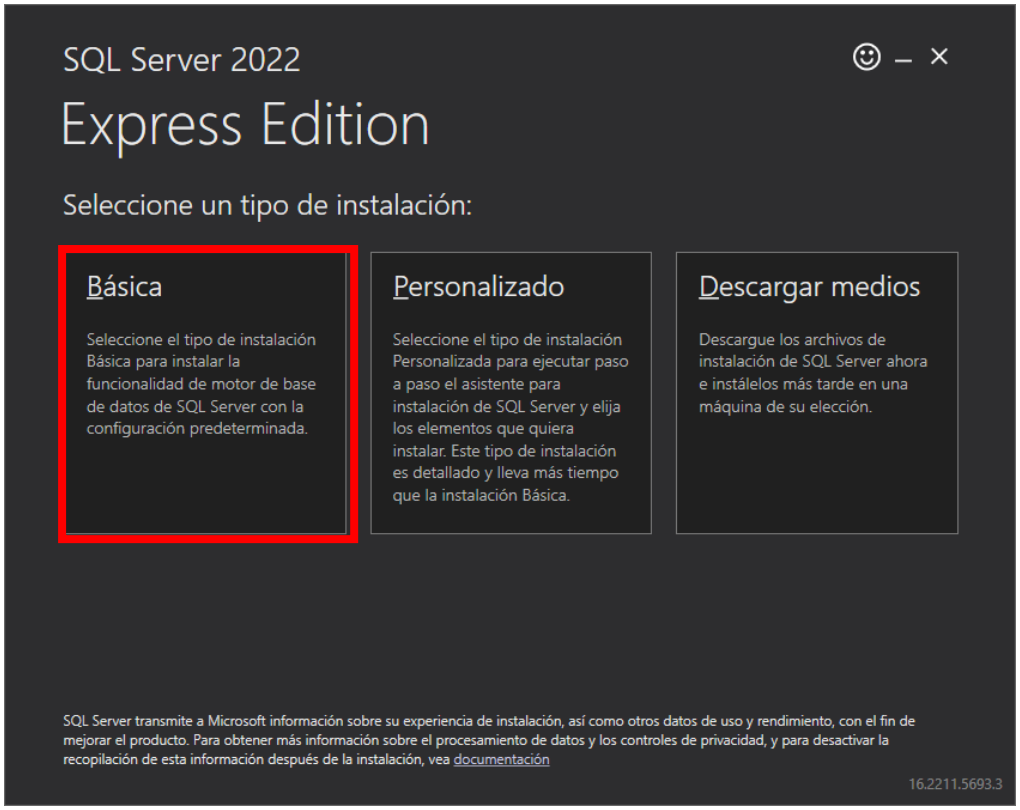
Desde el siguiente enlace vamos a descargar la aplicación:

https://www.microsoft.com/es-es/sql-server/sql-server-downloads

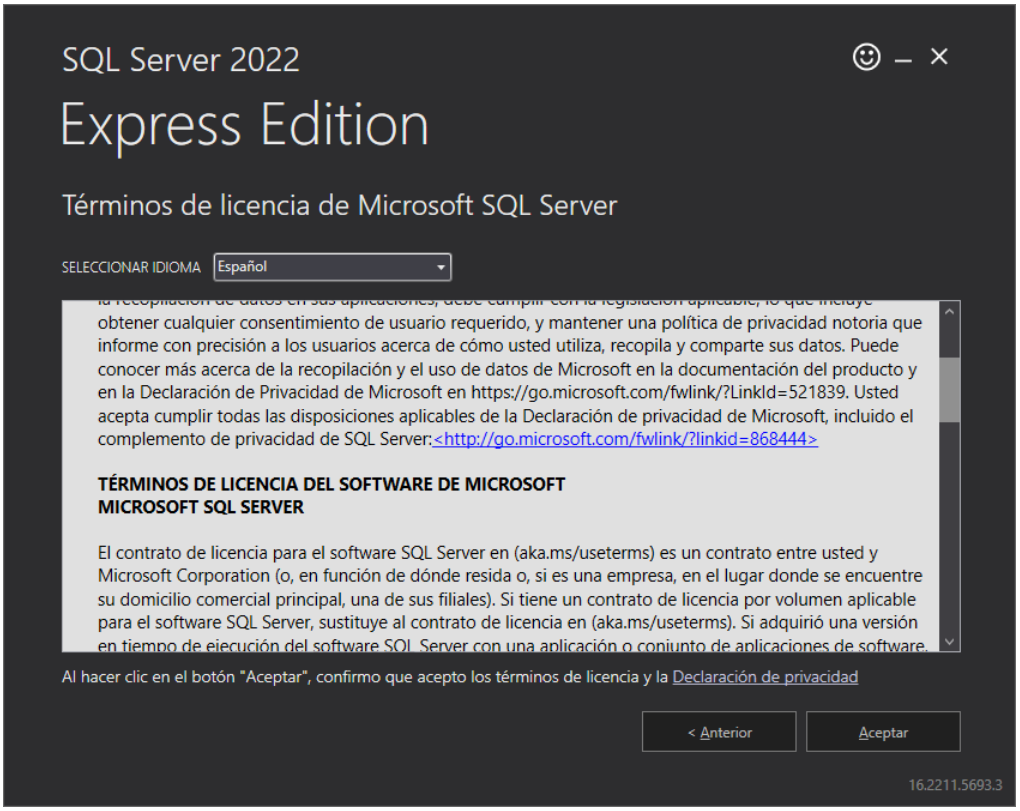


Descargamos la opción Express.

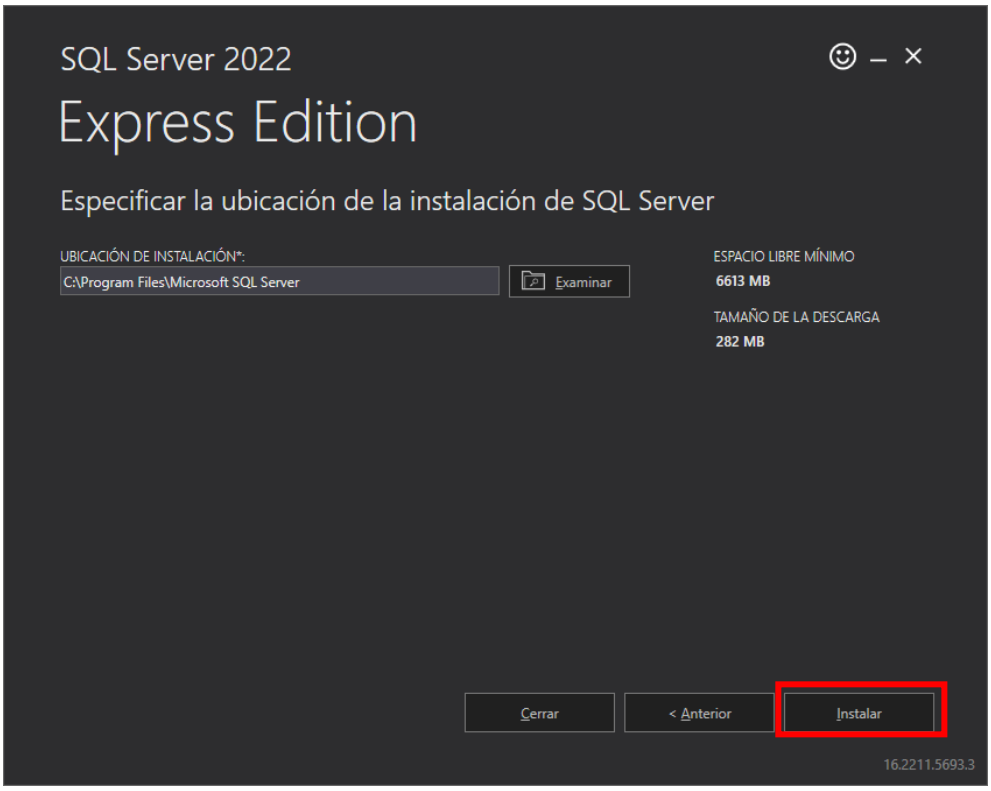
En la carpeta Descargas encontraremos el archivo de instalación, haremos doble click.



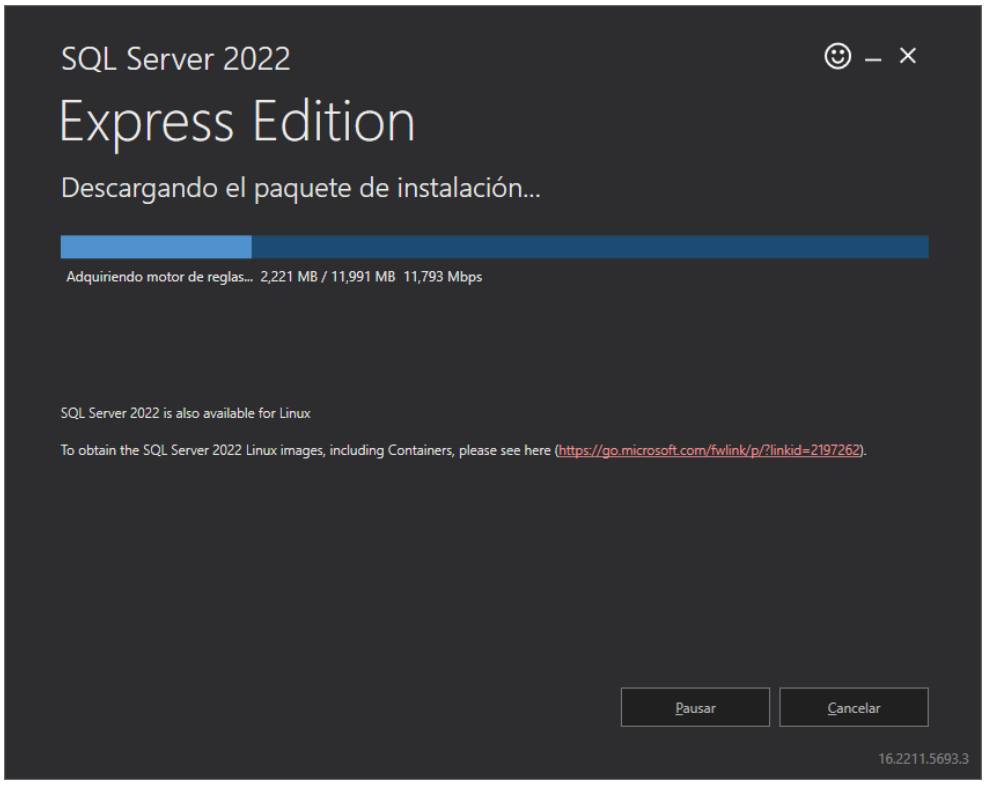
De las tres pociones seleccionaremos la Básica.



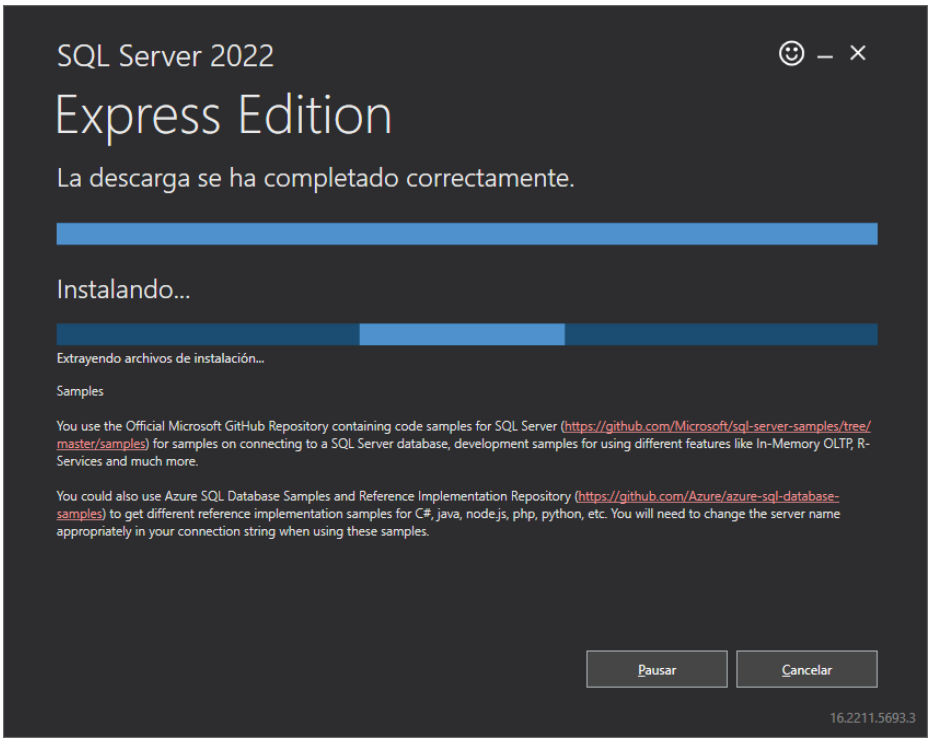
Le daremos a aceptar.



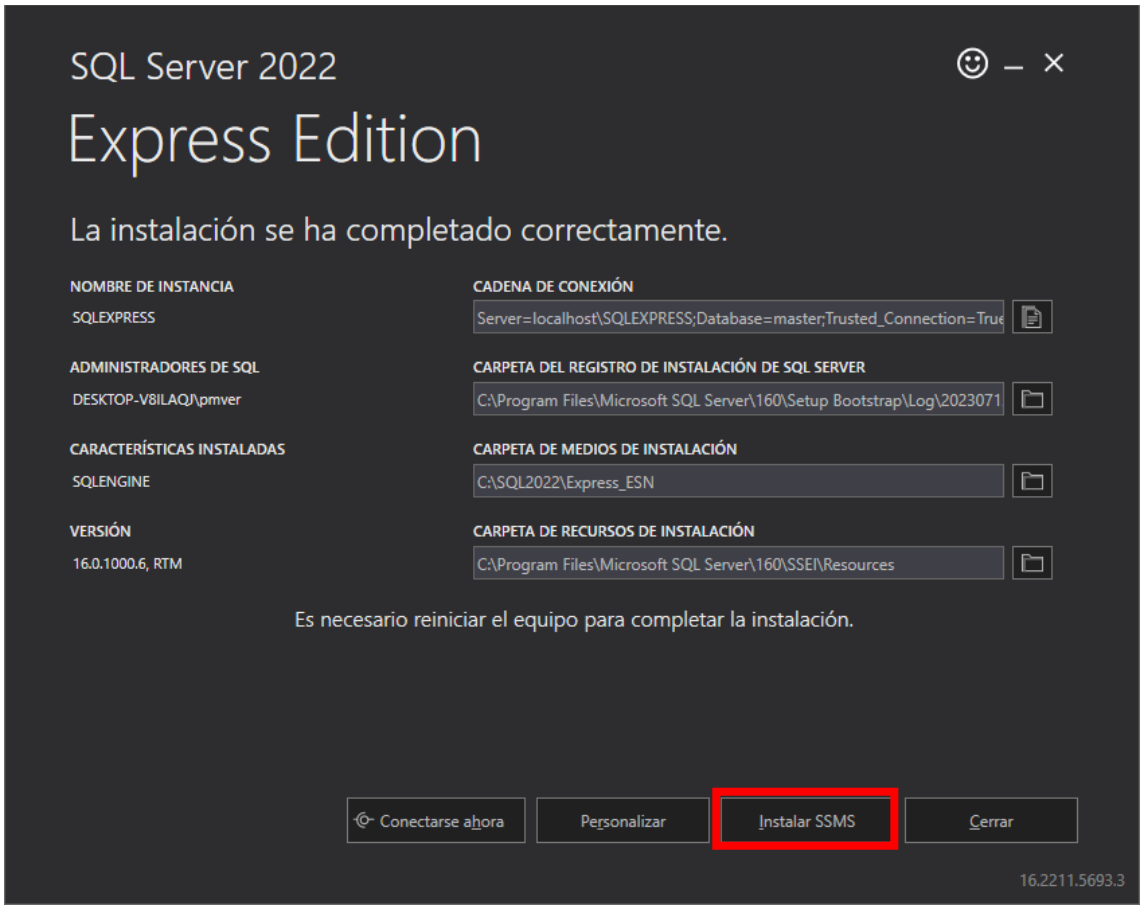
Dejaremos la ubicación por defecto y seleccionaremos el botón instalar.



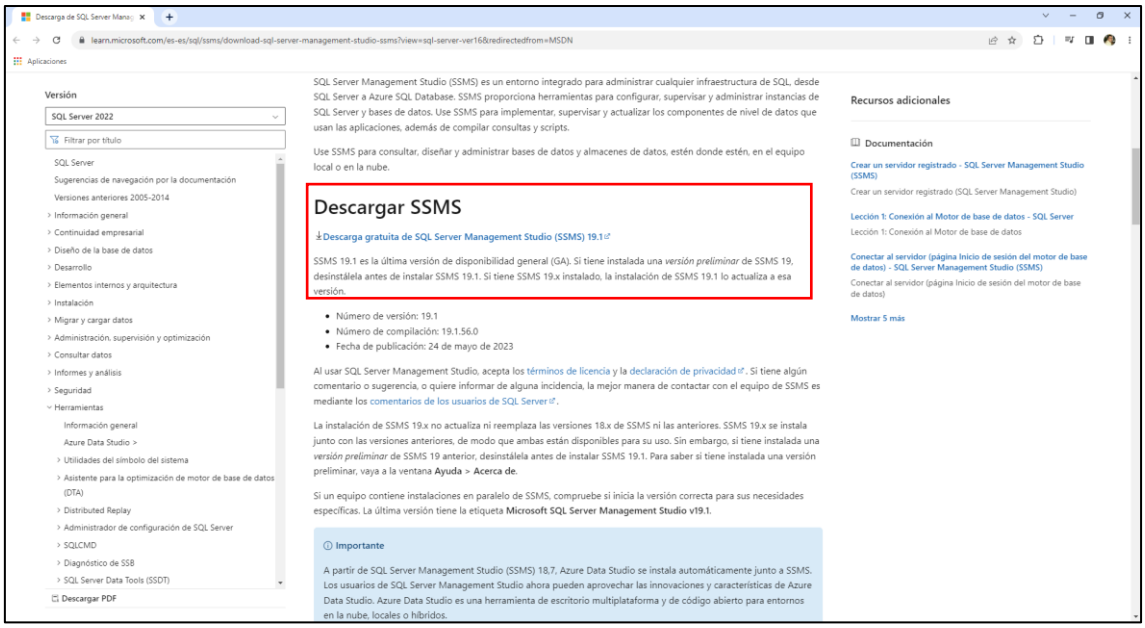
Esperaremos que finalice este proceso.



