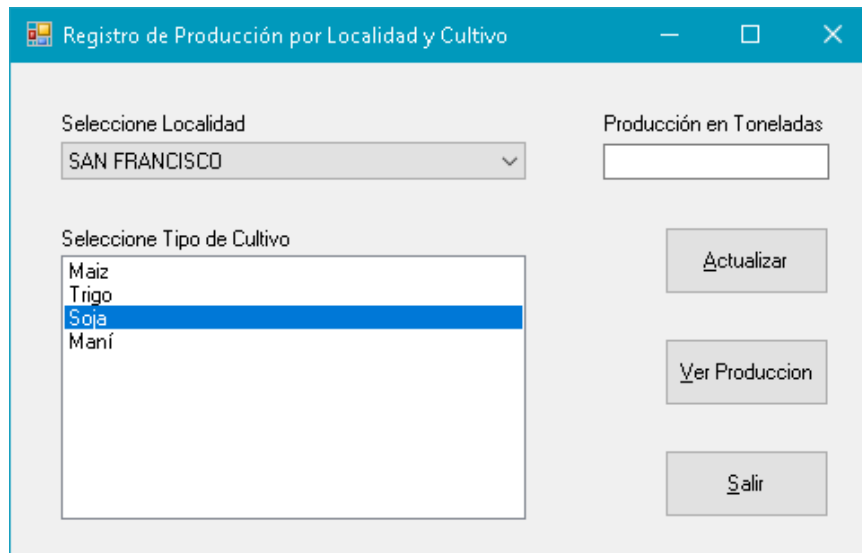


Situación profesional 4: Agricultura

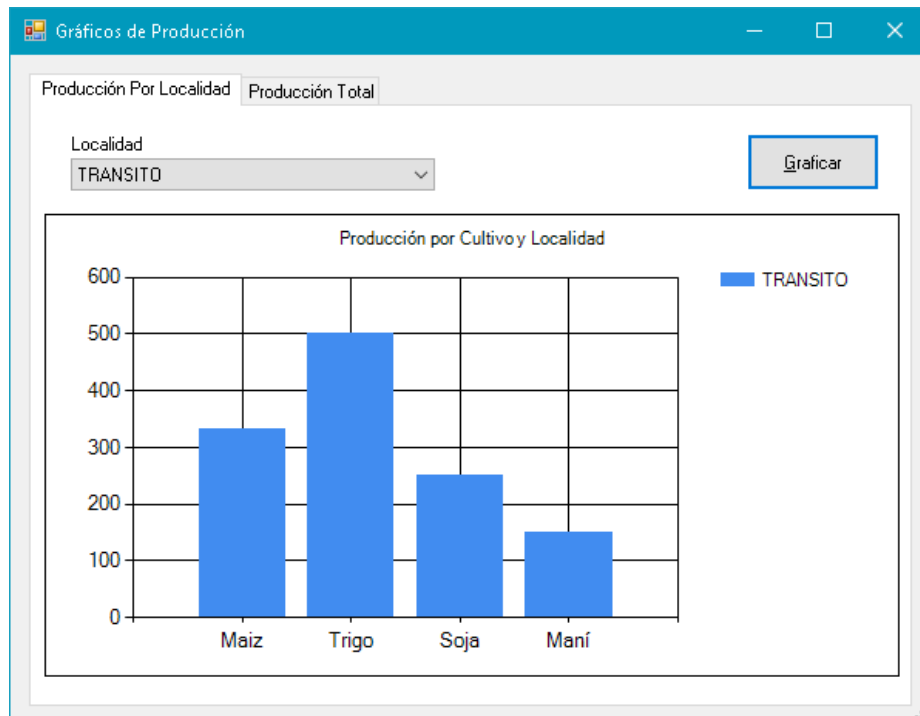
Un funcionario de la Secretaría de Agricultura de la provincia de Córdoba, le ha solicitado a usted, quien realiza en esa dependencia una pasantía, el desarrollo de una aplicación para registrar y graficar la producción en toneladas de los distintos tipos de cultivos de las localidades de la provincia.

