CONTROLES COMBOBOX, LISTBOX Y DATAGRIDVIEW

Guía de laboratorio – Capítulo VI

Contenidos:

- ✓ Herramientas ComboBox y ListBox.
  - Propiedades y métodos comunes de las mismas.
  - Manipulación de los elementos de los controles.
- ✓ Control DataGridView
  - Creación y edición de columnas
  - Agregado de filas
  - Obtener información de filas y celdas
  - Eliminación de filas

HERRAMIENTAS: COMBOBOX – LISTBOX

Como vimos anteriormente en la guía "Introducción a las herramientas", los controles ComboBox y ListBox son similares en cuanto a la posibilidad de mostrar al usuario un listado de Ítems y darle la posibilidad de seleccionarlos para su uso.

Veremos a continuación qué otras acciones nos ofrecen ambos:

COMBOBOX

Comencemos con este control y para ello crearemos un proyecto de Windows forms arrastrando un objeto comboBox hacia el formulario creado y modificaremos algunas de sus propiedades:

comboBox1	
Name	cmbAnimales
DropDownStyle	DropDownList
Sorted	True *(los valores que contengan la lista de elementos, aparecerán ordenados)*
Items	Gato Perro Hamster

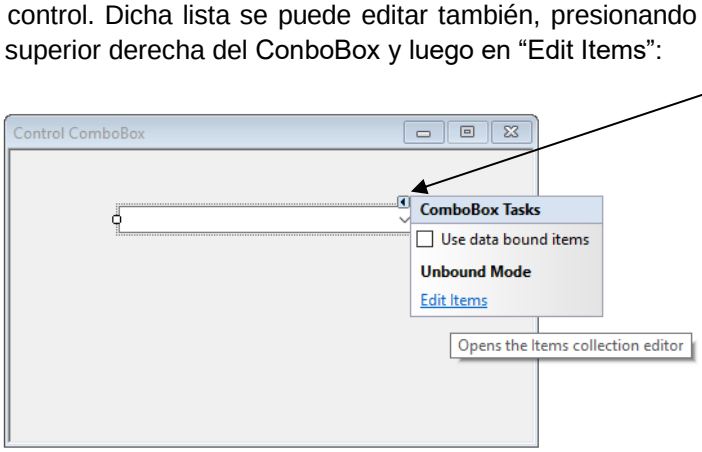
DropDownStyle ofrece tres opciones: Simple, DropDown y DropDownList.

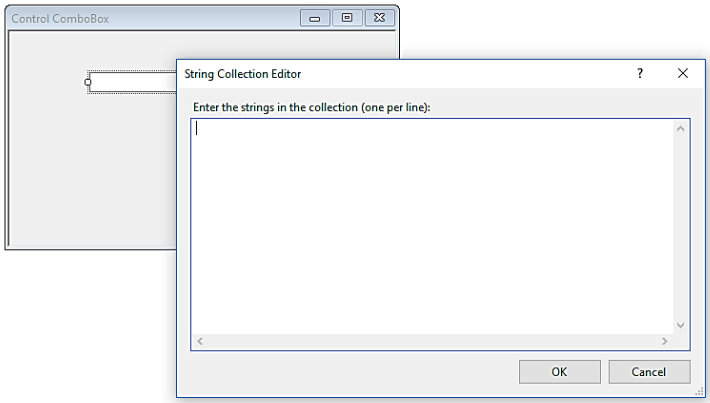
La opción "Simple" permite que la lista del control esté siempre visible y que el usuario pueda ingresar un nuevo valor y no esté limitado a seleccionar un valor existente en la lista.

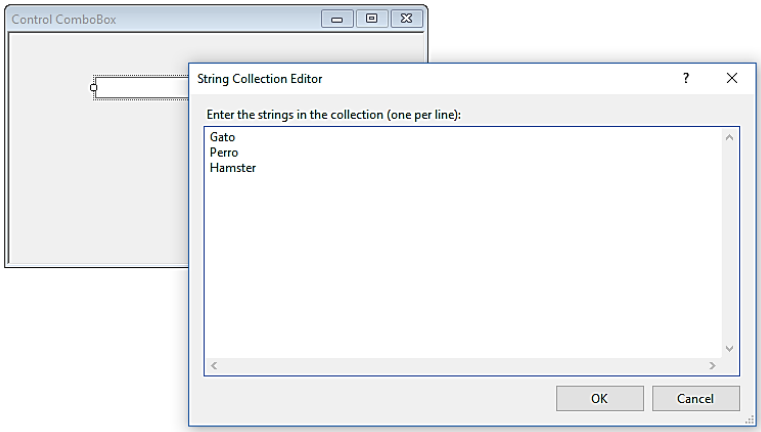
"DropDown" es el estilo por defecto; permite que la lista sea visible al presionar la flecha del control y que el usuario pueda ingresar un nuevo valor y no esté limitado a seleccionar un valor existente en la lista, también permite visualizar la propiedad Text si deseamos que inicie con un texto específico.