Empieza la instalación.



Tenemos que instalar la interfaz gráfica SSMS.

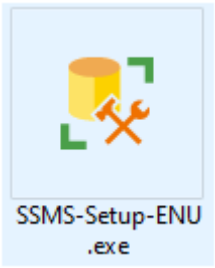


Descargar SSMS

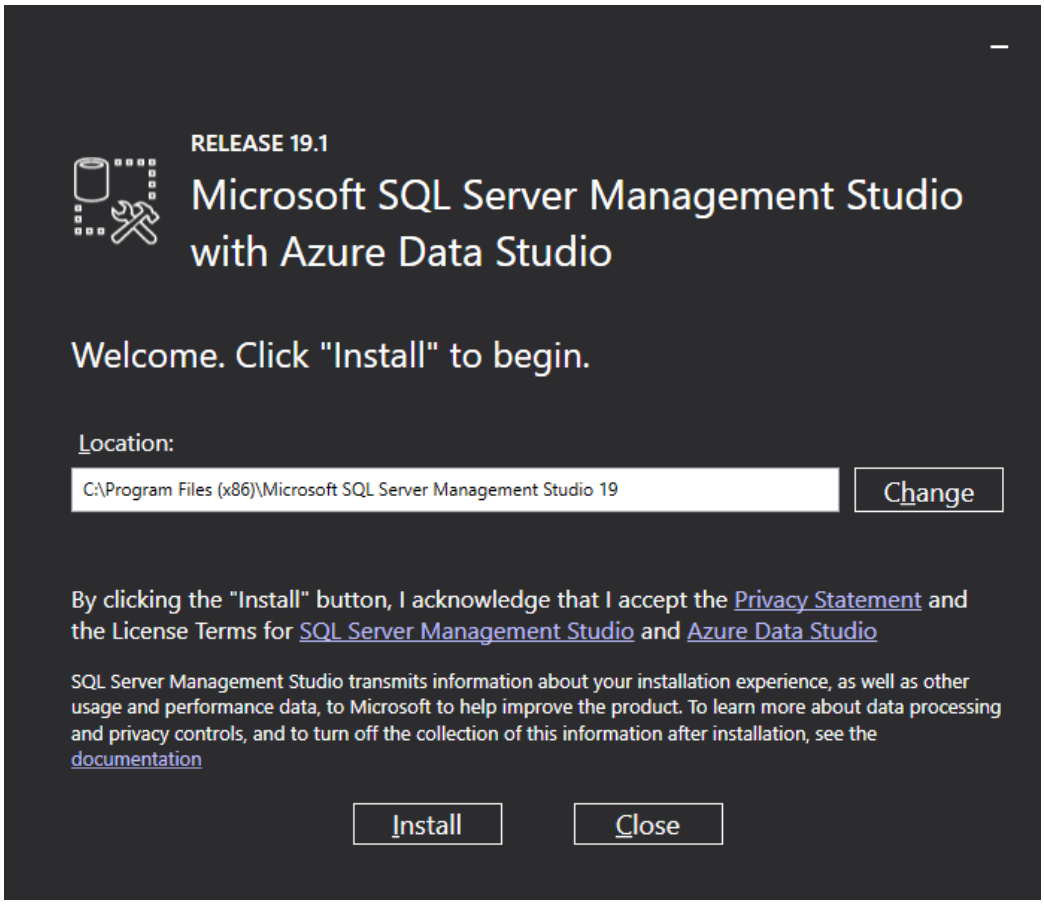
Descarga gratuita de SQL Server Management Studio (SSMS) 19.1

SSMS 19.1 es la última versión de disponibilidad general (GA). Si tiene instalada una *versión preliminar* de SSMS 19, desinstálela antes de instalar SSMS 19.1. Si tiene SSMS 19.x instalado, la instalación de SSMS 19.1 lo actualiza a esa versión.

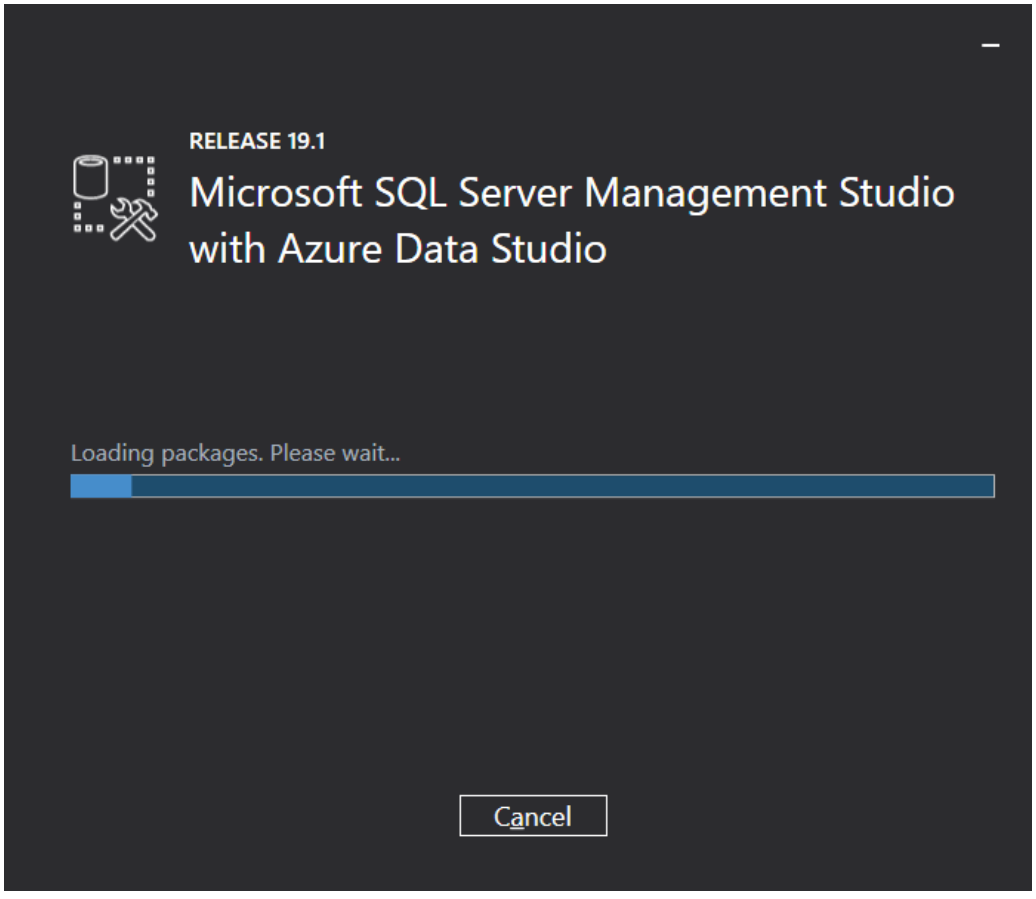
Accederemos a la descarga.

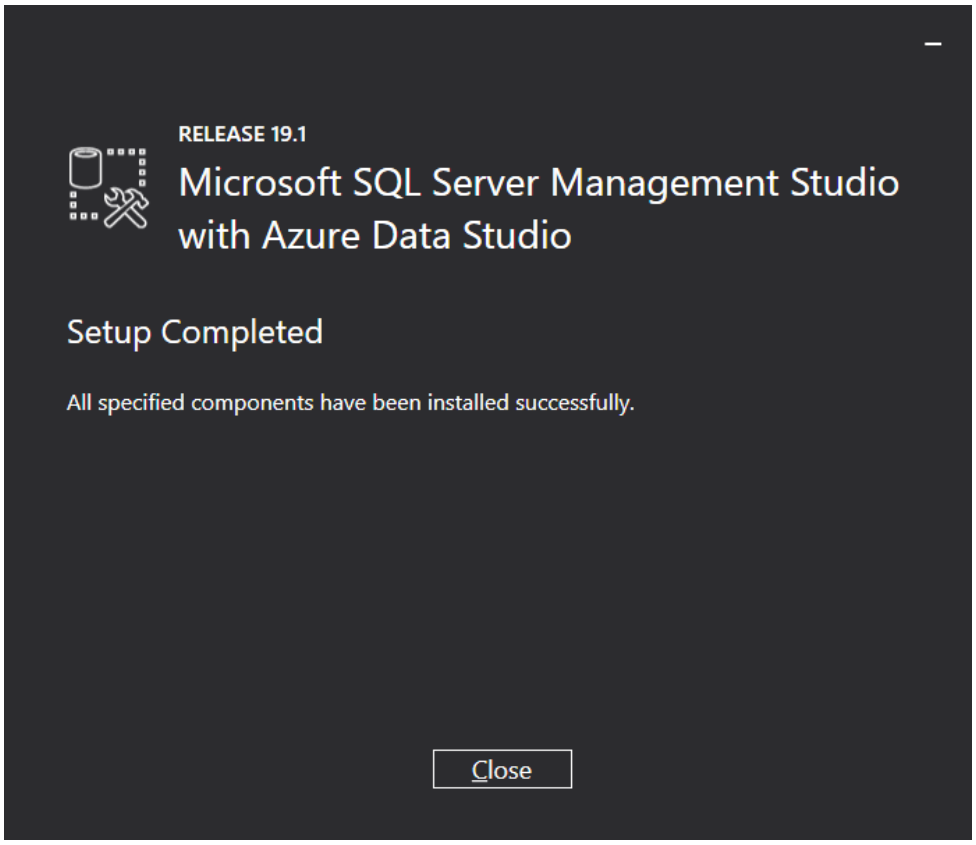


Haremos doble click sobre el instalador.

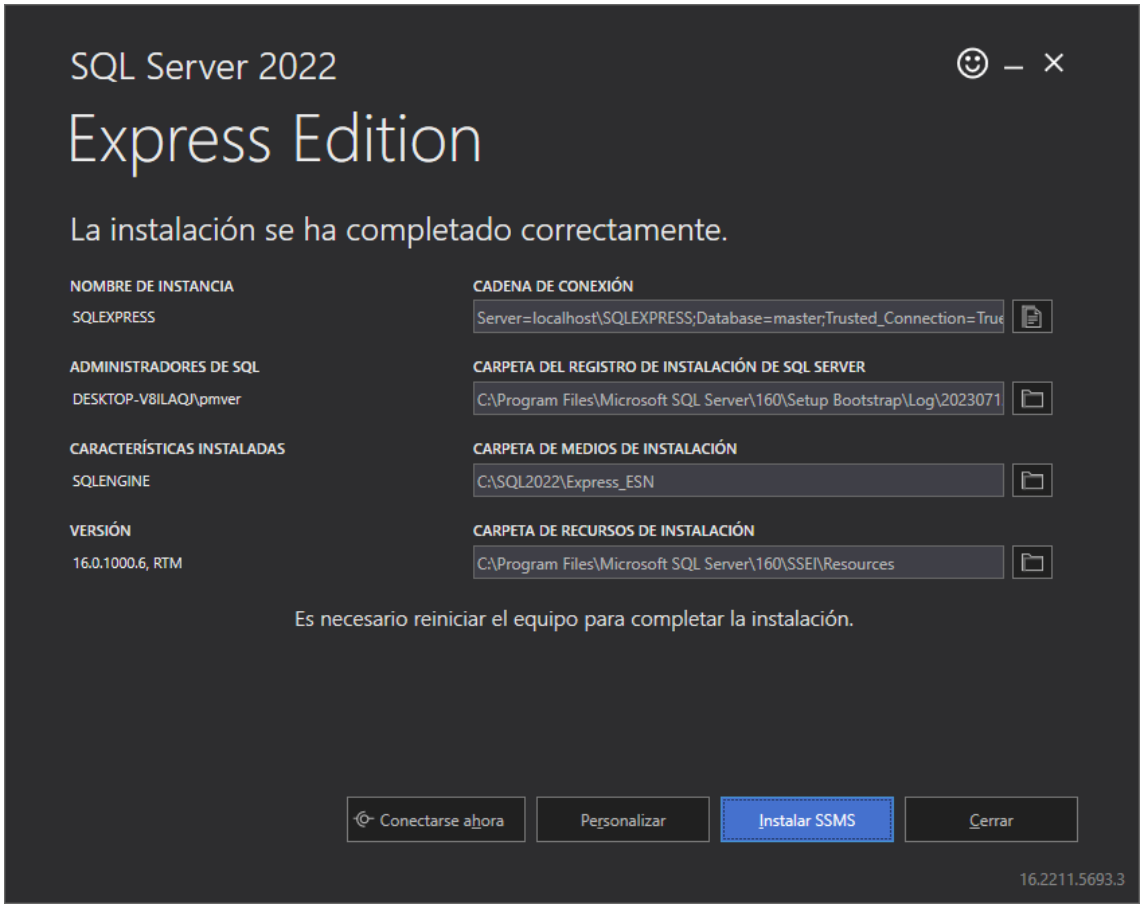


Le damos al botón Install.

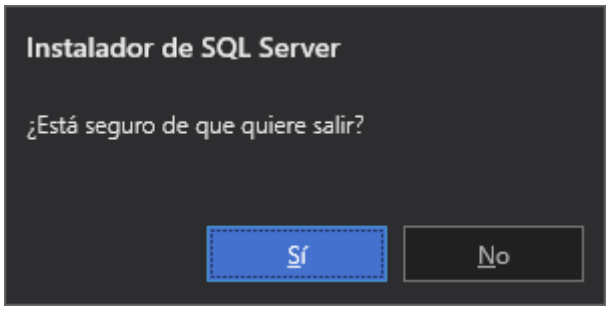




Cuando la instalación finalice le damos al botón Close.

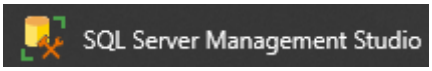


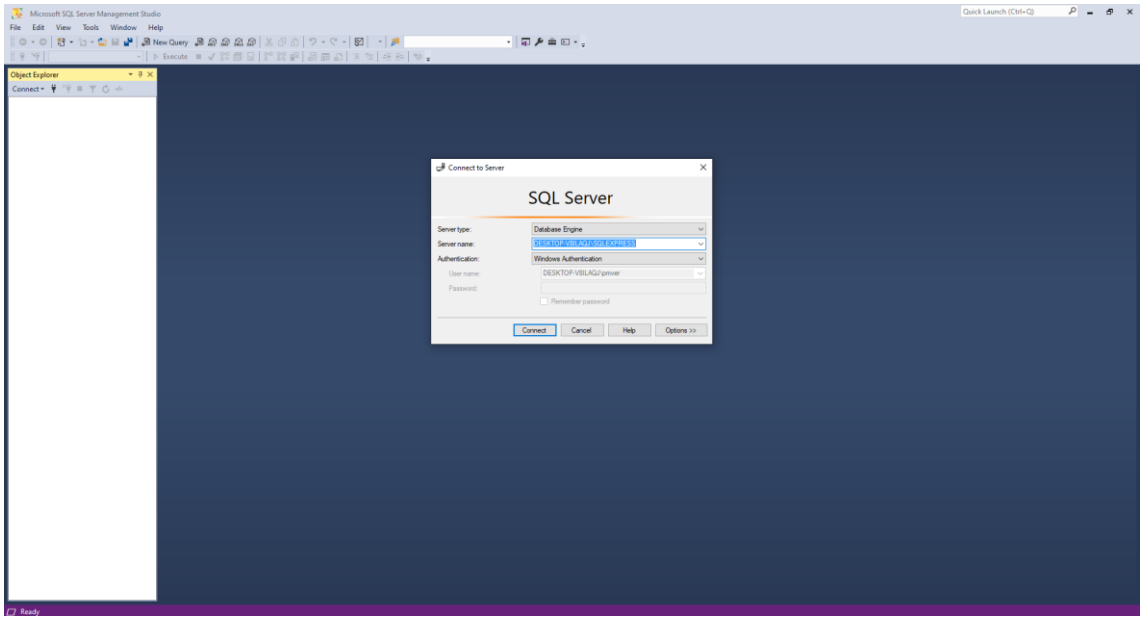
Regresamos a la instalación del SQL Express Edition y seleccionamos el botón Cerrar.



Le diremos que Sí.