Para registrar la producción en toneladas de una localidad y de un tipo de cultivo, el usuario debe poder seleccionar el nombre de una localidad de un cuadro combinado, seleccionar el nombre del cultivo de un cuadro combinado, ingresar las toneladas en una caja de texto y, al pulsar un botón de comando, los datos deberían actualizarse.

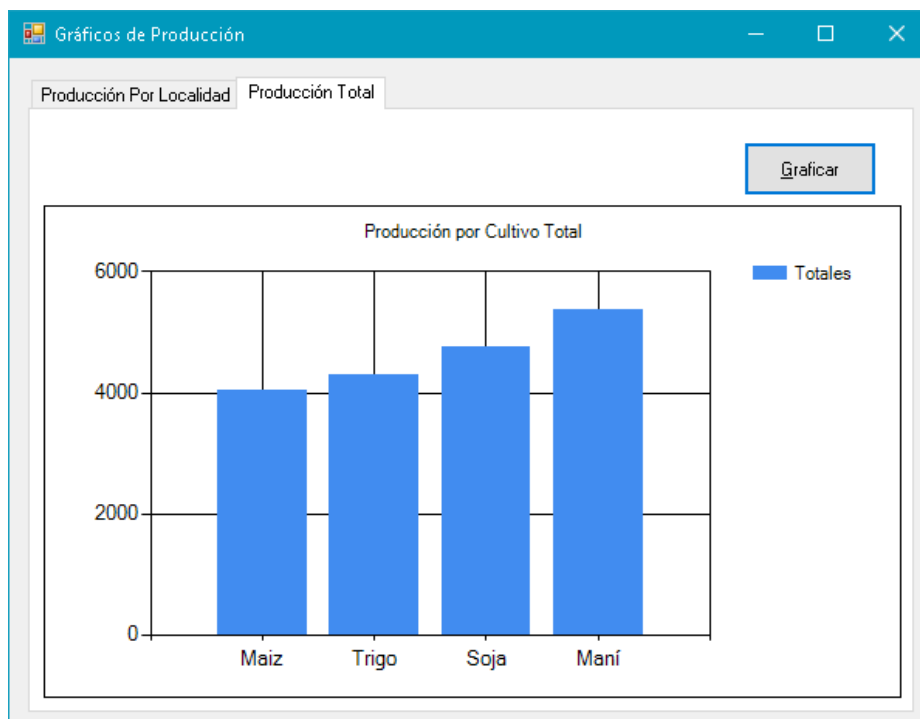
Si al seleccionar una localidad y tipo de cultivo, las toneladas ya fueron ingresadas, el dato se muestra en la caja de texto para poder ser modificado y actualizado al pulsar el botón de comando correspondiente. Si para esa localidad y cultivo no existe todavía ningún registro de producción entonces la caja de texto quedará vacía, en es caso el usuario podrá ingresar un valor y con el botón Actualizar se grabará un nuevo registro.

La imagen muestra una ventana de aplicación con el título "Registro de Producción por Localidad y Cultivo". Dentro de la ventana, hay dos cuadros combinados: "Seleccione Localidad" con "SAN FRANCISCO" seleccionado, y "Seleccione Tipo de Cultivo" con "Soja" seleccionado. A la derecha de estos, hay un campo de texto para "Producción en Toneladas". Debajo de los cuadros combinados, hay tres botones: "Actualizar", "Ver Produccion" y "Salir".

La interfaz cuenta con un botón de comando que, al ser pulsado, abre otro formulario que presentará un control TabControl para mostrar 2 gráficos estadísticos, su primera solapa del control Tab se denomina "Producción Por Localidad", y mostrará en un control Chart la producción en toneladas de cada localidad y por tipo de cultivo, la localidad se selecciona de un control ComboBox. Las filas del gráfico representan a los cultivos, las columnas del gráfico representan la producción en toneladas de la localidad seleccionada, y la referencia del gráfico muestra el nombre de la localidad.