"DropDownList" permite que la lista sea visible al presionar la flecha del control y que la parte del texto no sea editable, por lo que el usuario no podrá agregar otro valor que no sean los establecidos en la lista.

La propiedad "**Items**" permite crear en modo de diseño, la lista de elementos que conforman la colección y que el usuario verá al desplegar el control. Dicha lista se puede editar también, presionando el pequeño cuadro con una flecha ubicado en la esquina superior derecha del ConboBox y luego en "Edit Items":







Agregaremos al formulario un textBox (txtAnimal) y dos button (btnAgregar y btnEliminar).

Agregar ítems:

Tal como ya lo hemos hecho en ejercicios anteriores, en el evento click del **btnAgregar** colocaremos la siguiente línea de código:

```
cmbAnimales.Items.Add(txtAnimal.Text);
```

La misma, accede a la lista de ítems del cmbAnimales y con el método "Add" le agrega un nuevo valor para la lista desplegable. Dicho valor será un string, en este caso tomado del txtAnimal al que previamente le asignamos un texto.

También se pueden utilizar datos almacenados en arrays para agregar en este control:

Lo que haremos ahora será crear un array con valores preliminares para agregar posteriormente al cmbAnimales:

En el evento Load() de nuestro formulario agregaremos el siguiente código:

```
private void Form1_Load(object sender, EventArgs e)
{
    string [] arrayAnimales = { "Mono", "Oso", "Jirafa", "Jaguar", "Puma" };

    Array.Sort(arrayAnimales);//ordenamos de forma descendente los Items

    for (int i = 0; i < 5; i++)//recorremos nuestro array de 5 elementos
    {
        cmbAnimales.Items.Add(arrayAnimales[i].ToString());
    }
}
```

En el caso de no conocer la cantidad de elementos de nuestro array, podemos usar un foreach en vez del for:

```
foreach (string s in arrayAnimales)
{
    cmbAnimales.Items.Add(s);
}
```

Como resultado, nuestro cmbAnimales, tendrá los 3 valores agregados en diseño, los 5 valores agregados con este código y lo que agreguemos con el texBox y el botón Agregar.

Eliminar ítems:

Una vez comprobado que el valor es registrado, podemos agregar al evento Click del **btnEliminar** la siguiente línea de código:

```
if (cmbAnimales.SelectedIndex != -1) //evitamos el error si no hay una selección previa
cmbAnimales.Items.RemoveAt(cmbAnimales.SelectedIndex);
```

La segunda línea permite remover sólo el elemento seleccionado de la lista de ítems. En el caso de querer eliminar todos los elementos de la colección, utilizamos el método Clear().

```
cmbAnimales.Items.Clear();
```

Tener en cuenta que al eliminar un ítem, se ejecuta el evento **SelectedIndexChanged**. Controlar posibles errores derivados de la falta de control en este caso.

Buscar ítems:

Otra acción que podemos realizar con el comboBox es buscar un valor en la lista de elementos y que este valor sea seleccionado. Para ejemplificar, agregaremos al formulario un nuevo button (btnBuscar) y un label (lblResultado):

En el código del evento Click del **btnBuscar**, colocaremos las siguientes líneas:

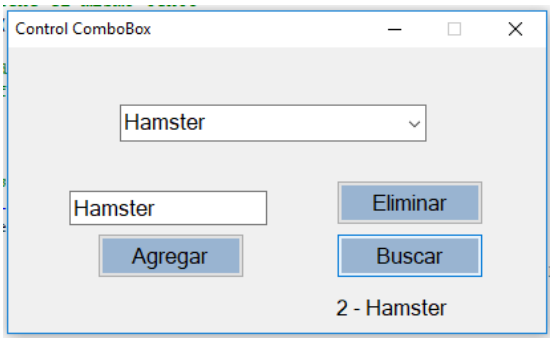
```
//buscamos el valor y asignamos su índice a la variable index
int index = cmbAnimales.FindString(txtAnimal.Text);

//establecemos el valor que buscamos como ítem seleccionado
cmbAnimales.SelectedIndex = index;

//Controlamos posible error y
//mostramos en el lblResultado el índice y el valor seleccionado
if (index != -1)
{ lblResultado.Text = index + " - " + cmbAnimales.Items[index].ToString(); }
else
{ lblResultado.Text = "Sin resultados"; }
```

En dichas líneas, creamos una variable entera llamada index, que guardará el resultado de la búsqueda realizada con el método FindString, el cual devuelve el índice donde se encuentra el valor aproximado buscado dentro de la colección.