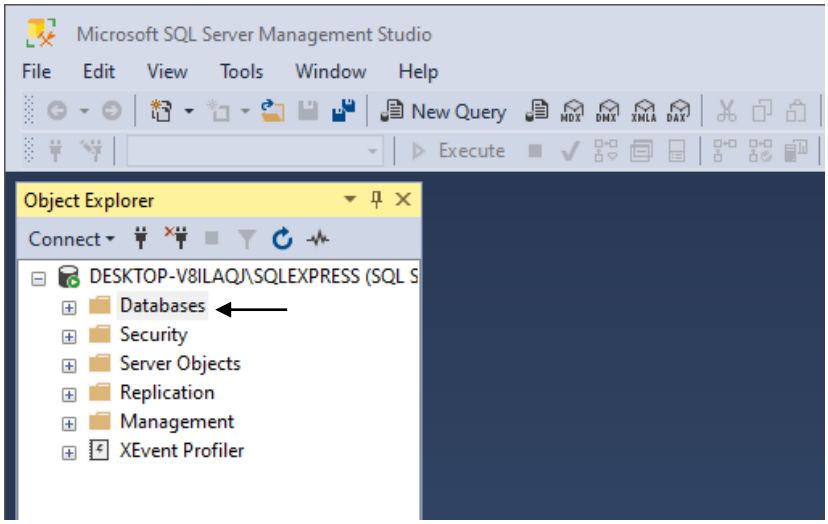
Ahora vamos a ejecutar:



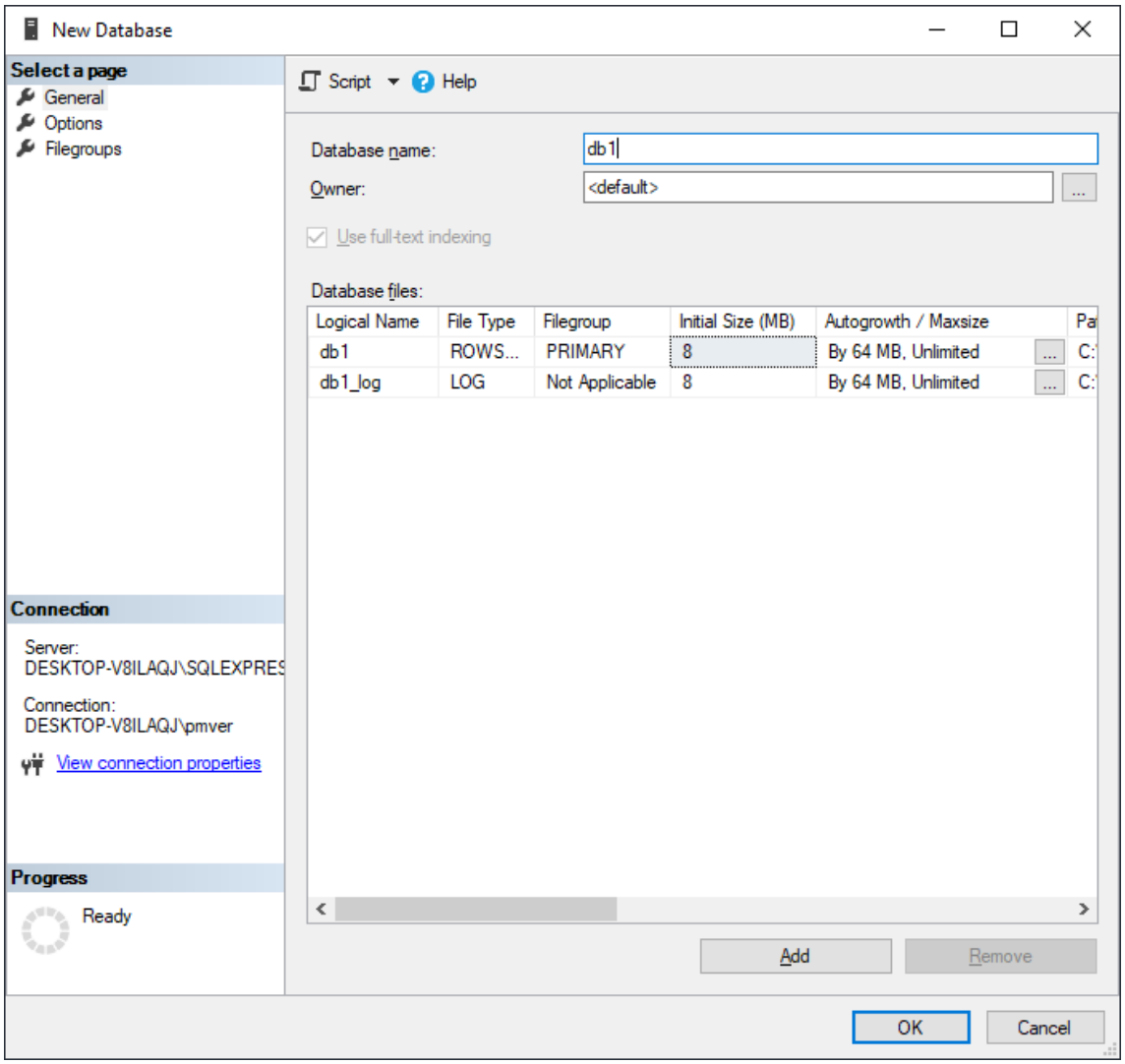


Si la aplicación se ejecuta correctamente quiere decir que la instalación ha ido bien.

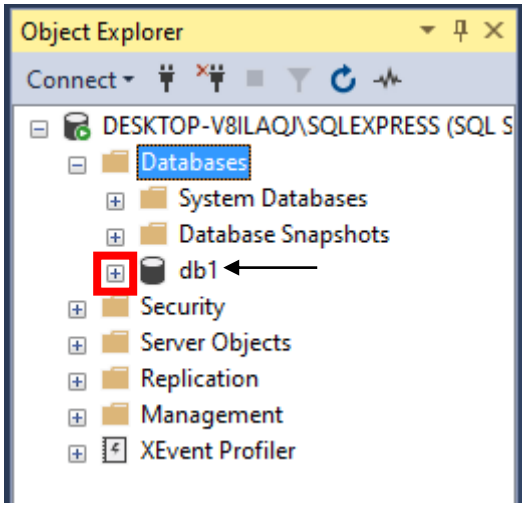
Después de la instalación vamos a seguir con el curso.



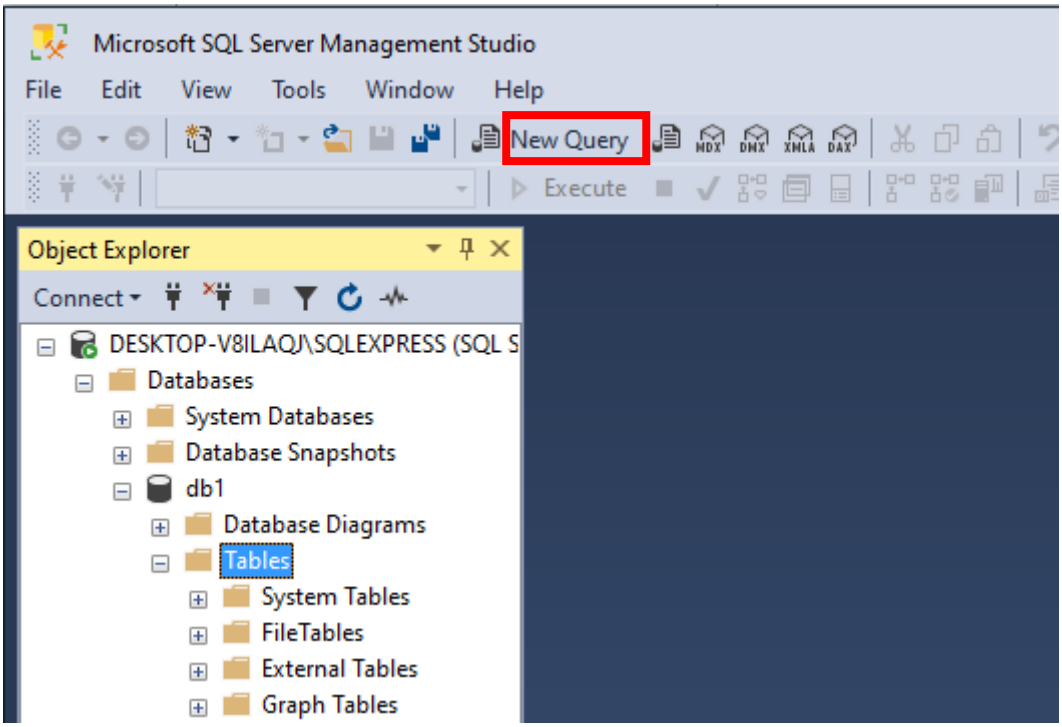
Botón derecho sobre Database y seleccionaremos New Database.



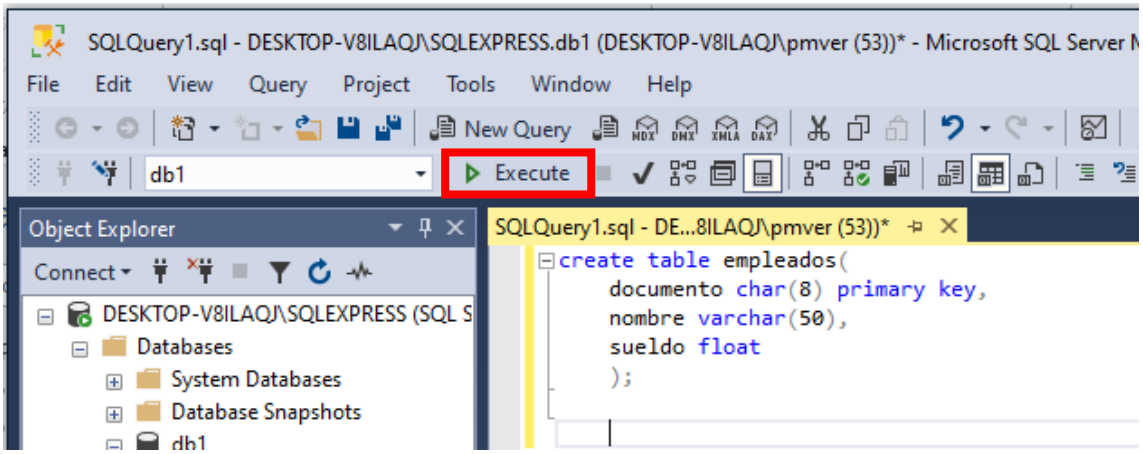
La llamaremos db1 seguido del botón OK.



Ya hemos creado la base de datos ahora vamos a selección el botón + de expandir.

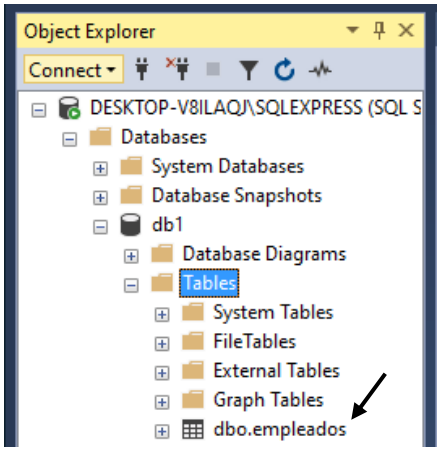


A continuación seleccionaremos el botón New Query.

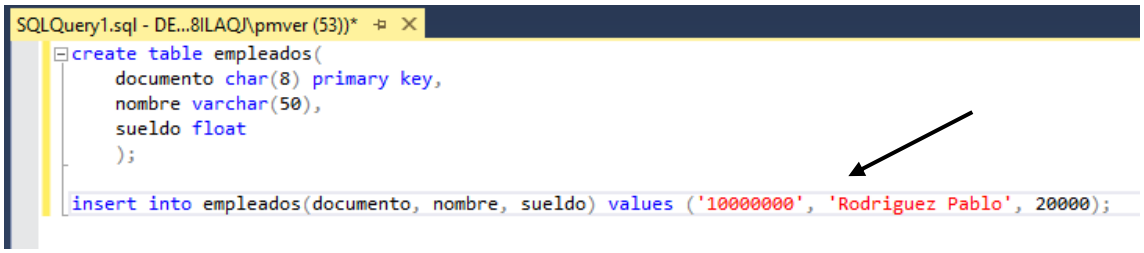


Seleccionamos todo el texto y le damos al botón Execute.

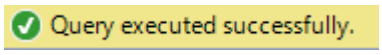
Si seleccionamos la tabla con el botón derecho del mouse y hacemos un Refresh.



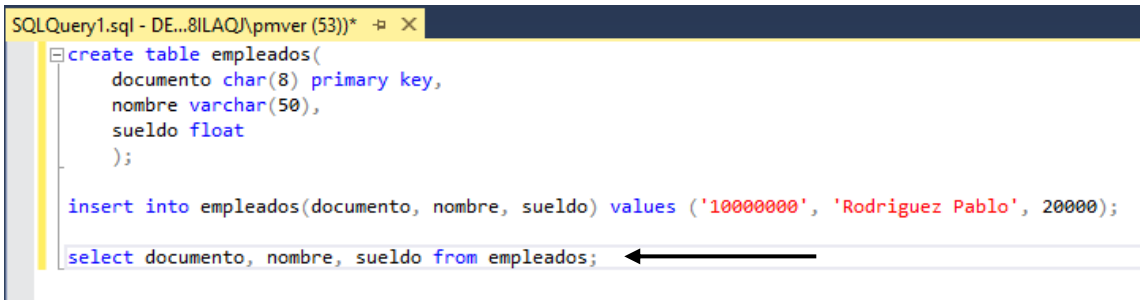
Ahora vamos a insertar una fila.



Seleccionamos el nuevo código y le damos de nuevo al botón Execute.



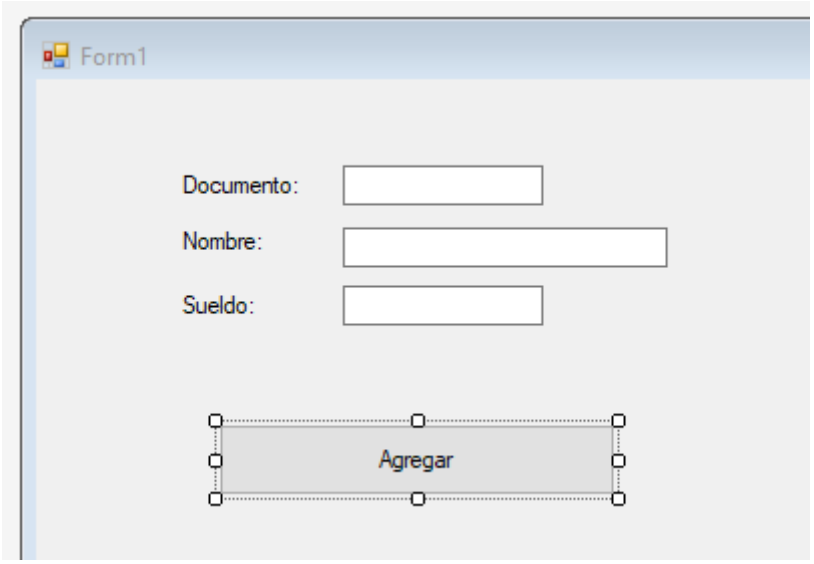
Queremos consultar por el registro que hemos añadido.



Lo seleccionamos y le damos al botón Execute.

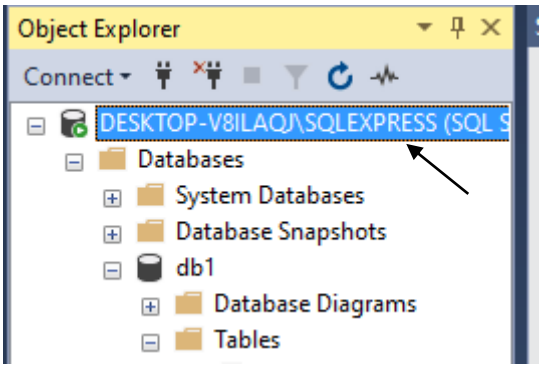
	documento	nombre	sueldo
1	10000000	Rodriguez Pablo	20000

Ahora vamos Microsoft Visual Strudio para crear un nuevo proyecto.

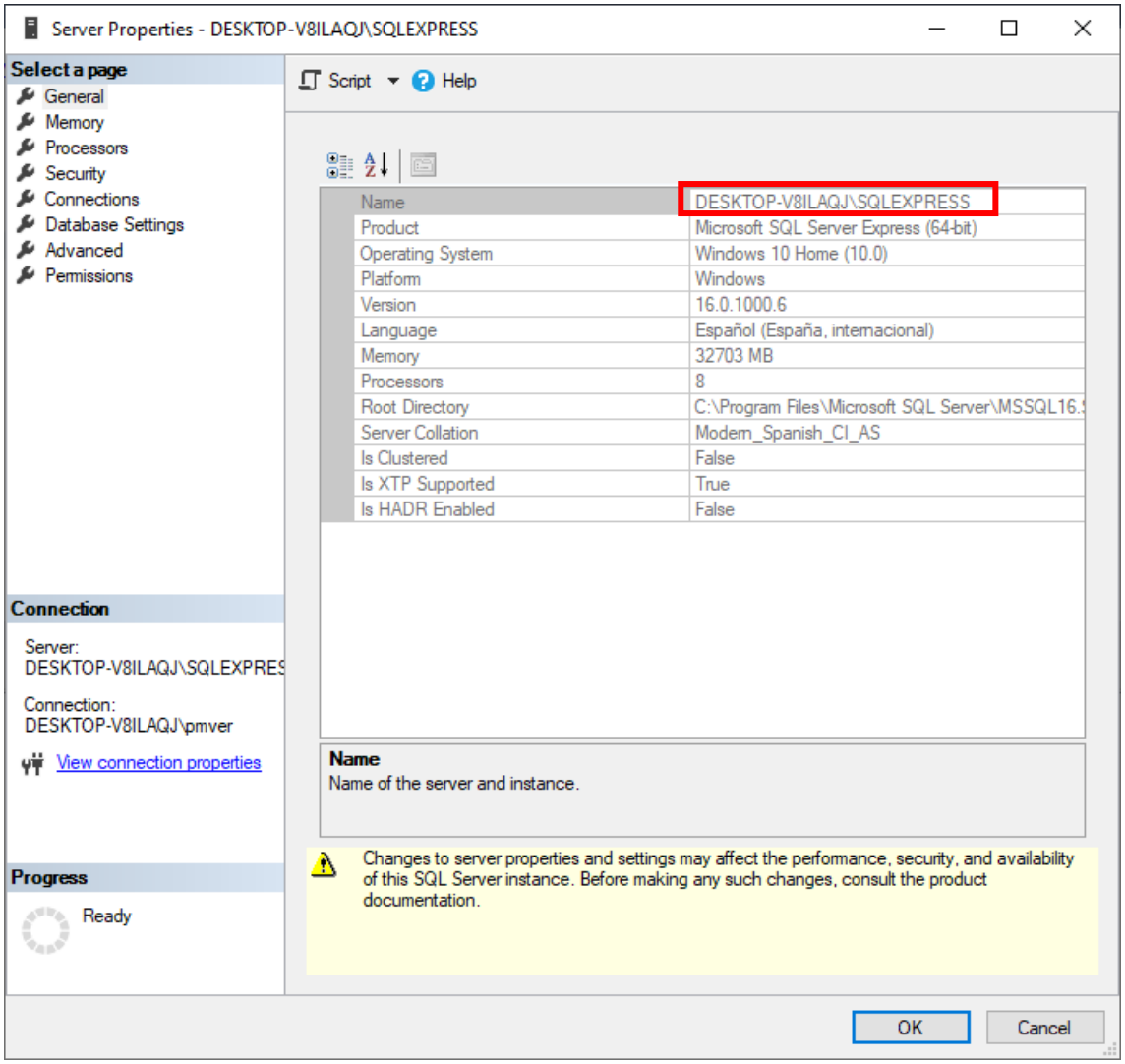


Agregamos 3 label 3 textBox y 1 button, haremos doble click sobre el botón para añadir el siguiente código:

Vamos al programa SQL



Botón derecho sobre la zona que se muestra seleccionada y del menú seleccionaremos propiedades.



Tendrás que poner el nombre que pone en el apartado Name.