La segunda solapa del control Tab, lleva por título “Producción Total” y debe mostrar el gráfico de columnas con los totales de producción para cada cultivo:



Cada solapa del control Tab contiene un control Chart particular.

La base de datos contiene tres tablas: Localidades, Cultivos y Producción.

Los datos de la tabla "localidades" son: un número para identificar a la localidad y el nombre de la localidad. La clave principal de la tabla es el número de localidad.

Localidades	
Nombre del campo	Tipo de datos
localidad	Número
nombre	Texto corto

Los datos de la tabla "cultivos" son: un número para identificar al cultivo y el nombre del cultivo. La clave principal de la tabla es el número de cultivo.

Cultivos	
Nombre del campo	Tipo de datos
Cultivo	Número
Nombre	Texto corto

Los datos de la tabla "producción" son: un número para identificar a la localidad, un número para identificar al cultivo y un número para la cantidad de toneladas. La clave principal de la tabla está compuesta por las columnas número de localidad y número de cultivo.

Produccion	
Nombre del campo	Tipo de datos
localidad	Número
Cultivo	Número
Produccion	Número

Las tablas de localidades y cultivos deberán cargarse manualmente con cierta cantidad de registros previo a ejecutar la aplicación. La tabla de producción debe estar inicialmente vacía.

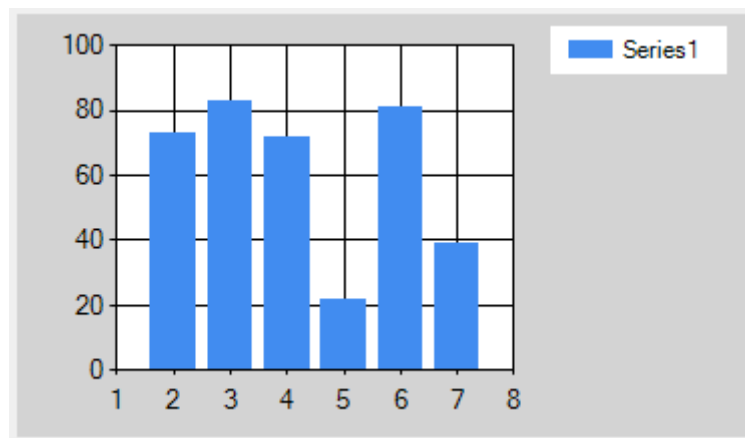
SP4/H1: Control MS Chart

Para resolver lo que solicita Agricultura Córdoba, debemos desarrollar una aplicación que permita generar gráficos estadísticos.

El control que nos permite realizar esto se llama **Chart**, y sirve para trazar datos en gráficos de acuerdo con sus especificaciones. Es posible crear un gráfico al establecer datos en la página de propiedades del control o al recuperar datos para trazarlos desde otro origen como una base de datos de distintos formatos (Excel, Access, etc.). En la situación planteada la información está registrada en una base de datos de Access como origen de datos.

La definición de la clase Chart está establecida en este espacio de nombre:

System.Windows.Forms.DataVisualization.Charting.Chart

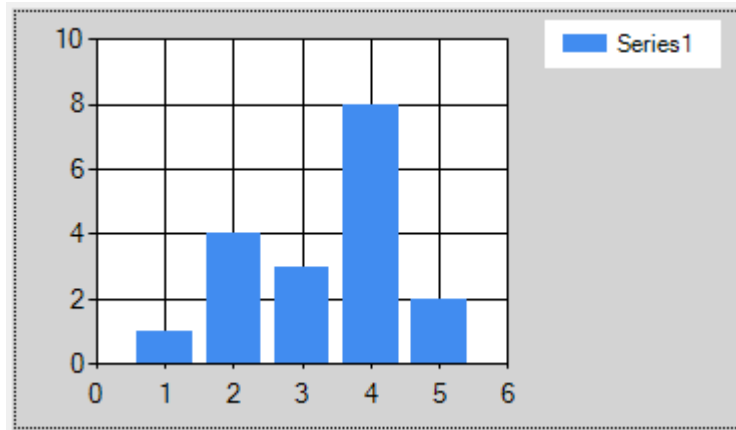


Trazar datos con arreglos y la propiedad Series

La forma más sencilla de trazar un gráfico consiste en crear un arreglo de valores numéricos y, a continuación, generar puntos de datos con cada valor del arreglo, los puntos formarán parte de una serie de datos en el control Chart, como se muestra en el siguiente ejemplo:

```
int[] valores = new int[] {1,4,3,8,2 };  
for(int i=0; i<valores.Length; i++)  
{  
    chart1.Series[0].Points.Add(valores[i]);  
}
```

Obtendremos este gráfico de columnas:



Los componentes principales de un control Chart

Vamos a tomar un momento para analizar las piezas más importantes de un gráfico y explorar algunos de los tecnicismos. El control Chart hace una sola **Imagen Gráfica**. La Imagen Gráfica puede estar compuesto por múltiples gráficos, por ejemplo, un gráfico de líneas y un gráfico de barras.

Cada gráfico dentro de la Imagen Gráfica se conoce como un área de gráfico (ChartAreas). Normalmente, el control Chart sólo tendrá un área de gráfico.

Un cuadro contiene una o más series, que se asocian con un área de gráfico particular. Una serie es una colección de puntos de datos. El modo de representar la serie depende de su tipo. Una serie configurada para mostrar como una línea hará que sus puntos de datos formen una línea continua. Para tener varias líneas en el gráfico se deberá definir una serie para cada línea. Los puntos de datos que componen una serie suelen tener dos componentes: un valor X y un valor de Y. Aunque algunos tipos de series solo necesitan un solo punto de datos.

La línea y columna de la serie en el valor X indica la posición del punto de datos a lo largo del eje del área del gráfico de X y el valor de Y indica la posición de la línea o la altura de la columna a lo largo del eje del área del gráfico de Y.

Especificación de datos de la gráfica

El área de gráfico, las series, y los puntos de datos se pueden especificar de forma declarativa (ya sea por entrar en el marcado declarativo a mano o a través de la ventana Propiedades en modo gráfico) o mediante programación en la clase de código subyacente.

Normalmente, las series y el área del gráfico se especifican de forma declarativa y los puntos de datos se llenan mediante programación, a partir de una consulta a una base de datos o alguna otra base de datos dinámica. De esa forma el formato general del gráfico queda fijo y el

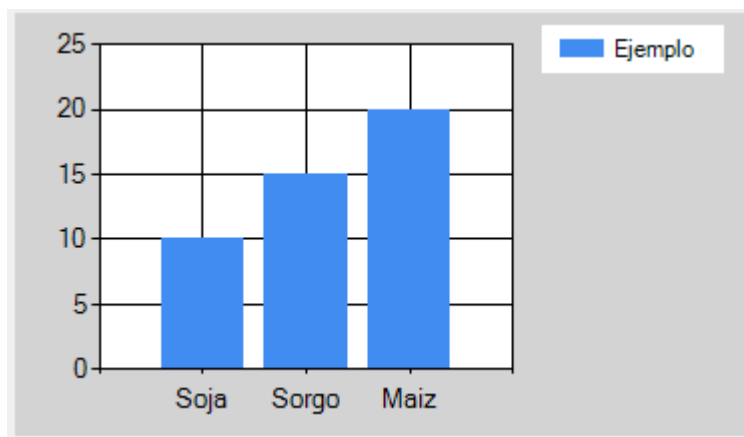
contenido del mismo se actualiza dinámicamente de acuerdo a los valores obtenidos de la base de datos que seguramente irán cambiando en el tiempo.