En caso de no obtener resultados, devuelve el valor “-1”, por lo que debemos controlar este posible error.



Otra variante en la búsqueda es “FindStringExact” el cual devuelve el índice del ítem siempre que sea exactamente igual al valor buscado, de no serlo devuelve **-1**.

LISTBOX

Como ya hemos visto, este control contiene una colección de elementos que, al contrario del comboBox, se mantiene de forma estática y no en una lista desplegable.

Los usuarios pueden seleccionar uno o más elementos de la lista. Se puede usar un ListBox para mostrar varias columnas, aunque se recomienda para un mejor diseño e implementación utilizar un ListView.

Las propiedades comunes a la mayoría de los controles vistos también forman parte de la lista de propiedades del listBox.

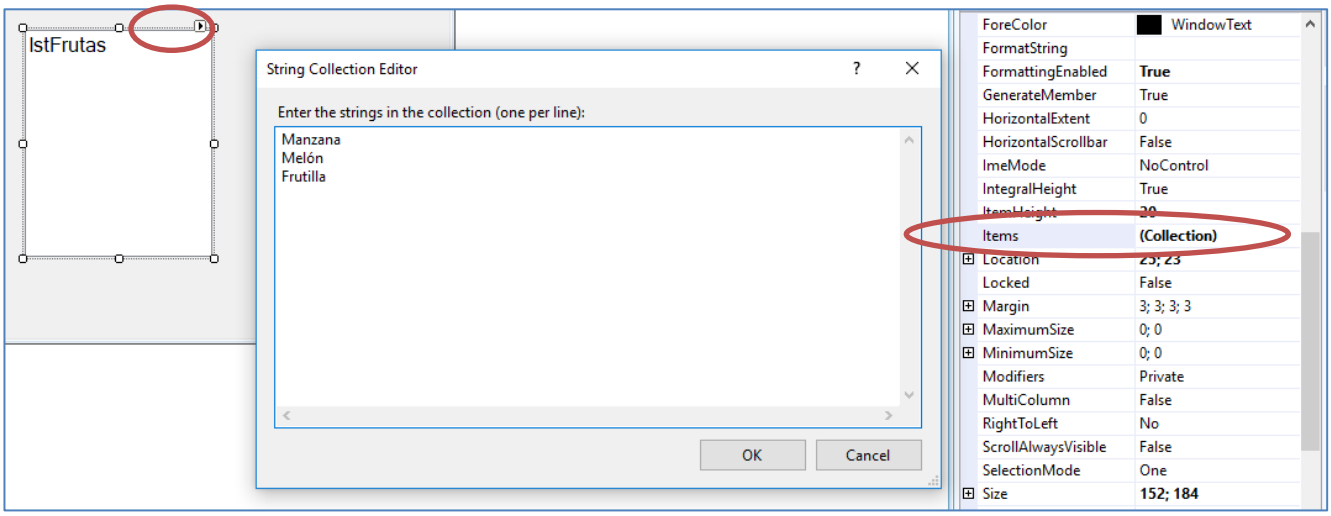
Para conocer mejor este control comenzaremos con un nuevo formulario dentro del mismo programa y colocaremos un textBox, un button (Agregar) y un ListBox con las siguientes propiedades:

listBox1	
Name	lstFrutas
Items	Manzana Melón Frutilla
Sorted	True

Agregar ítems

Para agregar la lista de elementos, podremos hacerlo de la misma forma que en el comboBox, presionando la pequeña flecha en el control, por medio de la propiedad **Items**, o por medio del código en tiempo de ejecución (dentro del evento Click del button **Agregar**):

```
lstFrutas.Items.Add(txtFrutas.Text);
```



Con la propiedad **Sorted** en "true", lograremos que cada elemento agregado a la lista sea ordenado de forma tradicional, alfabéticamente o de menor a mayor, sea durante el diseño o en tiempo de ejecución.

Eliminar ítems

Para eliminar los elementos del control, podemos utilizar el evento **DoubleClick** del listBox y agregar el siguiente código:

```
int indice = lstFrutas.SelectedIndex;

if (indice > -1)
{
    lstFrutas.Items.RemoveAt(indice);
}
```

Tener en cuenta que, al eliminar un ítem, se ejecuta el evento **SelectedIndexChanged**. Esto lo controlamos con el If.