El nombre que le hemos asignado a Data Source lo hemos obtenido de la siguiente forma:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto153
{
    public partial class Form1 : Form
    {
        private SqlConnection conexion = new SqlConnection("Data Source=DESKTOP-
V8ILAQJ\\SQLEXPRESS; Initial Catalog=db1; Integrated Security=True");

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            conexion.Open();
            string sql = $"insert into empleados(documento, nombre, sueldo) values
('{textBox1.Text}', '{textBox2.Text}', {textBox3.Text})";
            SqlCommand comando = new SqlCommand(sql, conexion);
            comando.ExecuteNonQuery();
            textBox1.Text = "";
            textBox2.Text = "";
            textBox3.Text = "";
            MessageBox.Show("Los datos del empleado fue cargado...");
            conexion.Close();
        }
    }
}
```

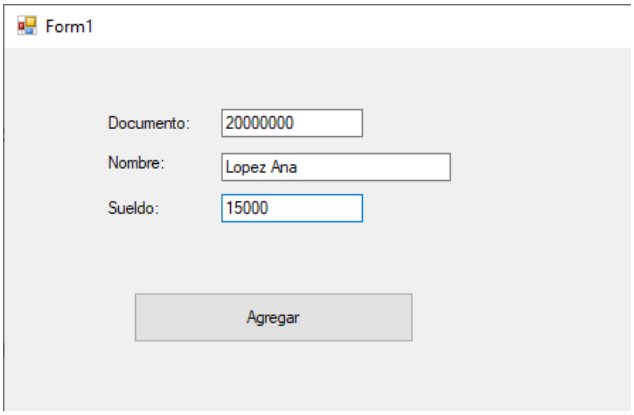
Nombre de la base de datos.

Se ha agregado una barra \ de más al tratarse de un carácter especial.

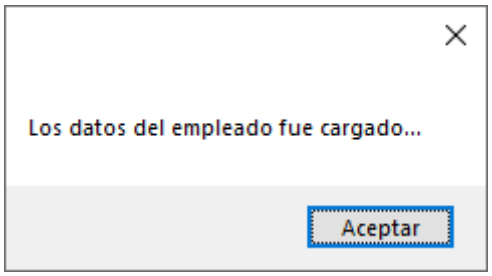
```
string sql = $"insert into empleados(documento, nombre, sueldo) values
('{textBox1.Text}', '{textBox2.Text}', {textBox3.Text})";
```

Se utiliza el símbolo $ para pasar los valores que tienen almacenados los textBox.

Vamos a ejecutar:



Le damos al botón Agregar.



Nos muestra el mensaje de que se ha agregado un registro, para verificar si se ha realizado correctamente nos vamos al programa SQL Server y consultamos si se ha agregado el registro.

```
select documento, nombre, sueldo from empleados;
```

	documento	nombre	sueldo
1	10000000	Rodriguez Pablo	20000
2	20000000	Lopez Ana	15000

Podrás observar que se ha agregado el segundo registro.

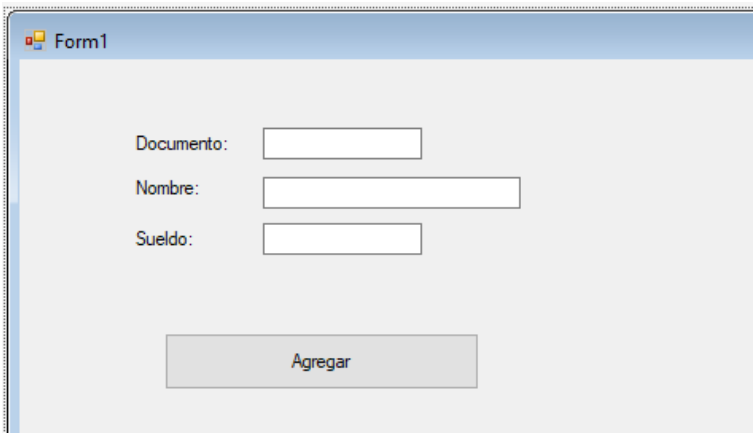
Capítulo 140.- Consultar filas empleando solo las clases SqlConnection y SqlDataReader

SqlConnection: Esta clase permite configurar la conexión a una base de datos de un servidor de base de datos de SQLK Server.

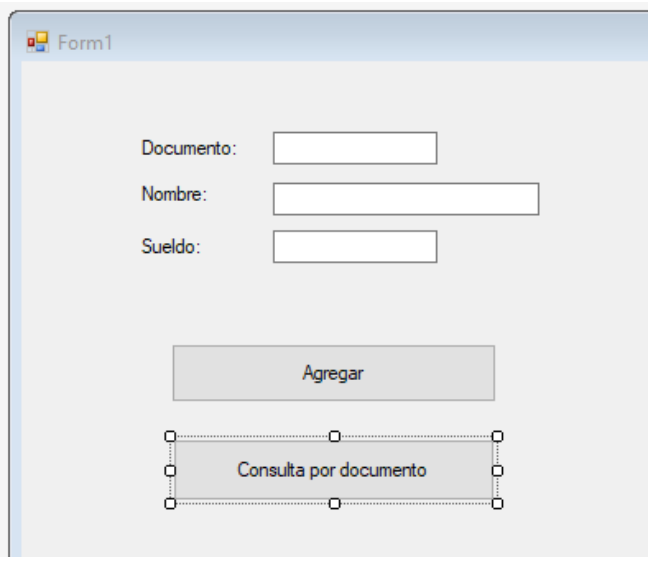
SqlCommand: Nos permite configurar el comando SQL (insert, delete, update, select, etc.) y su posterior comunicación con el servidor de SQL Server configurando previamente mediante un objeto de la clase SqlConnection.

SqlDataReader: Almacena el resultado devuelto por un comando 'select' que se ejecuta en el servidor.

Seguimos con el proyecto del capítulo anterior.



Vamos a agregar un segundo botón "Consulta por documentos.



Vamos a ejecutar el programa SQL para consultar por documento.

```
SQLQuery1.sql - DE...8ILAQJ\pmver (55))*
use db1;

select * from empleados;
```

Este será el resultado:

	documento	nombre	sueldo
1	10000000	Rodriguez Pablo	20000
2	20000000	Lopez Ana	15000
3	30000000	Martinez Pablo	18000

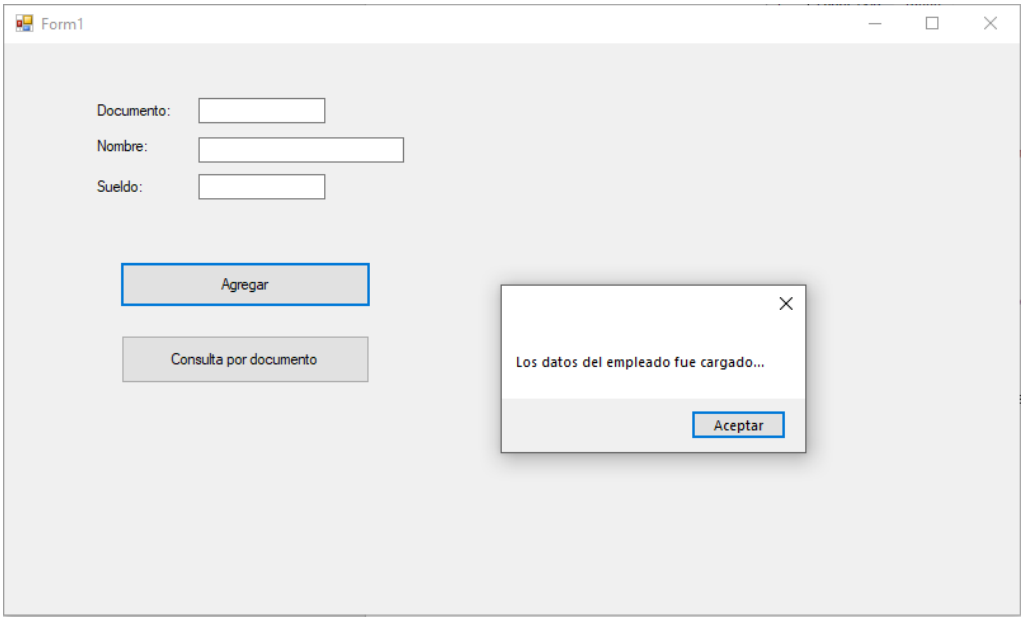
```
select nombre, sueldo from empleados where documento='20000000';
```

	nombre	sueldo
1	Lopez Ana	15000

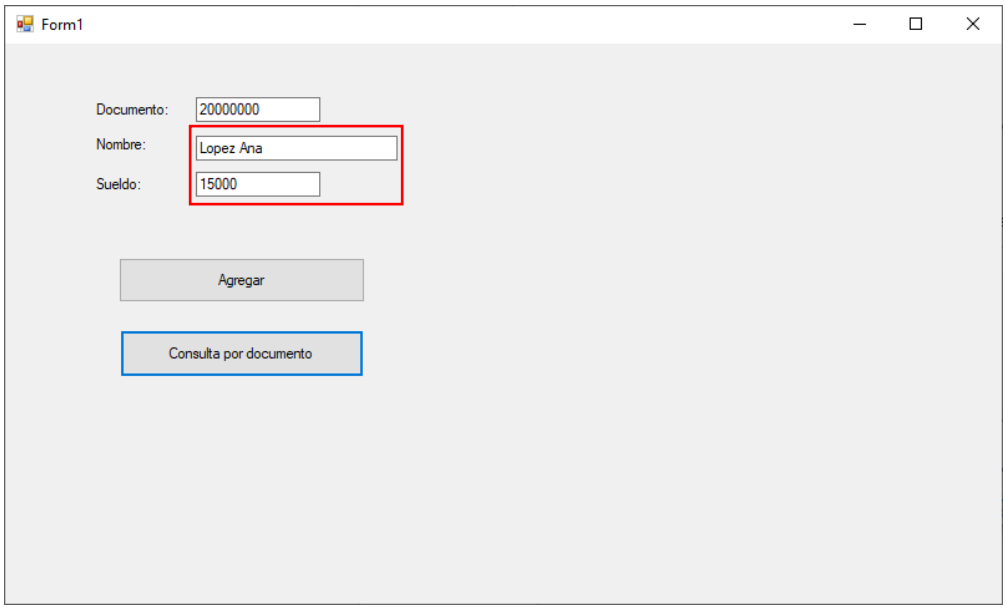
Vamos a programar el botón de nuestro proyecto.

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = $"select nombre, sueldo from empleados where documento='{textBox1.Text}'; ";
    SqlCommand comando = new SqlCommand(sql, conexion);
    SqlDataReader registro = comando.ExecuteReader();
    if (registro.Read())
    {
        textBox2.Text = registro["nombre"].ToString();
        textBox3.Text = registro["sueldo"].ToString();
    }
    else
        MessageBox.Show("No existe un empleado con dicho documento");
    registro.Close();
    conexion.Close();
}
```

Vamos a ejecutar e introducir un numero de documento que no existe.



Ahora vamos a introducir un número de documento que si existe.



Nos muestra la información.

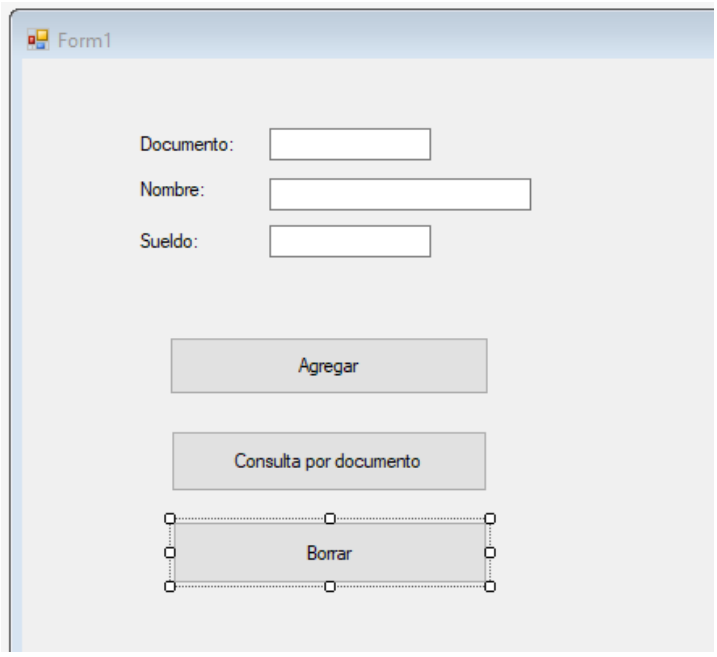
Capítulo 141.- Borrado de filas de una tabla empleando solo las clases SqlConnection y SqlCommand.

SqlConnection: Esta clase permite configurar la conexión a una base de datos de un servidor de base de datos SQL Server.

SqlCommand: Nos permite configurar un comando SQL (insert, delete, update, select, etc.) y su posterior comunicación con el servidor de SQL Server configurado previamente mediante un objeto de la clase SqlConnection.

Seguimos trabajando con el ejercicio de capítulo anterior.

Vamos a agregar un botón para Borrar.



Vamos a programa SQL para ver como se hace el borrado.

```
use db1;
select * from empleados;
```

Consultamos por todos los registros.

	documento	nombre	sueldo
1	10000000	Rodriguez Pablo	20000
2	11000000	Rodriguez Pablo	20000
3	20000000	Lopez Ana	15000
4	30000000	Martinez Pablo	18000

```
delete from empleados where documento='11000000';
```

Eliminamos el registro cuyo documento es '11000000'.

```
select * from empleados;
```

Consultamos de nuevo los registros:

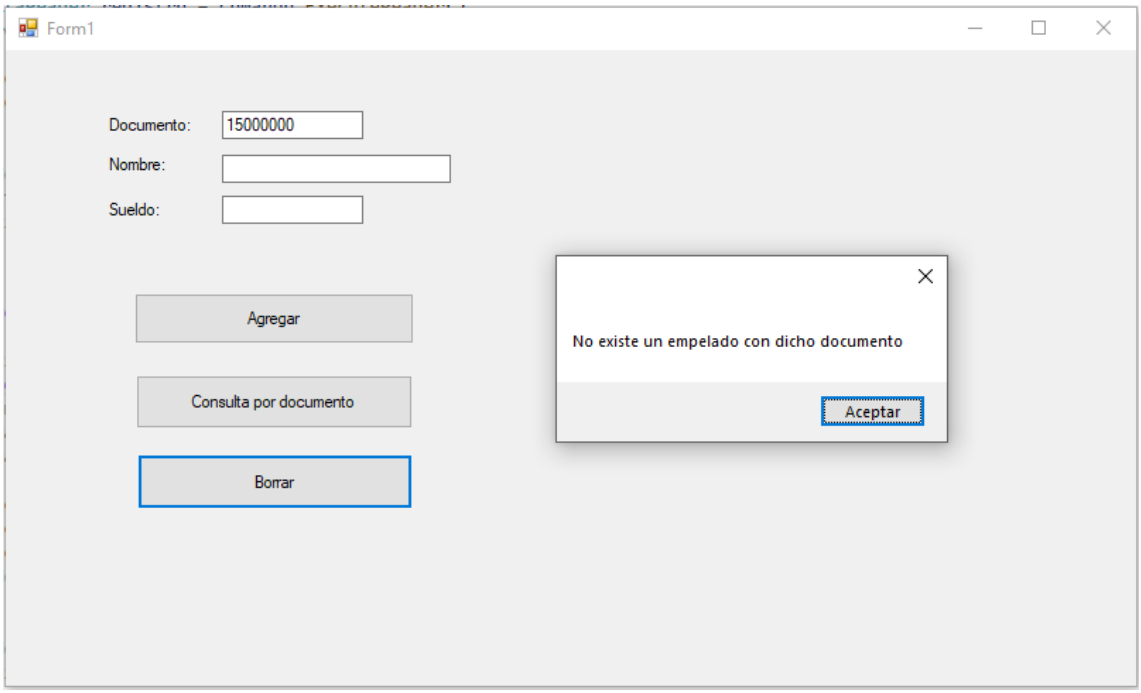
	documento	nombre	sueldo
1	10000000	Rodriguez Pablo	20000
2	20000000	Lopez Ana	15000
3	30000000	Martinez Pablo	18000

El registro cuyo numero de documento es '11000000' ya se ha eliminado.

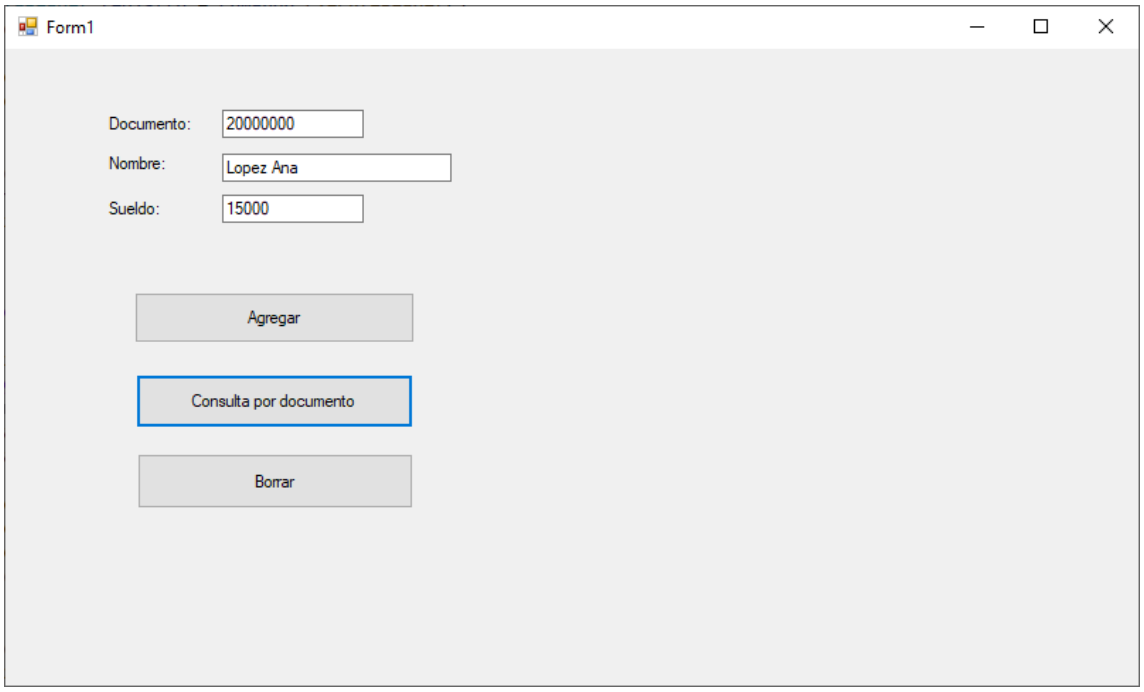
Vamos a programa el botón Borrar en el evento click.

```
1 referencia
private void button3_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = $"delete from empleados where documento='{textBox1.Text}';";
    SqlCommand comando = new SqlCommand(sql, conexion);
    int cant = comando.ExecuteNonQuery();
    if (cant == 1)
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        MessageBox.Show("Se ha eliminado en empleado con dicho documento");
    }
    else
        MessageBox.Show("No existe un empelado con dicho documento");
    conexion.Close();
}
```

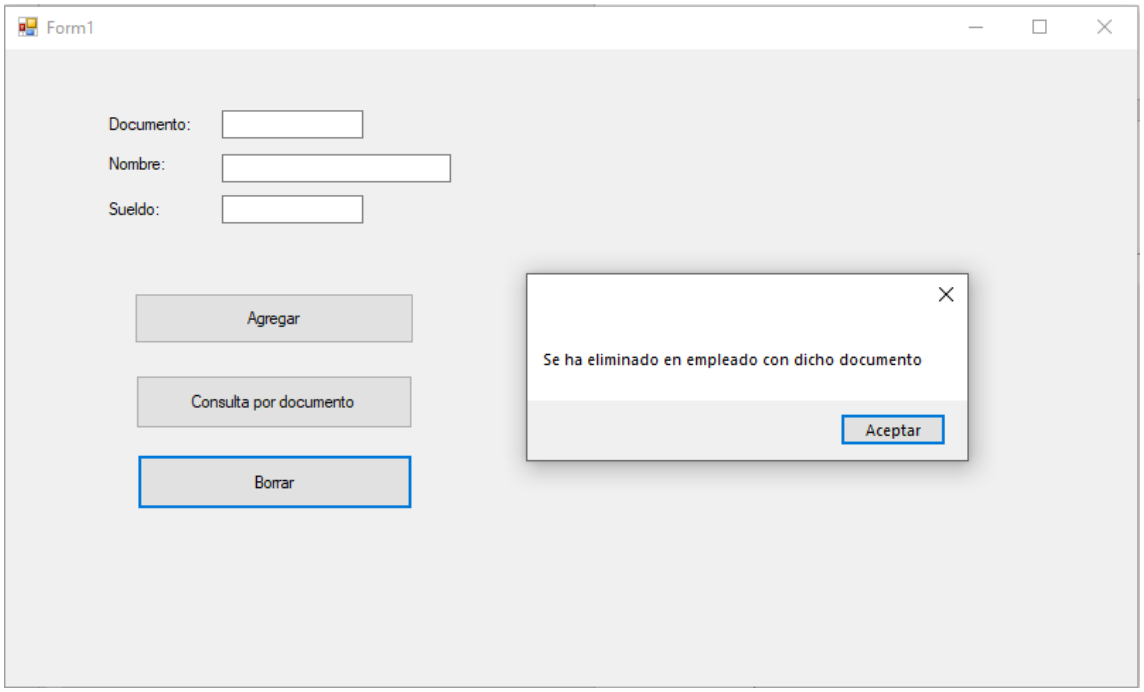
Queremos borrar un registro con un número de documento que no existe por ejemplo el 15000000.