Hemos realizado un ejemplo donde se muestra un gráfico cuyos gráficos de áreas, series, y los datos de todos los puntos se especifican de forma declarativa.

Probemos el siguiente código en el evento LOAD de nuestro formulario de ejemplo:

```
chart1.Series.Clear(); // se limpia la colección de Series
chart1.Series.Add("Ejemplo"); // se agrega una nueva serie
chart1.Series[0].Points.Add(10); // se agrega un punto a la serie
chart1.Series[0].Points[0].AxisLabel = "Soja"; // se agrega una etiqueta al punto
chart1.Series[0].Points.Add(15);
chart1.Series[0].Points[1].AxisLabel = "Sorgo";
chart1.Series[0].Points.Add(20);
chart1.Series[0].Points[2].AxisLabel = "Maiz";
```

El resultado es este:



Resulta similar al ejemplo anterior, pero acá tenemos más información al haber agregado una etiqueta a cada punto que permite identificar de mejor manera su ubicación en el gráfico.

Observe que la asignación de las etiquetas se realiza considerando la posición que ocupa el punto dentro de la serie.

Tenga en cuenta que el control Chart tiene una sección "Series" y una sección "ChartAreas", que definen la serie y las áreas de gráfico, respectivamente. La sección "ChartAreas" define un único ChartArea denominado **MainChartArea**.

La sección "Series" define una serie única denominada "Ejemplo". Esta serie está configurada para hacer que una columna se muestre en la MainChartArea. A continuación, sus puntos de datos se definen a través de la colección "Points". Hay tres puntos de datos, cada punto de datos

muestra un nombre a través de la propiedad “**AxisLabel**” sobre el eje X y muestra el valor numérico sobre el eje Y, valores que se almacenan en la propiedad “**YValues**”.

Enlace de datos a Base de datos

En general cuando necesitamos graficar cierta información que tenemos almacenada en una base de datos, tenemos dos alternativas, una es recorrer la tabla de la base registro por registro y tomar sus valores para agregar nuevos puntos de datos a la serie, se resuelve fácilmente con un ciclo repetitivo sobre los registros de la tabla, por ejemplo:

```
// suponemos que 'tabla' es un objeto de tipo DataTable
// con los campos 'ValorY' y 'ValorX'
chart1.Series.Clear(); // se limpia la colección de Series
chart1.Series.Add("Datos de la Tabla"); // se agrega una nueva serie
int i = 0;
foreach (DataRow dr in tabla.Rows)
{
    // se agrega un nuevo punto de datos a la serie
    chart1.Series[0].Points.Add((double)dr["ValorY"]); // valor para el eje Y
    // al mismo punto se asigna el valor para el eje X (etiqueta)
    chart1.Series[0].Points[i].AxisLabel = dr["ValorX"].ToString();
    i++; // incrementa el índice para referenciar al punto agregado
}
```

Observe que el método “**Add()**” requiere que el parámetro sea de tipo “double” y que el valor asignado a la propiedad “**AxisLabel**” sea de tipo “String”.

Además de añadir puntos específicos para una serie a través del método Add(), el control Chart incluye métodos para la unión de un conjunto de datos de una tabla en tan solo una o dos líneas de código.

Si tenemos una colección de datos con dos propiedades que contienen los valores X e Y, puede utilizar el método **DataBindTable** del control Chart para enlazar la tabla con la representación gráfica de los valores, es importante definir correctamente el dato que se asigna al eje X y el qué se asigna al eje Y.

El siguiente fragmento de código trabaja con el método DataBindTable, vinculando directamente todo el contenido de la tabla para crear el gráfico en el control Chart. El código es el siguiente:

Disponemos de la tabla “Datos” con esta estructura:

Datos		
	Nombre del campo	Tipo de datos
🔑	Produccion	Número
🔑	Cultivo	Texto corto

El primer campo contiene la cantidad de toneladas que se quiere graficar, es el valor que se colocará en el eje Y, eso es importante para que el gráfico se dibuje correctamente. El segundo campo contiene el nombre del cultivo y se usará como etiqueta en el eje X.