También crearemos un botón “Limpiar” para eliminar todos los elementos de la lista. El código es el siguiente:

```
lstFrutas.Items.Clear();
```

Buscar ítems

De la misma forma que con el comboBox, podemos buscar un elemento de la lista de ítems de la siguiente manera:

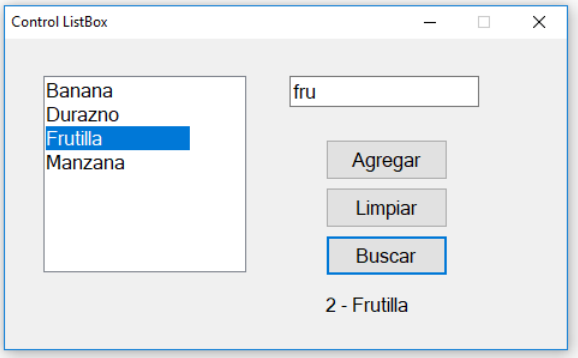
```
//buscamos el valor y asignamos su índice a la variable index
int index = lstFrutas.FindString("fru");

//establecemos el valor que buscamos como ítem seleccionado
lstFrutas.SelectedIndex = index;

//Mostramos en el lblResultado el índice y el valor seleccionado
if (index != -1)
{ lblResultado.Text = index + " - " + lstFrutas.Items[index].ToString(); }
else
{ lblResultado.Text = "Sin resultados"; }
```

Para mostrar el elemento encontrado, utilizar un Label y un Button para la búsqueda.

También podemos utilizar el método **FindStringExact** donde el valor a buscar debe ser exactamente igual al de la lista de ítems.

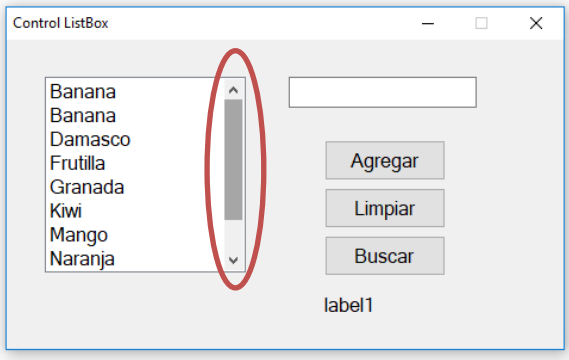


También podemos utilizar la clase String para la búsqueda de datos en nuestras listas:

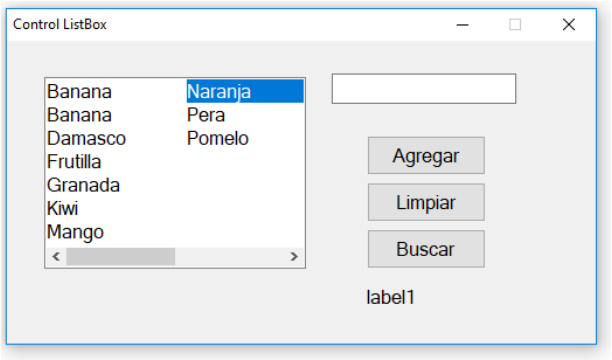
Columnas en un ListBox

- ✓ Por cantidad de ítems

Cuando una lista de ítems supera el tamaño visible del ListBox, por lo general se establece la propiedad **ScrollAlwaysVisible** en “True”, habilitando la barra desplazadora vertical de requerirse:



Pero existe otro modo de visualizar los datos, sin necesidad de utilizar únicamente la barra de desplazamiento. En este caso, si establecemos la propiedad **MultiColumn** en “True”, obtendremos el siguiente resultado:



De esta forma, dicha propiedad habilita a mostrar los datos horizontalmente en columnas y no en una única lista.

- ✓ Por variedad de datos:

En el caso de que los datos a mostrar deban estar en columnas diferentes, por ejemplo, las frutas y sus precios, deseamos utilizar un formato diferente al momento de agregar cada ítem. Aunque hay otros objetos que se podrían utilizar de mejor forma, veremos cómo implementar este formato con un listBox.

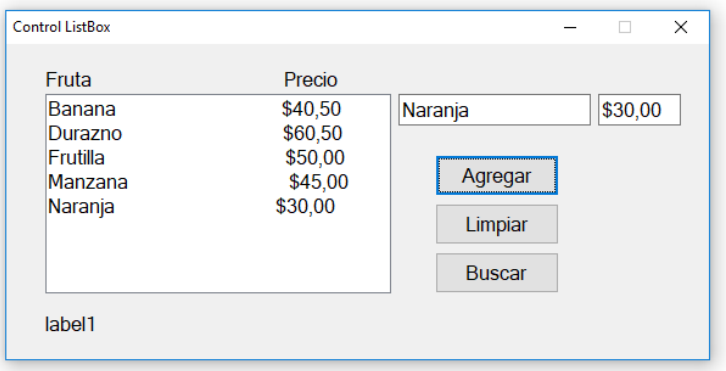
Luego del constructor, creamos una instancia de la clase String para indicar el formato en que serán presentados los ítems de la lista:

```
public frmList()//constructor
{
    InitializeComponent();
}
//creamos un elemento de la clase String y establecemos su formato
String StrColumnas = "{0, -40}{1, -40}";
```