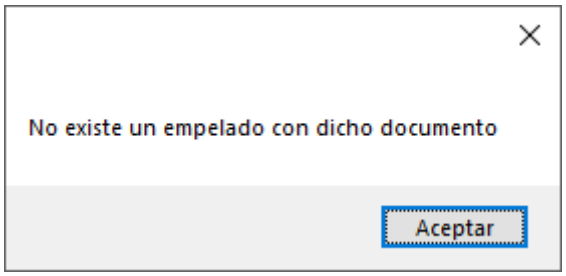
Vamos a consultar por el empleado con documento número 20000000, una vez nos muestre sus datos lo vamos a borrar.



Este registro existe ahora lo vamos a borrar.



Vamos a intentar borrar de nuevo el empelado con el número de documento 20000000.



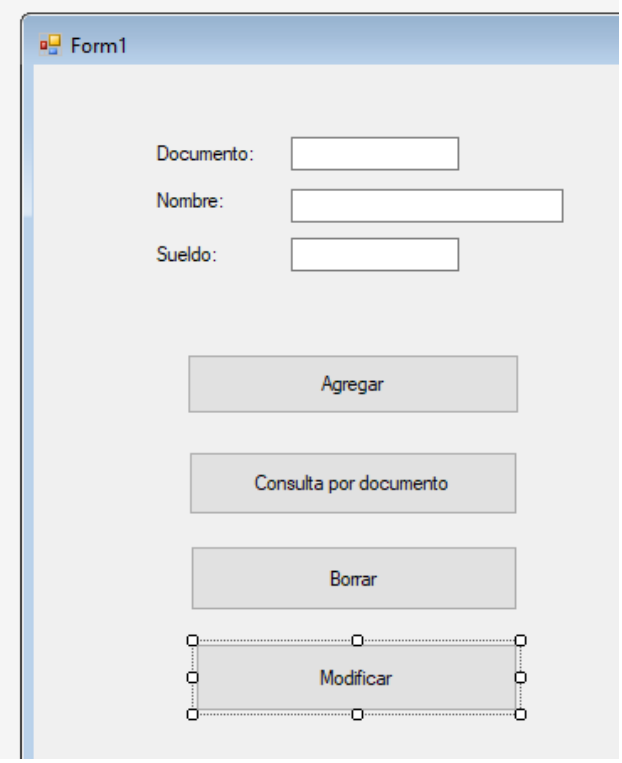
Capítulo 142.- Modificación de filas de una tabla empleando solo las clases SqlConnection y SqlCommand

SqlConnection: Esta clase permite configurar la conexión a una base de datos de un servidor de base de datos SQL Server.

SqlCommand: Nos permite configurar un comando SQL (insert, delete, update, select, etc.) y su posterior comunicación con el servidor de SQL Server configurado previamente mediante un objeto de la clase SqlConnection.

Seguimos con la práctica del capítulo anterior.

Vamos a agregar un nuevo botón para modificar.



Vamos a hace que se pueda modificar el nombre y el sueldo.

Vamos al programa SQL.

```
SQLQuery1.sql - DE...8ILAQJ\pmver (77))*
use db1;
select * from empleados;
```

Este será el resultado:

	documento	nombre	sueldo
1	10000000	Rodriguez Pablo	20000
2	30000000	Martinez Pablo	18000

Ejecutamos la siguiente actualización:

```
update empleados set nombre='Rodriguez Lopez Pablo', sueldo=18500  where documento='10000000';
```

```
select * from empleados;
```

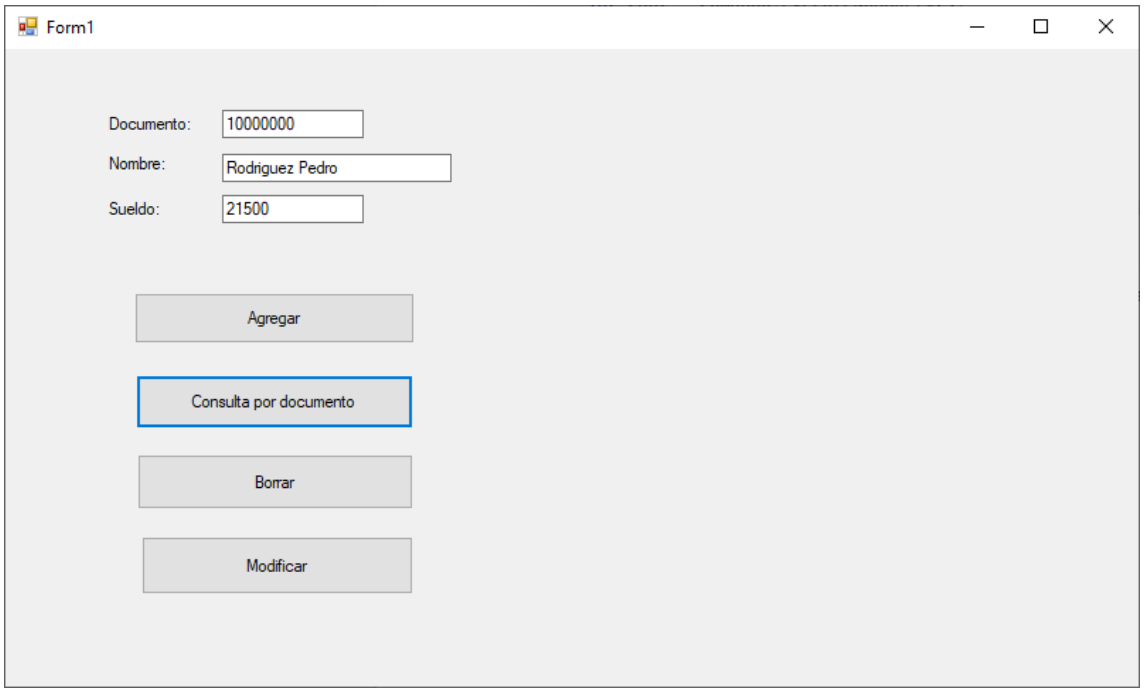
Consultamos de nuevo la tabla:

	documento	nombre	sueldo
1	10000000	Rodriguez Lopez Pablo	18500
2	30000000	Martinez Pablo	18000

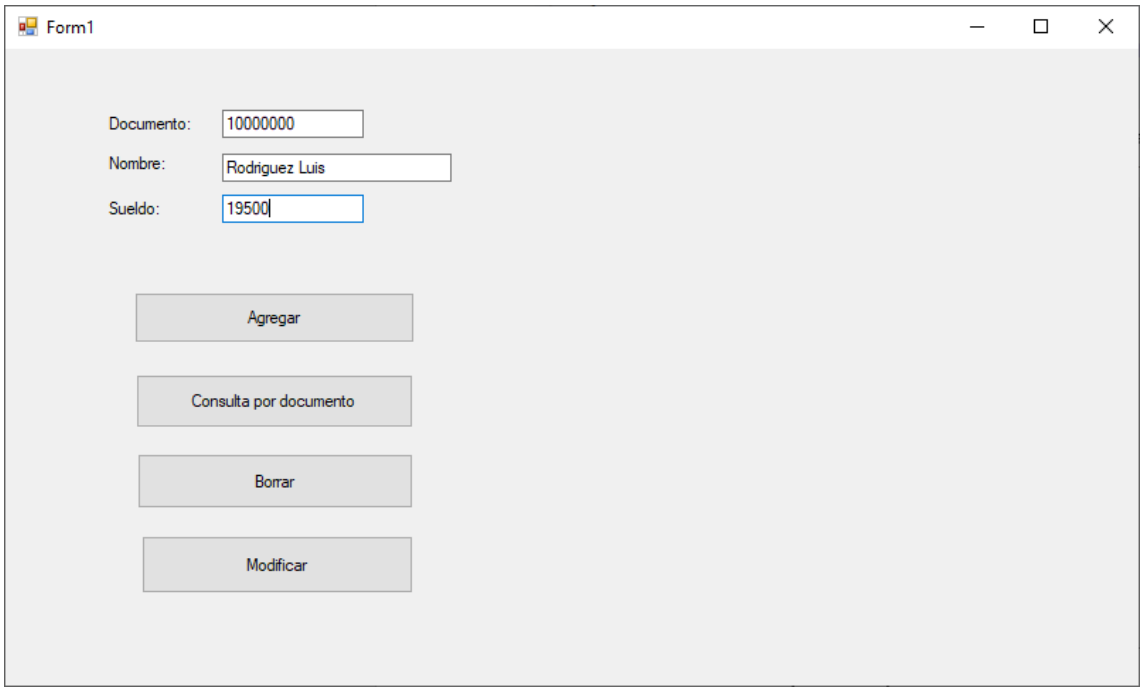
Vamos agregar el código del botón Modificar en el evento click.

```
1 referencia
private void button4_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = $"update empleados set nombre='{textBox2.Text}', " +
        $"sueldo={textBox3.Text}  where documento='{textBox1.Text}';";
    SqlCommand comando = new SqlCommand(sql, conexion);
    int cant = comando.ExecuteNonQuery();
    if (cant == 1)
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        MessageBox.Show("Se ha modificado el empleado con dicho documento");
    }
    else
        MessageBox.Show("No existe un empelado con dicho documento");
    conexion.Close();
}
```

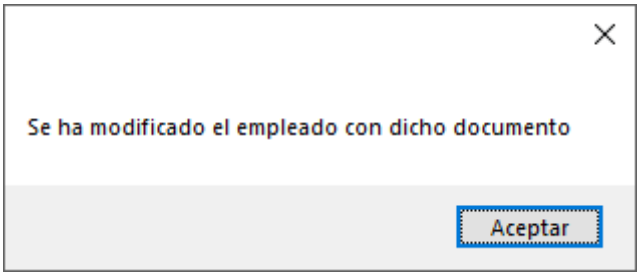
Ya puedes consultar pon un determinado número de documento realiza la modificación.



Vamos a modificar el nombre y el sueldo y le daremos a modificar.



Le damos a Modificar.



Desde el programa SQL vamos a consultar la tabla:

```
select * from empleados;
```

	documento	nombre	sueldo
1	10000000	Rodriguez Luis	19500
2	30000000	Martinez Pablo	18000

Podrás comprobar cómo se han actualizado los datos.

En estos últimos capítulos hemos creado una aplicación para acceder a una base de datos en SQL para añadir, consultar, modificar y eliminar registros.

Capítulo 143.- Mostrar filas en un control DataGridView recuperando los datos de SqlServer

SqlConnection: Esta clase permite configurar la conexión a una base de datos de un servidor de base de datos SQL Server.

SqlCommand: Nos permite configurar un comando SQL (insert, delete, update, select, etc.) y su posterior comunicación con el servidor de SQL Server configurado previamente mediante un objeto de la clase SqlConnection.

SqlDataReader: Almacena el resultado devuelto por un comando 'select' que ejecuta en el servidor.

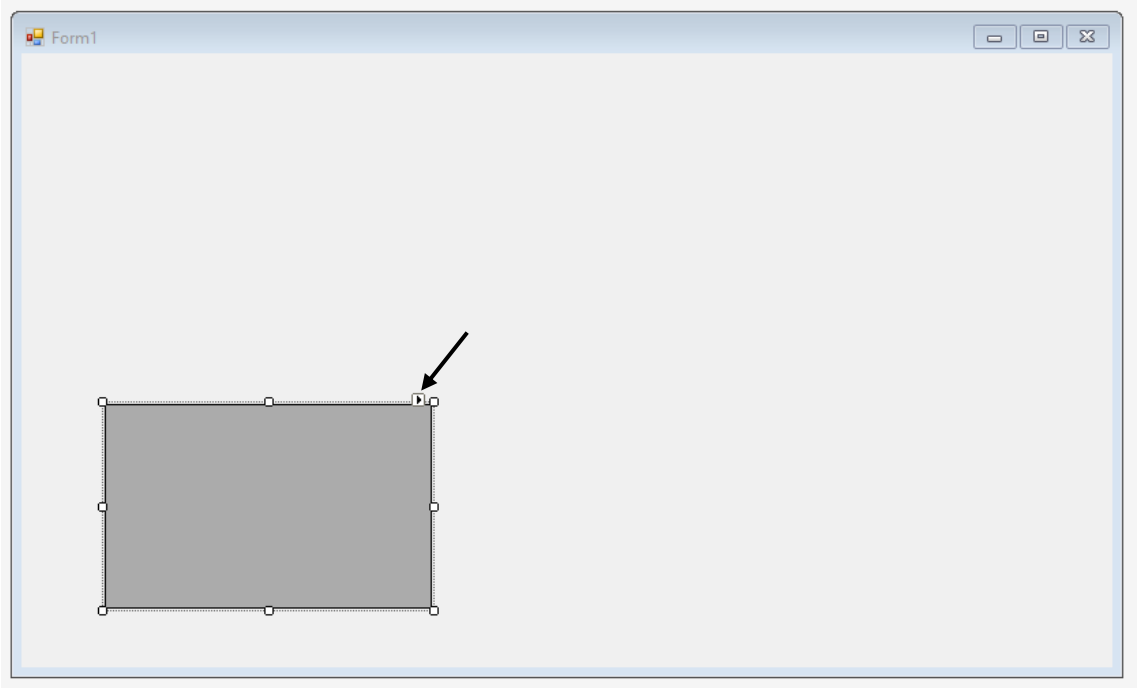
```
SQLQuery1.sql - DE...8ILAQJ\pmver (62))
  /****** Scrip for SelectTopNRows command from SSMS ******/
SELECT TOP(1000) [documento]
      ,[nombre]
      ,[sueldo]
FROM [db1].[dbo].[empleados]
```

Si ejecutamos este será el resultado:

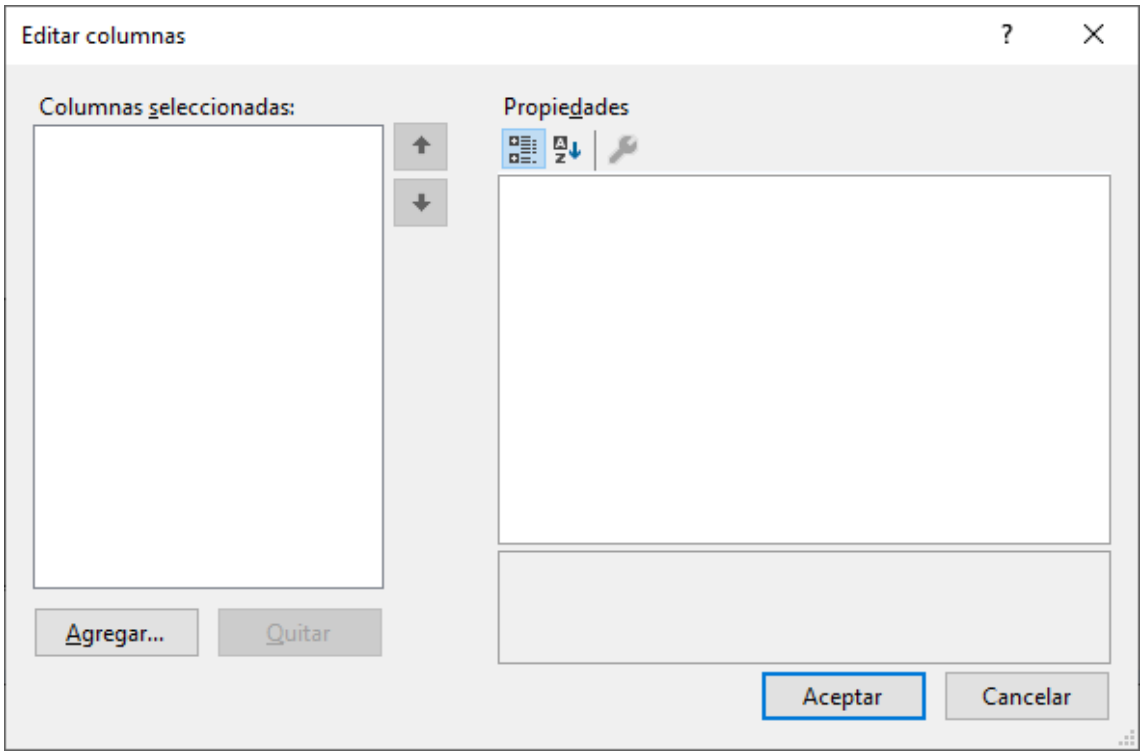
	documento	nombre	sueldo
1	10000000	Rodriguez Luis	19500
2	30000000	Martinez Pablo	18000

Vamos a crear un nuevo proyecto con C#.