El contenido de la tabla es:

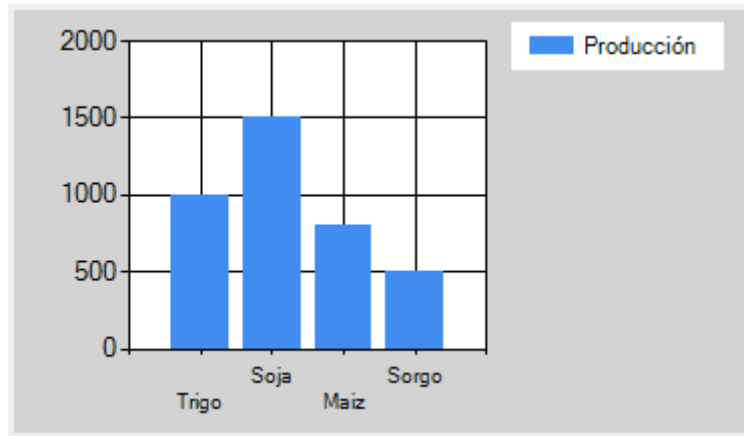
Datos	
Produccion	Cultivo
1000	Trigo
1500	Soja
800	Maiz
500	Sorgo

Si la base de datos no posee una tabla con la estructura necesaria para generar el gráfico, siempre podremos crear una tabla temporal con la estructura de campos correcta y rellenarla con los datos que tenemos en otras tablas de la base de datos. Esa opción es especialmente indicada cuando los datos del gráfico deben ser resultado de aplicar un filtro a la tabla completa para obtener un conjunto reducido de registros que cumplan con la condición del filtro, por ejemplo, a los datos de nuestra tabla le podemos aplicar un filtro con la condición “Producción \geq 1000”, se obtendrán solamente los dos primeros registros para graficar en el control Chart.

El código para crear el gráfico usando el método DataBindTable es este:

```
DataTable tabla = DS.Tables["Datos"];
chart1.Series.Clear(); // se limpia la colección de Series
chart1.DataBindTable(tabla.DefaultView, "Cultivo");
chart1.Series[0].Name = "Producción";
```

Y se obtiene:



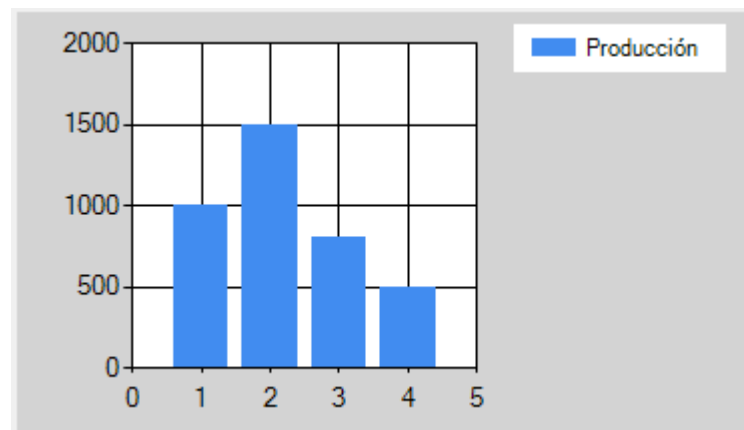
El método “DataBindTable” recibe como primer parámetro un objeto de tipo “DataView”, en este caso la propiedad “DataView” de la tabla que contiene los datos a graficar.

Un “DataView” contiene una representación de la tabla que se puede enlazar (bindeable) a diferentes controles y que admite filtrado, ordenamiento, búsquedas y edición de los datos contenidos en la tabla.