Dentro de cada conjunto de llaves, establecemos el índice de cada columna, iniciando en 0 (cero), siendo **0** la primera columna y **1** la segunda. También se indica el ancho de cada columna, colocando el guión para establecer la alineación a la izquierda. (Para alinear a la derecha se escribe sin guión)
Al momento de agregar cada elemento a la lista de ítems, el código a utilizar sería el siguiente:

```
lstFrutas.Items.Add(String.Format(StrColumnas, "Manzana", "$45,00"));
```

De esta forma le estaremos agregando un nuevo ítem pero con el formato ya establecido en **StrColumnas**, debiendo a continuación colocar los valores correspondientes a cada columna. El resultado de esto sería similar al siguiente:



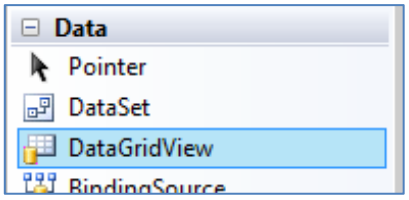
Utilizaremos un nuevo TextBox para agregar el precio al listBox y dos Label para el encabezado de las columnas.

La alineación de los datos en las columnas depende del tamaño y el tipo de fuente utilizados en el diseño. Estos detalles se trabajan de mejor forma con el control ListView.

CONTROL DATAGRIDVIEW

Esta herramienta es sumamente útil, ya que muestra los datos en una especie de cuadrícula o tabla gráfica y personalizable. La personalización de este control incluye la manipulación de celdas, filas, columnas y bordes mediante el uso de propiedades.

Se puede utilizar un DataGridView para mostrar datos con o sin un origen de datos. Sin especificar un origen de datos, se pueden crear columnas y filas que contengan datos y agregarlos directamente a DataGridView usando las propiedades Rows y Columns. También se puede usar la colección Rows para acceder a los objetos DataGridViewRow y la propiedad DataGridViewRow.Cells para leer o escribir valores de celda directamente.



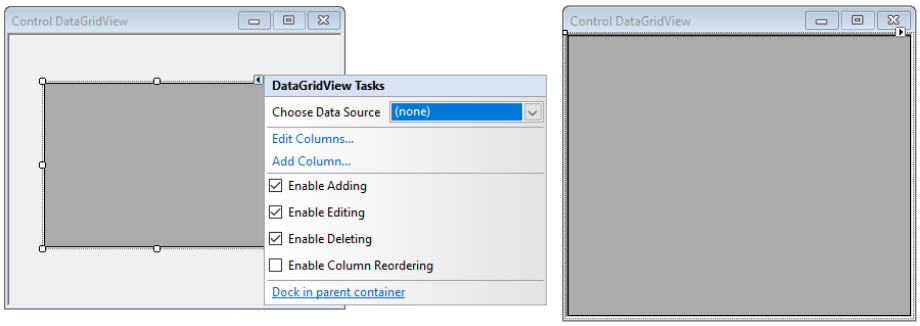
Como alternativa a llenar el control manualmente, puede configurar las propiedades DataSource y DataMember para enlazar DataGridView a una fuente de datos y rellenarla automáticamente con ellos.

Veremos todas las posibles formas de utilizar esta herramienta, comenzando con la carga directa de datos:

SIN ESPECIFICAR ORÍGEN DE DATOS

Para comenzar crearemos un formulario nuevo en el cual incluiremos inicialmente un DataGridView al que denominaremos **dgvDatos**.

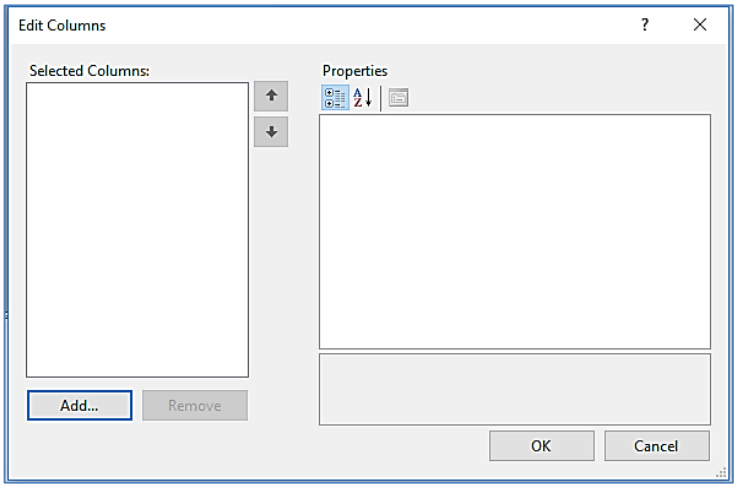
Se puede modificar su tamaño hasta llegar al deseado o adaptar su tamaño al del formulario (**Dock in parent container**):