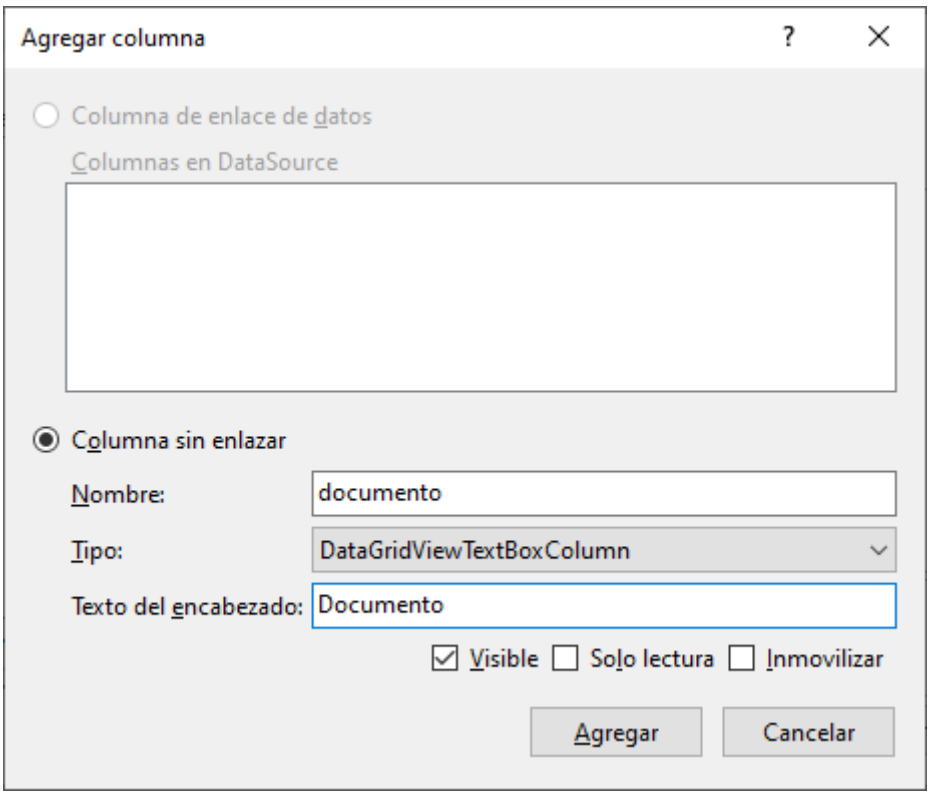
En la pestaña Datos vamos a agregar un control DataçgridView.



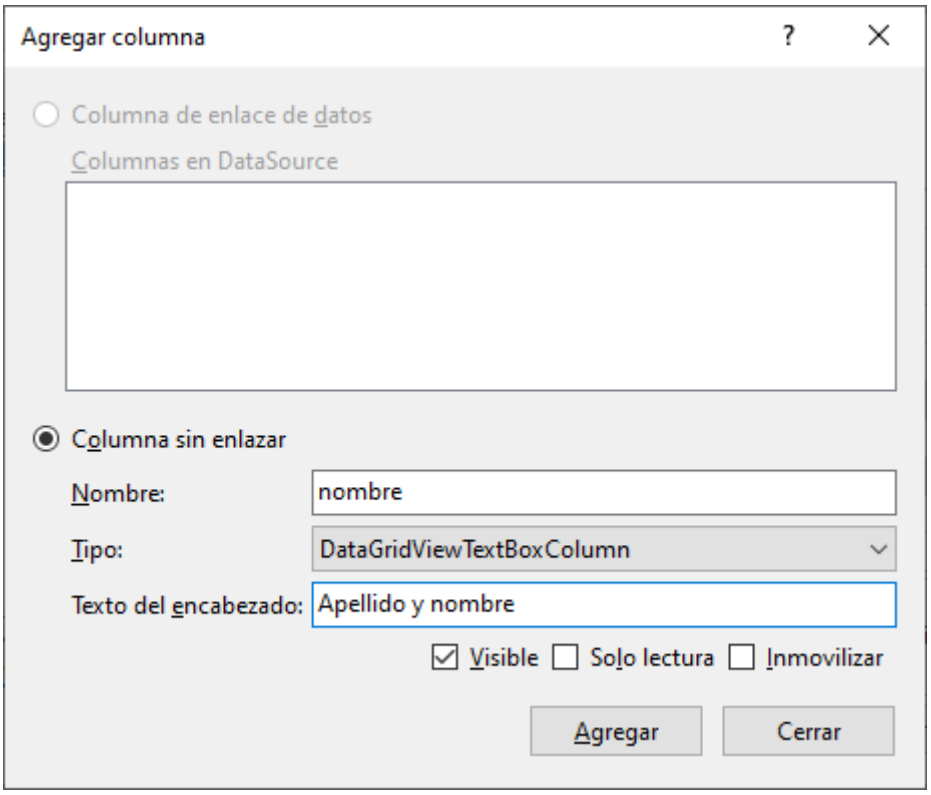
Vamos a editar columnas, (Edita columnas...).



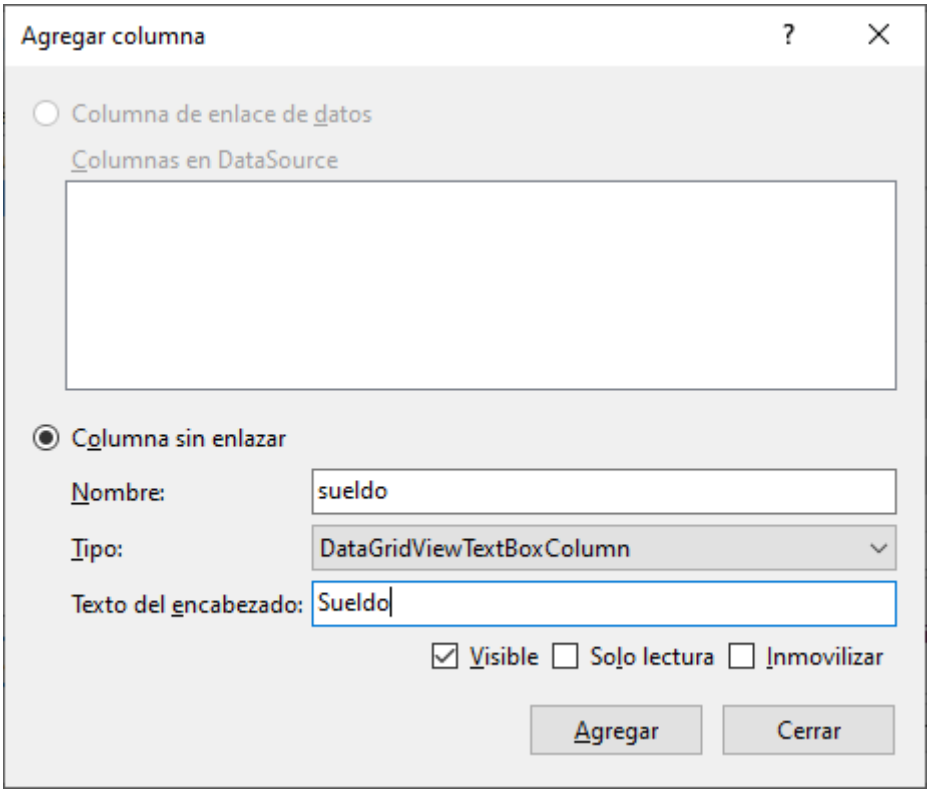
Seleccionamos Agregar…



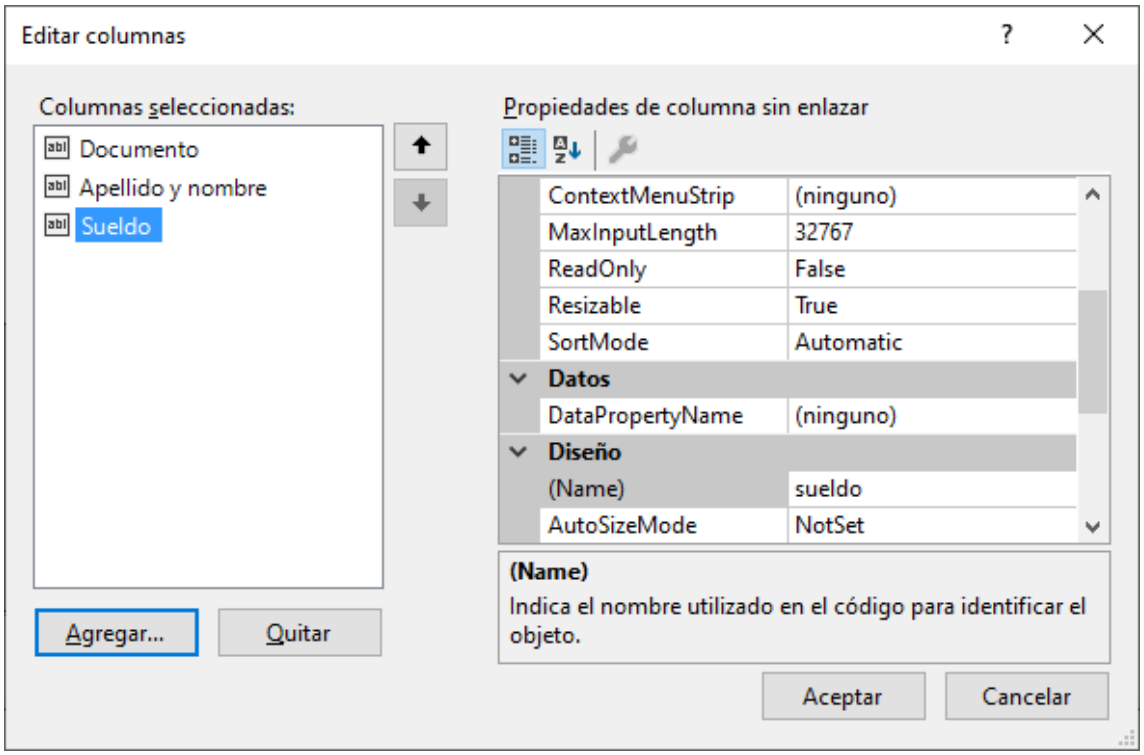
Le damos al botón Agregar.



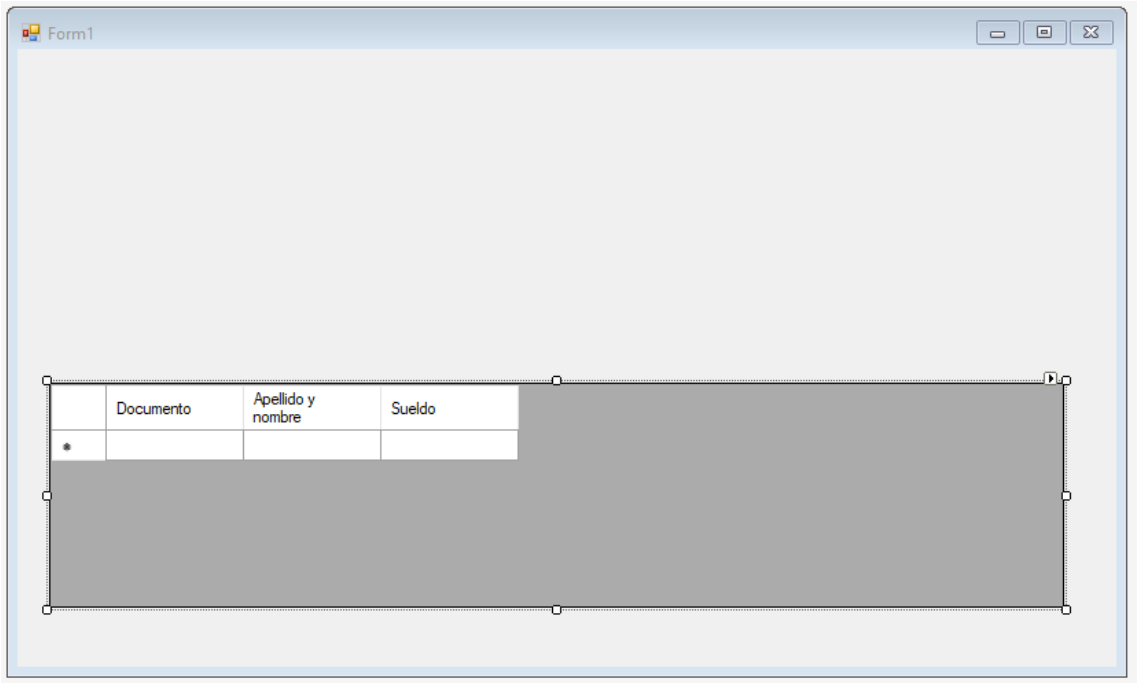
Le damos al botón Agregar.



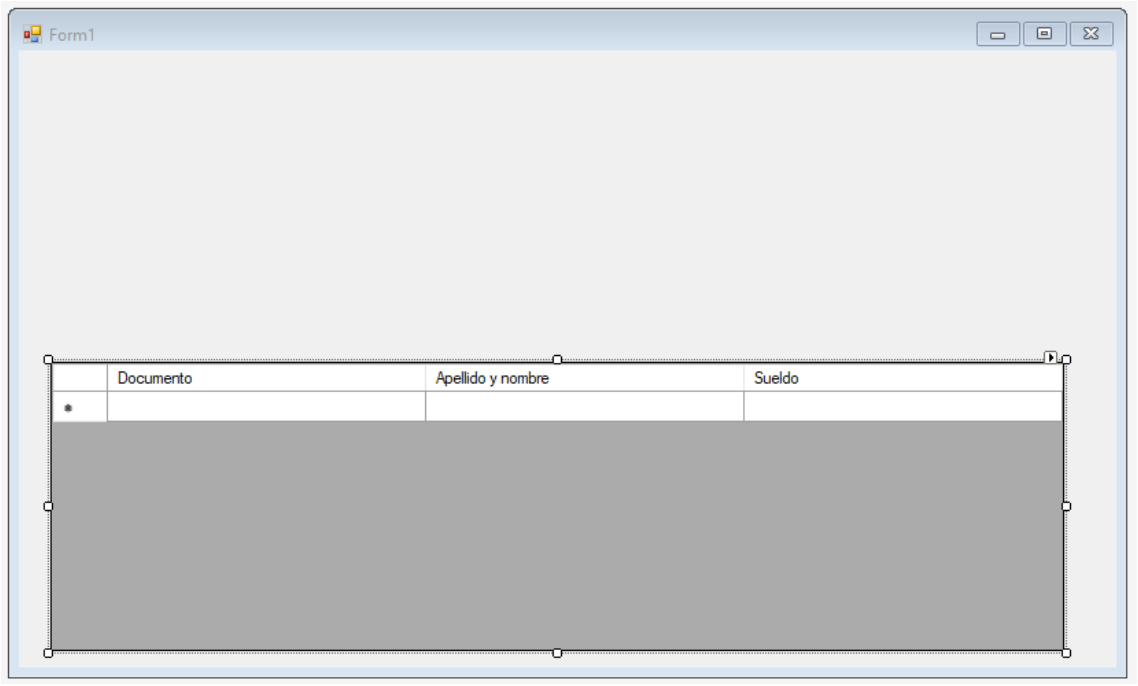
Le damos al botón Agregar y al botón Cerrar.



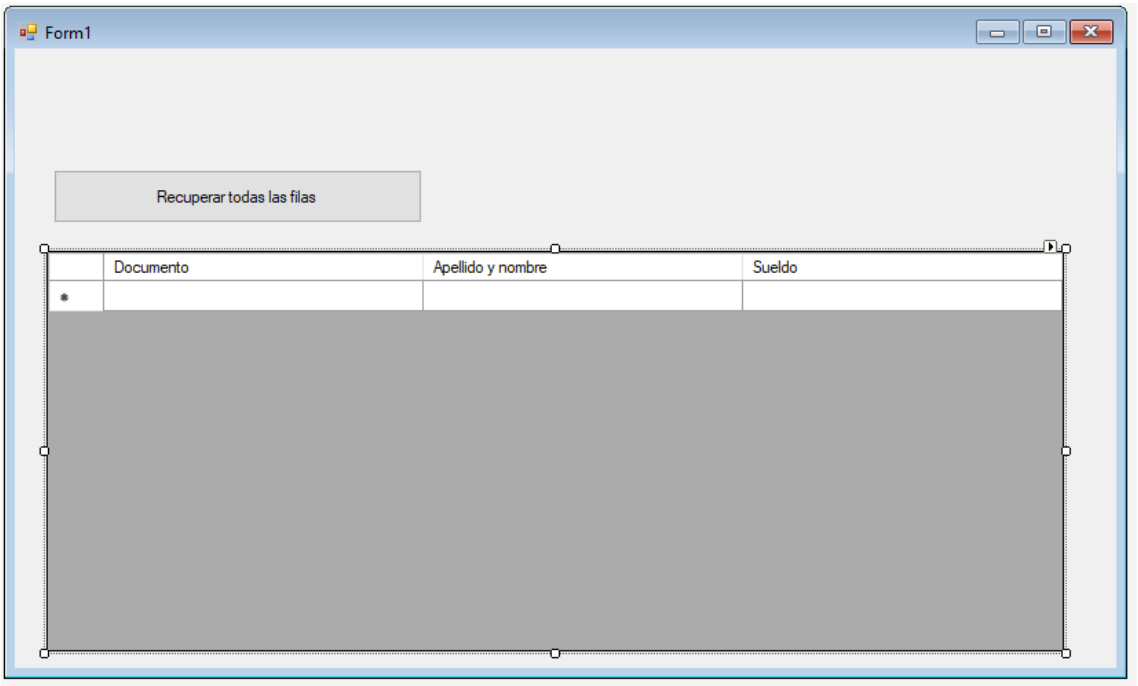
Le damos a Aceptar y lo modificamos en modo diseño para que ocupe todo el ancho.



Vamos a modificar la propiedad AutoSizeColumnsMode a Fill.



Vamos a agregar un botón con el texto "Recuperar todas las filas".

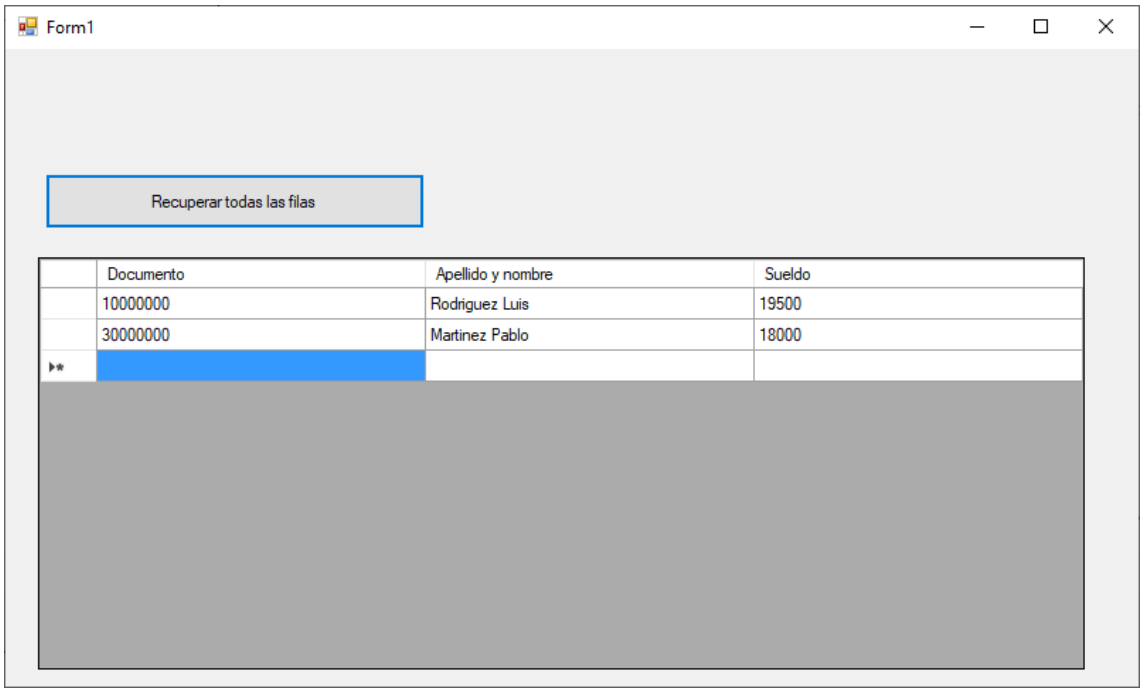


Vamos a codificar el botón con el evento click.

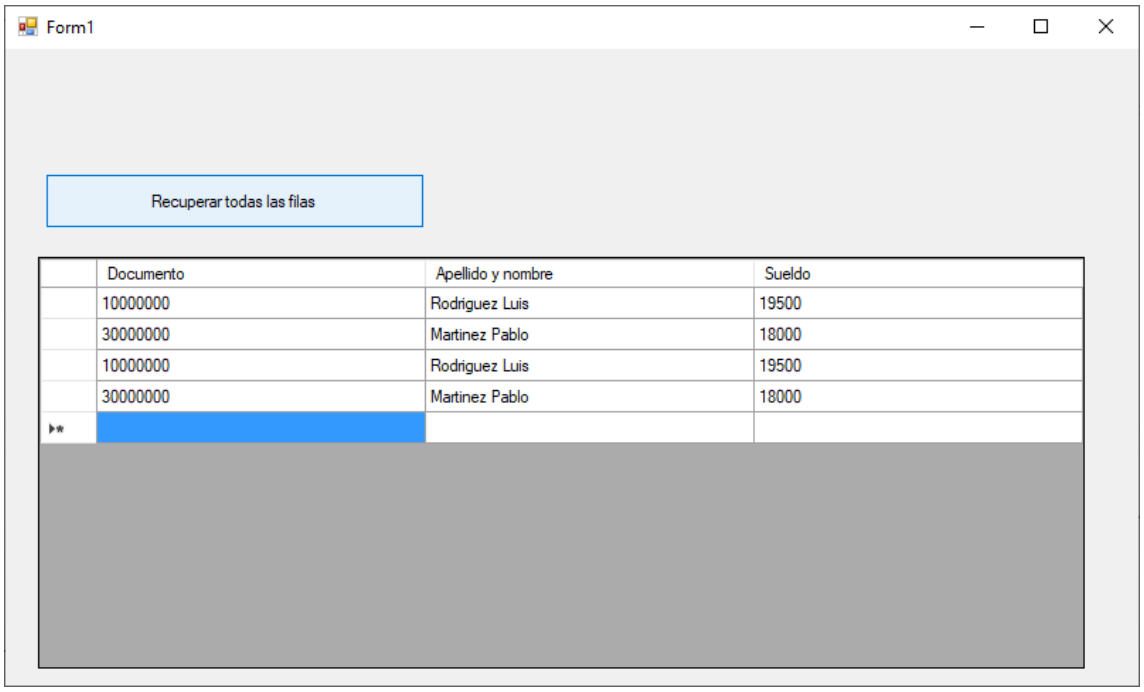
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto154
{
```

```
    public partial class Form1 : Form
    {
        private SqlConnection conexion = new SqlConnection("Data Source=DESKTOP-
V8ILAQJ\\SQLEXPRESS;Initial Catalog=db1;Integrated Security=True");

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            conexion.Open();
            string sql = "select documento, nombre, sueldo from empleados";
            SqlCommand comando=new SqlCommand(sql, conexion);
            SqlDataReader registros= comando.ExecuteReader();
            while(registros.Read())
            {
                dataGridView1.Rows.Add(registros["documento"].ToString(),
registros["nombre"].ToString(), registros["sueldo"].ToString());
            }
            conexion.Close();
        }
    }
}
```

Vamos a ejecutar la aplicación y hacer clic en el botón "Recuperar todas las filas".



Tenemos que tener cuidado de pulsar dos veces el botón.



Tenemos que modificar el código.

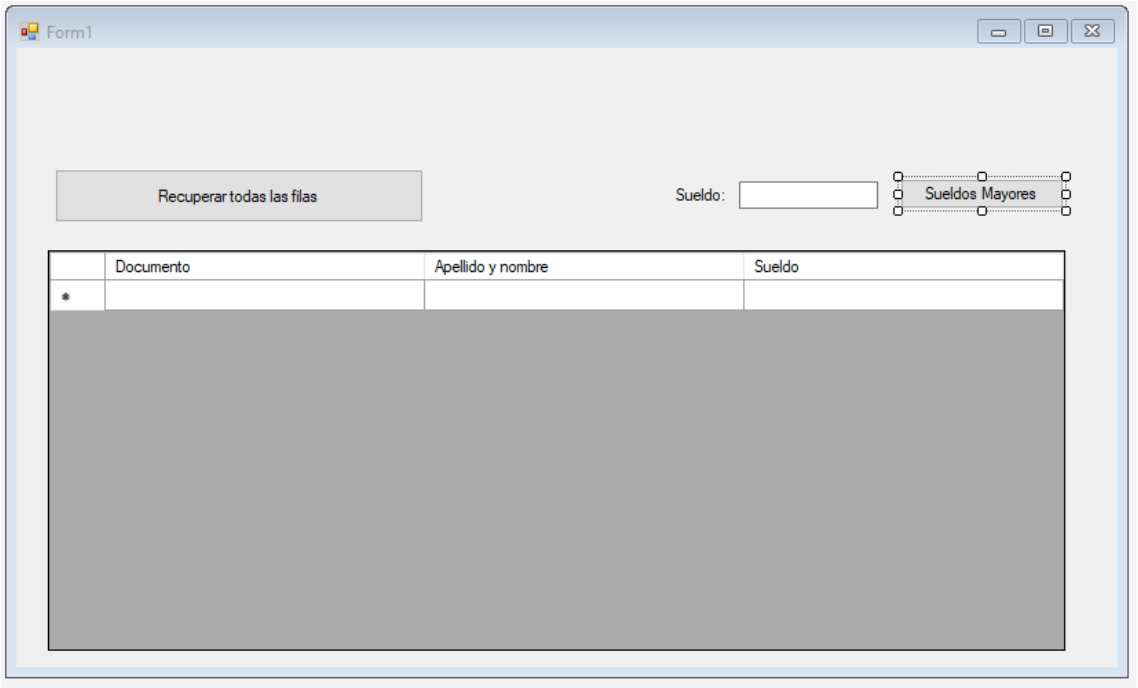
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = "select documento, nombre, sueldo from empleados";
    SqlCommand comando=new SqlCommand(sql, conexion);
    SqlDataReader registros= comando.ExecuteReader();
    dataGridView1.Rows.Clear();
    while(registros.Read())
    {
        dataGridView1.Rows.Add(registros["documento"].ToString(),
            registros["nombre"].ToString(), registros["sueldo"].ToString());
    }
    conexion.Close();
}
```

Agregamos la siguiente línea y cada vez que hagamos click en el botón primero borrará todas las filas.

Aunque presiones varias veces el botón las filas no se duplicarán.

Para que no se pueda editar en las propiedades de dataGridView1 pondremos ReadOnly a True.

Ahora queremos mostrar aquellos que tengan un sueldo mayor a la cantidad que le indiquemos, para ello vamos a agregar un Label un TextBox y un Button.



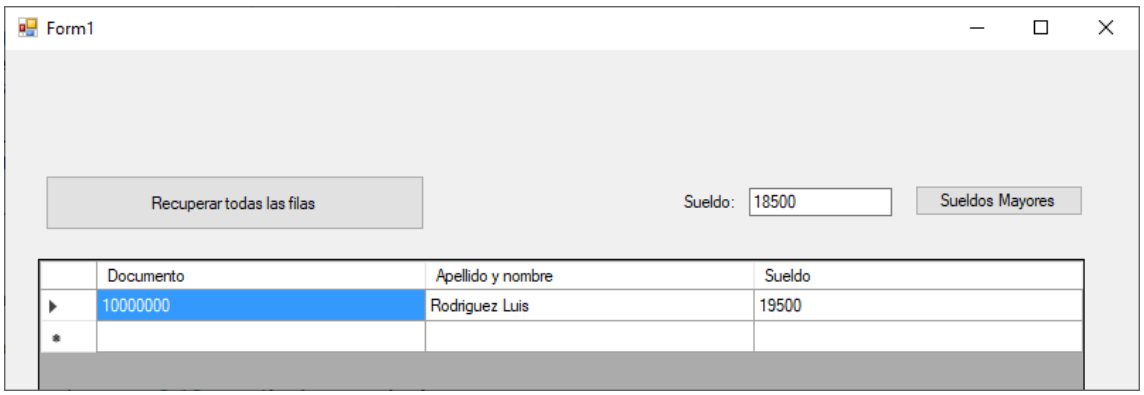
Vamos a programar el botón "Sueldos Mayores" el evento click.

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = $"select documento, nombre, sueldo from empleados where sueldo>{textBox1.Text}";
    SqlCommand comando = new SqlCommand(sql, conexion);
    SqlDataReader registros = comando.ExecuteReader();
    dataGridView1.Rows.Clear();
    while (registros.Read())
    {
        dataGridView1.Rows.Add(registros["documento"].ToString(),
            registros["nombre"].ToString(), registros["sueldo"].ToString());
    }
    conexion.Close();
}
```

Es una copia del código del botón "Recuperar todas las filas" a excepción de la modificación de la línea seleccionada.

Vamos a ejecutar:

Queremos consultar por los sueldos mayores de 18500.



.

Capítulo 144.- Inyección de SQL y Consultas parametrizadas

SqlCommand: Nos permite configurar un comando SQL (insert, delete, update, select, etc.) y su posterior comunicación con el servidor de SQL Server configurado previamente mediante un objeto de la clase SqlConnection.

Si la aplicación que desarrollamos va ha ser empleada solo por nosotros, la generación de los comandos SQL mediante la concatenación o interpolación de string es una opción válida. Seguramente no ingresamos datos para provocar fallos internacionales.

Por ejemplo si tenemos la tabla vehículos con la siguiente estructura:

```
if object_id('vehiculos') is not null
  drop table vehiculos;

create table vehiculos (
    patente char(7) primary key,
    marca varchar(50),
    precio float
);

insert into vehiculos(patente,marca,precio) values ('aa000aa','Renault',2000);
insert into vehiculos(patente,marca,precio) values ('bb000bb','Ford',6000);
insert into vehiculos(patente,marca,precio) values ('cc000cc','Audi',3000);
insert into vehiculos(patente,marca,precio) values ('dd000ee','Fial',2500);
```

Luego implementamos una aplicación para borrar un vehículo ingresando la patente e ingresamos por teclado la siguiente patente: 'or' 1'='1

Vamos al SQL.

Vamos a escribir el código SQL que tenemos en el ejemplo, lo seleccionamos todo y lo ejecutamos. Recuerda abrir la base de datos.

SQLQuery1.sql - DE...8ILAQJ\pmver (69))*

```
if object_id('vehiculos') is not null
    drop table vehiculos;

create table vehiculos (
    patente char(7) primary key,
    marca varchar(50),
    precio float
);

insert into vehiculos(patente,marca,precio) values ('aa000aa','Reanult',2000);
insert into vehiculos(patente,marca,precio) values ('bb000bb','Ford',6000);
insert into vehiculos(patente,marca,precio) values ('cc000cc','Audi',3000);
insert into vehiculos(patente,marca,precio) values ('dd000ee','Fial',2500);
```

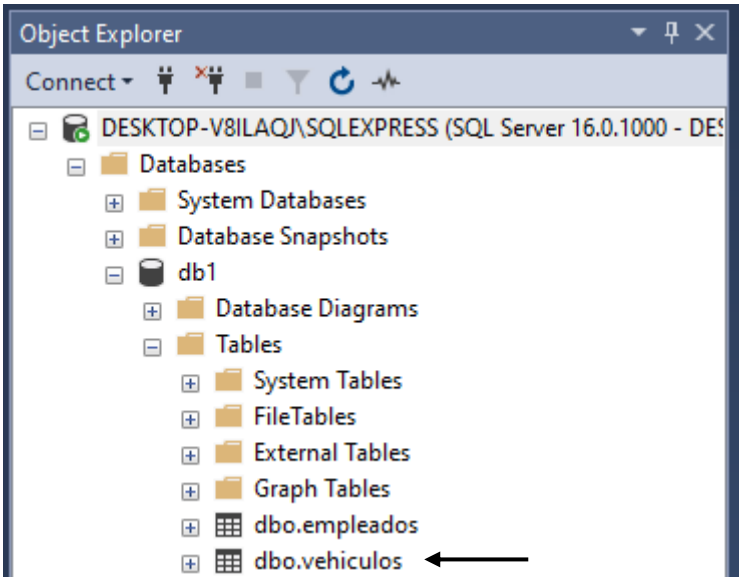
```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2023-07-13T06:17:32.1295844+02:00
```

Consultamos la tabla:

```
select * from vehiculos;
```

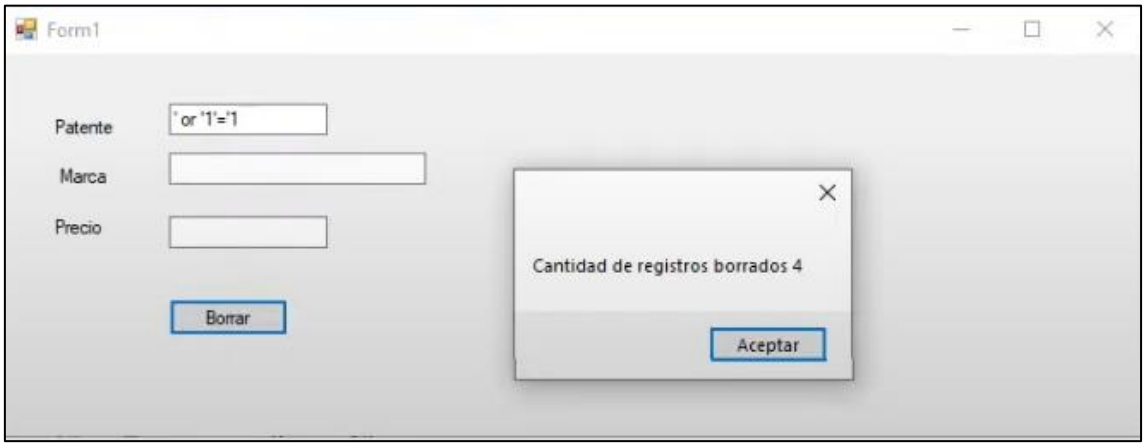
	patente	marca	precio
1	aa000aa	Reanult	2000
2	bb000bb	Ford	6000
3	cc000cc	Audi	3000
4	dd000ee	Fial	2500

Esta es la tabla vehículos.



Vamos empezar en proyecto nuevo en C#.

Para crear la siguiente interfaz:



Podemos comprobar que se han borrado las cuatro filas de la tabla vehículos. Esto es debido que hemos modificado el comando SQL que quería ejecutar. Hemos Inyctado SQL en nuestro comando original de SQL:

delete from vehiculos where patente='or '1'='1'

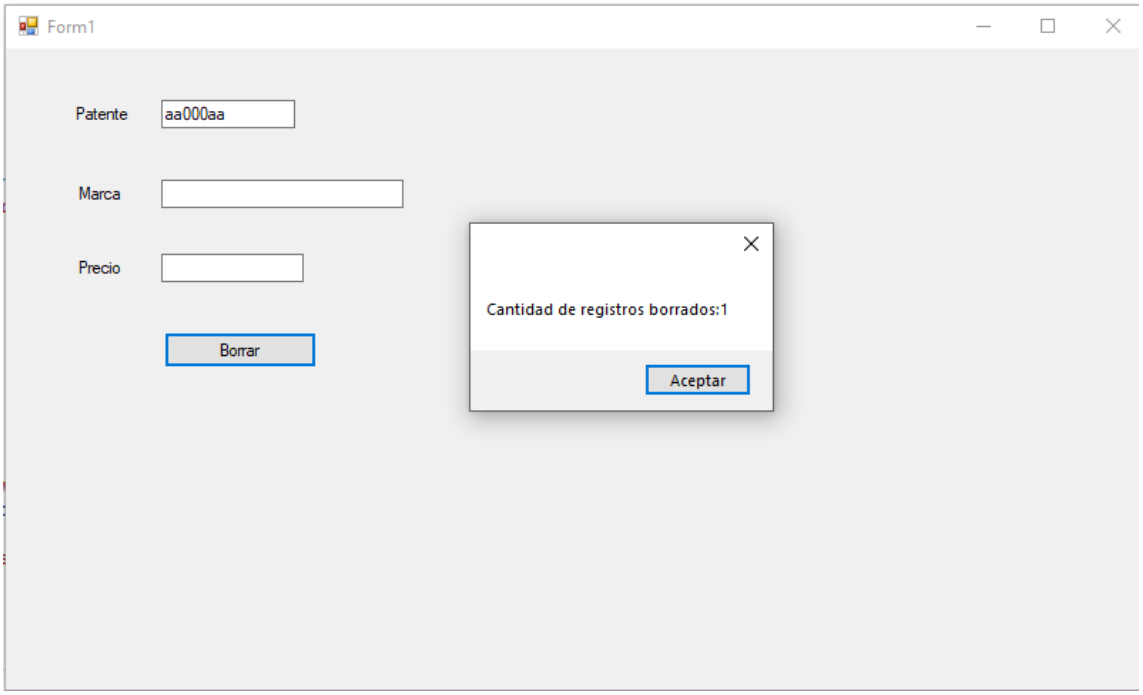
Vamos al Código del botón Borrar evento click.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Proyecto155
{
    3 referencias
    public partial class Form1 : Form
    {
        private SqlConnection conexion=new SqlConnection("Data Source=DESKTOP-V8ILAQJ\\SQLEXPRESS;" +
            "Initial Catalog=db1; Integrated Security=True");

        1 referencia
        public Form1()
        {
            InitializeComponent();
        }

        1 referencia
        private void button1_Click(object sender, EventArgs e)
        {
            conexion.Open();
            string sql = $"delete from vehiculos where patente='{textBox1.Text}'";
            SqlCommand comando = new SqlCommand(sql, conexion);
            int cant = comando.ExecuteNonQuery();
            MessageBox.Show("Cantidad de registros borrados:" + cant.ToString());
            conexion.Close();
        }
    }
}
```

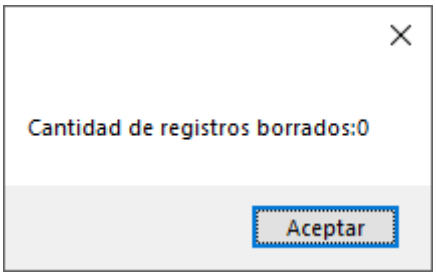
Configurar la conexión con el servidor SQL.