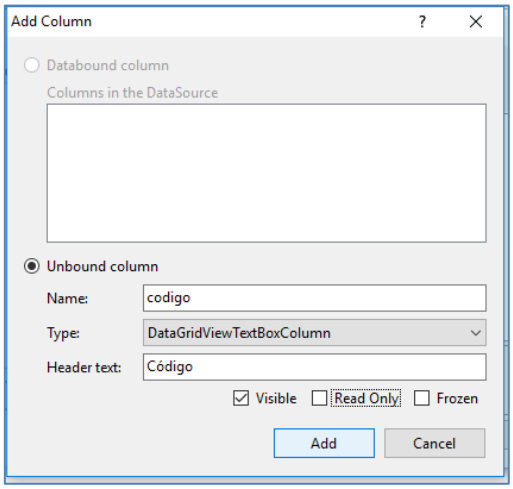
En este caso no utilizaremos esta opción ya que agregaremos más controles a continuación.

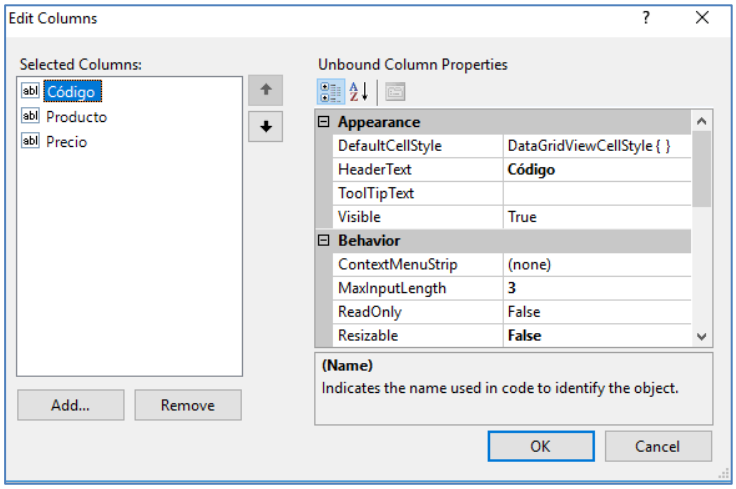
Agregar columnas:

Comenzaremos a agregar las columnas, haciendo click en la opción **Edit Columns**, mostrada en la imagen anterior, o haciendo click en la propiedad **Columns** y se visualizará la siguiente ventana:

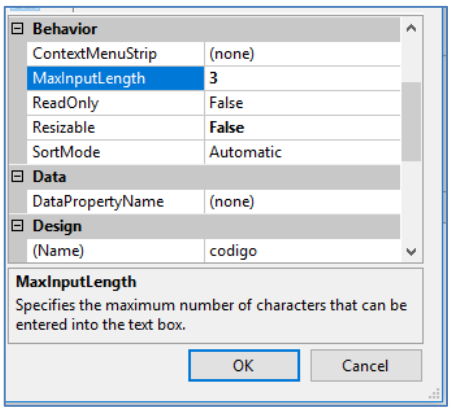


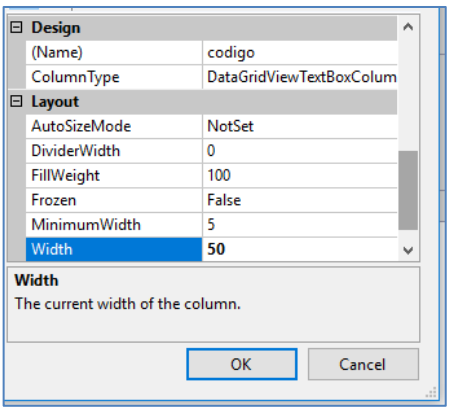
Mediante el botón **Add…** crearemos las columnas de nombre **codigo**, **producto** y **precio** con los nombres visibles (Header Text) **Código**, **Producto** y **Precio** respectivamente:



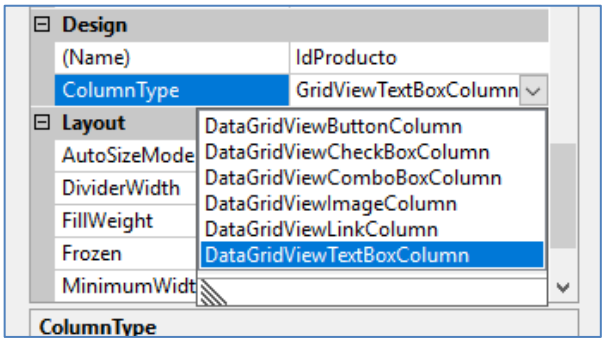


Notarán que, al finalizar de agregar las columnas, las mismas tienen sus propiedades y se visualizan en el sector derecho de la ventana. Con ellas podrán, entre otras cosas, modificar el estilo de la grilla (DefaultCellStyle) si así lo desean. Luego establecerán para la columna código el máximo de caracteres para ingresar **3** (MaxInputLength) y el ancho de las columnas código y grupo (Width):

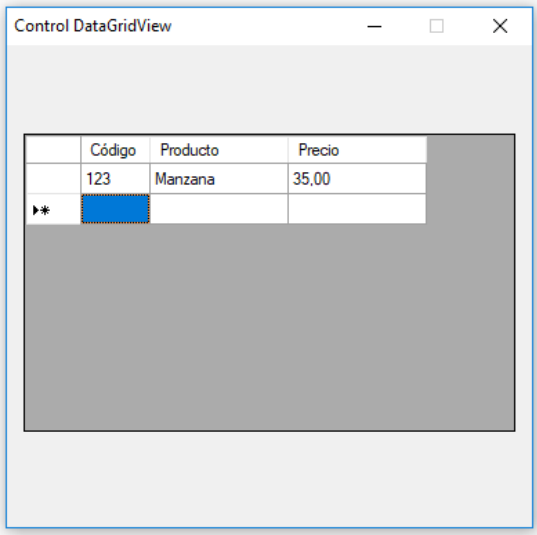




Podemos crear varios tipos de columnas, no exclusivamente de texto:

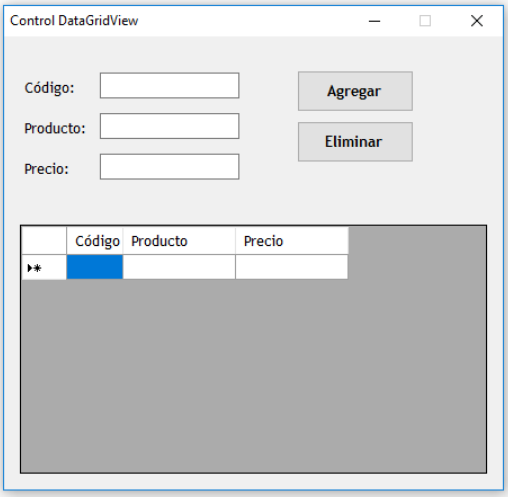


Una vez creadas las columnas, al ejecutar el programa, podremos cargar manualmente los datos en nuestro control y efectivamente, como lo hemos indicado, en la columna Código, no se permitirá el ingreso de más de tres caracteres:



Afregar filas:

A continuación, agregaremos 3 labels, 3 textBox y un button de la siguiente forma:



Modificar las siguientes propiedades de los controles:

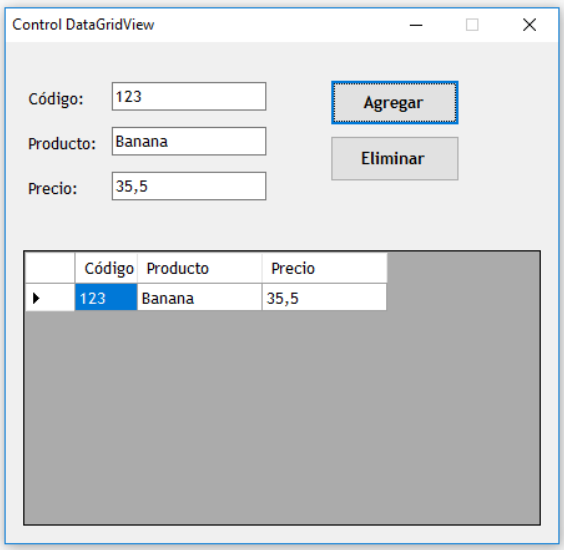
txtCodigo	
MaxLenght	3
dgvProductos	
AllowUserToAddRows	False (esto evita que el usuario cree nuevas filas mediante la edición de la misma grilla)

En el evento Click del **btnAgregar** escribiremos el siguiente código:

```
private void btnAgregar_Click(object sender, EventArgs e)
{
    int n = dgvProductos.Rows.Add();//Agregamos una nueva fila

    //Asignamos el valor del textBox a la celda que
    //corresponda de esa fila
    dgvProductos.Rows[n].Cells[0].Value = txtCodigo.Text;
    dgvProductos.Rows[n].Cells[1].Value = txtProducto.Text;
    dgvProductos.Rows[n].Cells[2].Value = txtPrecio.Text;
}
```

Primero se establece el índice de la nueva fila creada y se almacena en la variable de tipo int **n**. Luego asignamos los valores de los textBox a la celda que corresponda, indicando la fila (Rows[**n**]) y la columna (Cells[0]), en este caso como es la primera, corresponde el índice cero.