Retorna la cantidad de registros eliminados.

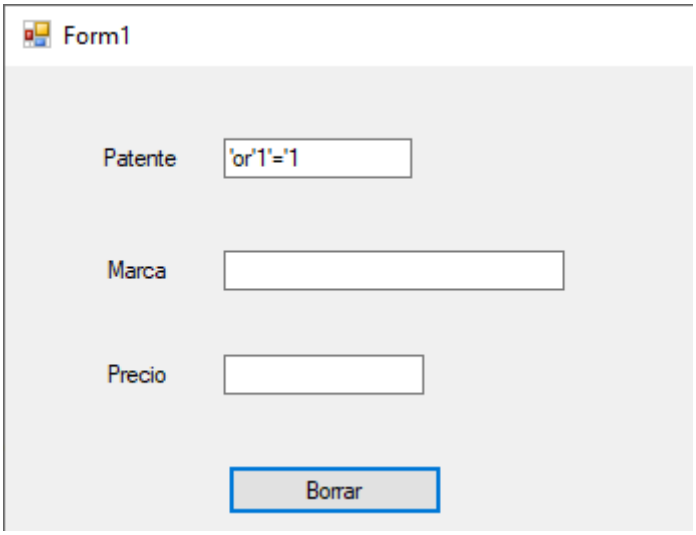
Vamos a ejecutar y eliminar el vehículo con la patente 'aa000aa'.

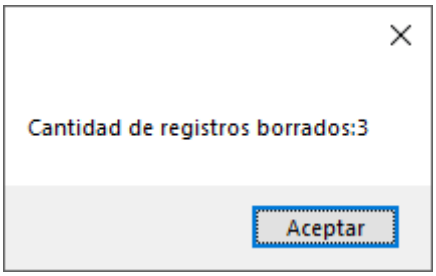


Si intentamos borrar la misma patente.



Que ocurre si en patente introducimos 'or'1'=1.





Se han borrado todas las filas de la tabla.

Con esta instrucción:

delete from vehículos where patente='' or '1'='1'.

Le estamos diciendo que borre los registros cuyo campo patente esté vacío o hacemos una comparación de 1 igual a 1 esto siempre dará verdadero, por este motivo elimina todas las filas.

Para que se pueda evitar que se eliminen todas las filas vamos a realizar lo siguiente:

Modificar el siguiente código:

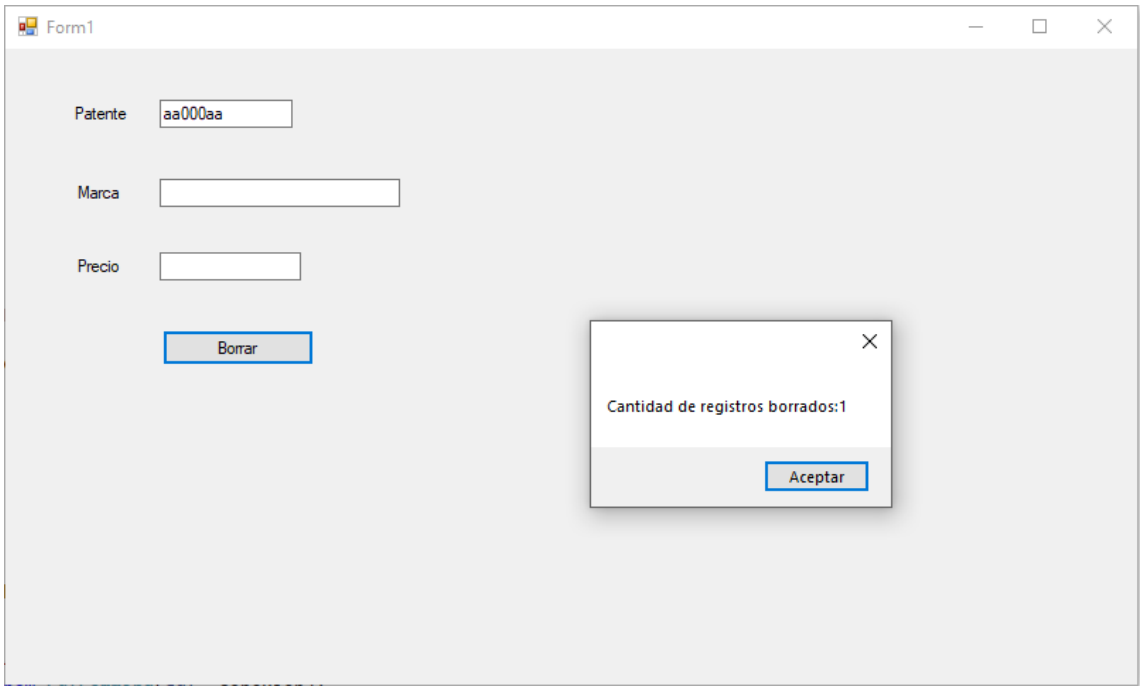
```
1 referencia
private void button1_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = "delete from vehiculos where patente=@patente";
    SqlCommand comando = new SqlCommand(sql, conexion);
    comando.Parameters.Add("@patente", SqlDbType.Char).Value=textBox1.Text;
    int cant = comando.ExecuteNonQuery();
    MessageBox.Show("Cantidad de registros borrados:" + cant.ToString());
    conexion.Close();
}
```

En el programa SQL vamos a cargar los registros de nuevo.

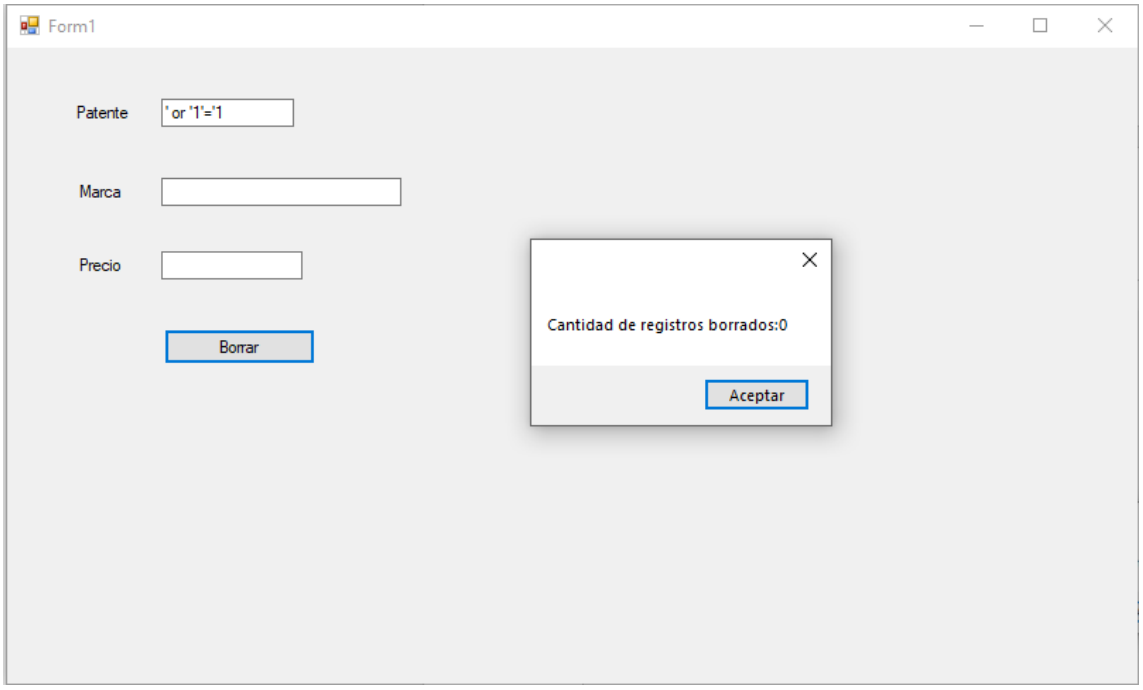
```
insert into vehiculos(patente, marca, precio) values ('aa000aa', 'Renault', 2000);
insert into vehiculos(patente, marca, precio) values ('bb000bb', 'Ford', 6000);
insert into vehiculos(patente, marca, precio) values ('cc000cc', 'Audi', 3000);
insert into vehiculos(patente, marca, precio) values ('dd000ee', 'Fial', 2500);
```

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

Completion time: 2023-07-13T09:17:01.4088917+02:00
```

Vamos a eliminar la patente 'aa000aa'.



Ahora intentamos pasar Sql. 'or'1'='1. esto se llama Inyección de sql.



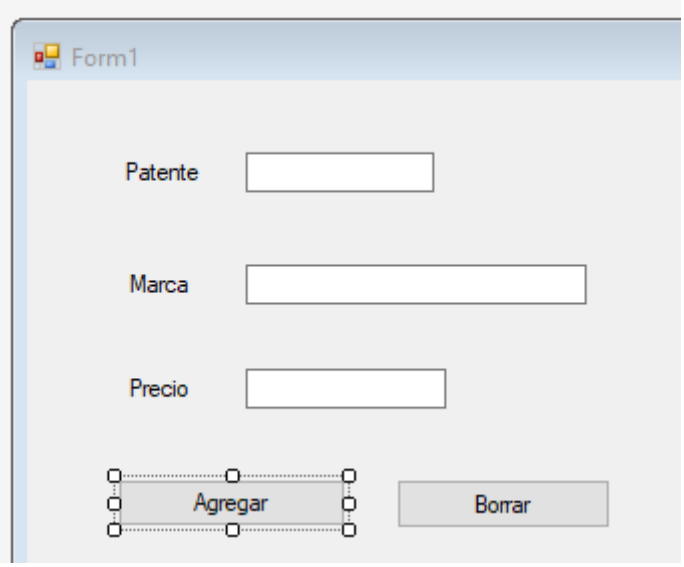
Vamos a consultar los registros desde SQL.

```
select * from vehiculos;
```

Este será el resultado:

	patente	marca	precio
1	bb000bb	Ford	6000
2	cc000cc	Audi	3000
3	dd000ee	Fial	2500

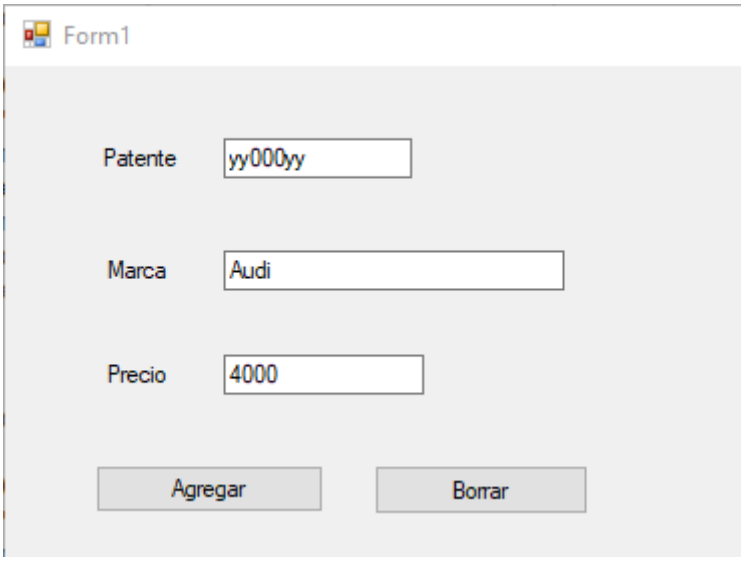
Vamos a añadir un segundo botón:



Vamos a programar el botón con el evento click.

```
1 referencia
private void button2_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = "insert into vehiculos(patente,marca,precio) values (@patente,@marca,@precio)";
    SqlCommand comando = new SqlCommand(sql, conexion);
    comando.Parameters.Add("@patente", SqlDbType.Char).Value = textBox1.Text;
    comando.Parameters.Add("@marca", SqlDbType.VarChar).Value = textBox2.Text;
    comando.Parameters.Add("@precio", SqlDbType.Float).Value = textBox3.Text;
    comando.ExecuteNonQuery();
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";
    conexion.Close();
}
```

Vamos a ejecutar y cargamos una patente.



Le damos al botón Agregar.

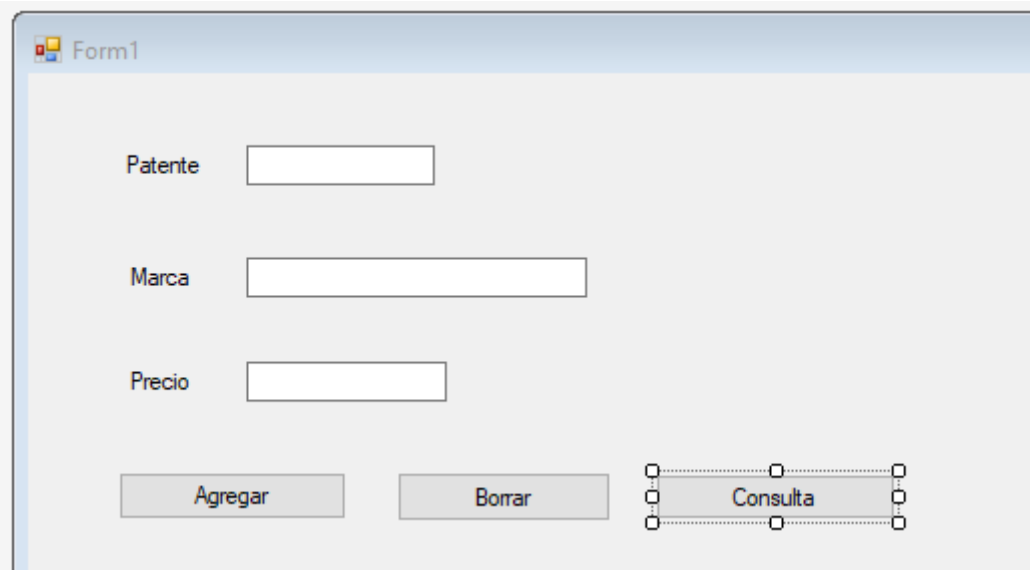
Vamos a consultar la tabla desde SQL.

```
select * from vehiculos;
```

Este será el resultado:

	patente	marca	precio
1	bb000bb	Ford	6000
2	cc000cc	Audi	3000
3	dd000ee	Fial	2500
4	yy000yy	Audi	4000

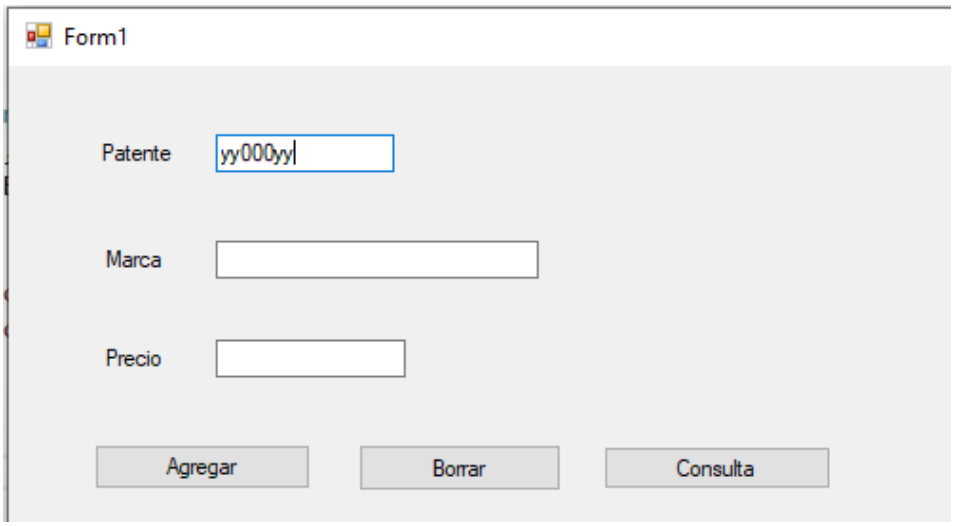
Vamos a agregar otro botón para realizar las consultas:



Vamos a codificar el botón Consulta con el evento click.

```
1 referencia
private void button3_Click(object sender, EventArgs e)
{
    conexion.Open();
    string sql = "select marca,precio from vehiculos where patente=@patente";
    SqlCommand comando = new SqlCommand(sql, conexion);
    comando.Parameters.Add("@patente", SqlDbType.Char).Value = textBox1.Text;
    SqlDataReader registro = comando.ExecuteReader();
    if (registro.Read())
    {
        textBox2.Text = registro["marca"].ToString();
        textBox3.Text = registro["precio"].ToString();
    }
    else
        MessageBox.Show("No existe un vehiculo con dicha patente");
    registro.Close();
    conexion.Close();
}
```

Vamos a ejecutar para consultar los datos de una patente.



Introducimos una patente seguido del botón Consulta.