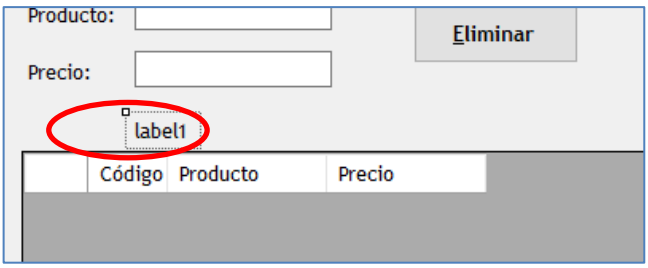

```
//OTRA FORMA DE CREAR UNA NUEVA FILA
//creamos un objeto con el formato de una fila
object[] fila = { txtCodigo.Text, txtProducto.Text, txtPrecio.Text };
//lo asignamos a la colección Rows de la grilla
dgvProductos.Rows.Add(fila);
```

Para modificar los valores simplemente se puede hacer dentro de la grilla si la propiedad **ReadOnly** tiene como valor **False**. Para evitar su modificación, establecer dicha propiedad en **True**.

Obtener información de las filas:

Si queremos obtener la información de una fila o celda en particular, disponemos del evento CellClick.

Para realizar esto, agregaremos un Label en el formulario.

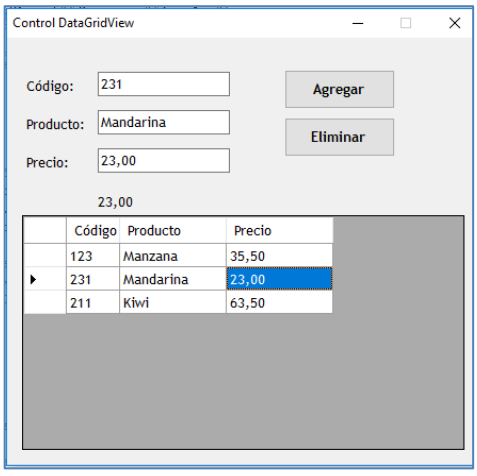



```
private void dgvProductos_CellClick(object sender, DataGridViewCellEventArgs e)
{
      //obtenemos el índice de la fila seleccionada
      //mediante la información que brinda el evento en su argumento e
      index = e.RowIndex; //declarar previamente la variable fuera de los métodos

      if (index != -1) //controlamos que exista una fila seleccionada
      {
         //Asignamos cada columna de la fila al textBox correspondiente
         txtCodigo.Text = dgvProductos.Rows[index].Cells[0].Value.ToString();
         txtProducto.Text = dgvProductos.Rows[index].Cells[1].Value.ToString();
         txtPrecio.Text = dgvProductos.Rows[index].Cells[2].Value.ToString();

          //asignamos al label el valor de la celda seleccionada
    lblCelda.Text = dgvProductos.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
      }
 }
```

Lo que hacemos con este código es volver a mostrar en los textBox los datos de la fila que seleccionemos del dataGridView y mostrar en el Label el valor sólo de la celda seleccionada:




```
//otra forma de obtener el valor de la celda:
lblCelda.Text = dgvProductos.CurrentCell.Value.ToString();
```

Eliminar filas:

Utilizaremos para ello la variable **index** previamente asignada en el evento CellClick:

```
private void btnEliminar_Click(object sender, EventArgs e)
{
    //al hacer click para seleccionar la fila a eliminar
    //se asigna el indice de la misma a la variable index

    if (index != -1)
    {
        //eliminamos la fila seleccionada
        dgvProductos.Rows.RemoveAt(index);
    }
}
```

6. Ejercicio práctico (no entregable): Modificar el programa "Lista de alumnos" para que, en vez de mostrar la lista en un listBox, lo haga en un DataGridView.

- ✓ *El nombre en una columna, el apellido en otra y su condición de Libre o Regular, en otra.*
- ✓ *El programa deberá permitir insertar registros en la grilla.*
- ✓ *El programa deberá permitir modificar dichos registros pero no desde la grilla, la misma deberá tener su propiedad ReadOnly en True y no deberá tener la opción de agregar nuevo registro.*
- ✓ *El programa deberá permitir eliminar registros de la grilla.*
- ✓ *Cada Modificación y Eliminación de registros deberá consultarse con un MessageBox con DialogResult.*