Puede consultar más detalles de la clase “DataView” en este enlace:

<https://learn.microsoft.com/en-us/dotnet/api/system.data.dataview?view=netframework-4.8.1>

El segundo parámetro del método “DataBindTable” es un String que debe contener el nombre del campo que se necesita usar como etiqueta para cada punto en el eje X. Si este segundo parámetro se omite se obtiene el gráfico sin valores en el eje X:



Ahora que sabemos conectar una base de datos con un Chart, veremos que podemos cambiar la forma de presentar la información, por ejemplo: gráfico de puntos, de líneas, de barras, de columnas, de área, circular y muchos tipos más.

El espacio de nombre donde encontraremos las características es:

System.Windows.Forms.DataVisualization.Charting

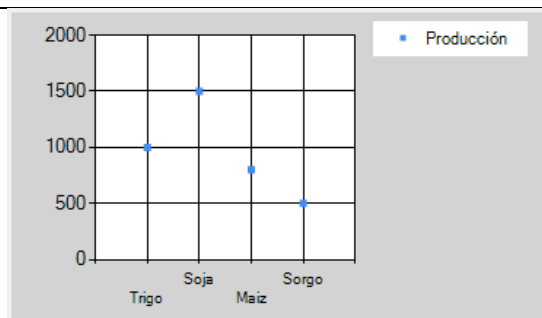
Los tipos de gráficos son:

Nombre de miembro	Descripción
Point	Tipo de gráfico de puntos.
FastPoint	Tipo de gráfico FastPoint.
Bubble	Tipo de gráfico de burbujas.
Line	Tipo de gráfico de líneas.
Spline	Tipo de gráfico de curvas spline.
StepLine	Tipo de gráfico StepLine.
FastLine	Tipo de gráfico FastLine.
Bar	Tipo de gráfico de barras.
StackedBar	Tipo de gráfico de barras apiladas.
StackedBar100	Tipo de gráfico de barras 100% apiladas.
Column	Tipo de gráfico de columnas.
StackedColumn	Tipo de gráfico de columnas apiladas.
StackedColumn100	Tipo de gráfico de columnas 100% apiladas.
Area	Tipo de gráfico de áreas.
SplineArea	Tipo de gráfico de áreas de spline.
StackedArea	Tipo de gráfico de áreas apiladas.
StackedArea100	Tipo de gráfico de áreas 100% apiladas.
Pie	Tipo de gráfico circular.
Doughnut	Tipo de gráfico de anillos.
Stock	Tipo de gráfico de cotizaciones.
Candlestick	Tipo de gráfico de vela japonesa.
Range	Tipo de gráfico de intervalos.
SplineRange	Tipo de gráfico de intervalos de spline.
RangeBar	Tipo de gráfico RangeBar.

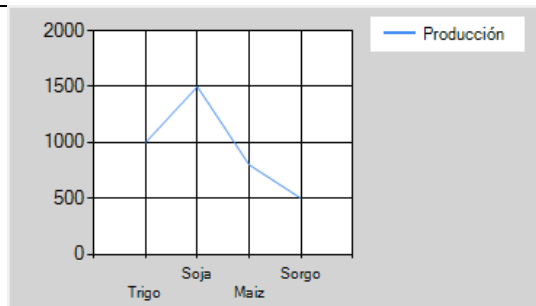
RangeColumn	Tipo de gráfico de columnas de intervalo.
Radar	Tipo de gráfico radial.
Polar	Tipo de gráfico polar.
ErrorBar	Tipo de gráfico de barras de error.
BoxPlot	Tipo de gráfico de diagrama de caja.
Renko	Tipo de gráfico Renko.
ThreeLineBreak	Tipo de gráfico ThreeLineBreak.
Kagi	Tipo de gráfico Kagi.
PointAndFigure	Tipo de gráfico PointAndFigure.
Funnel	Tipo de gráfico de embudo.
Pyramid	Tipo de gráfico piramidal.

Ejemplos de tipos de gráficos obtenidos con diferentes valores en la propiedad “ChartType”:

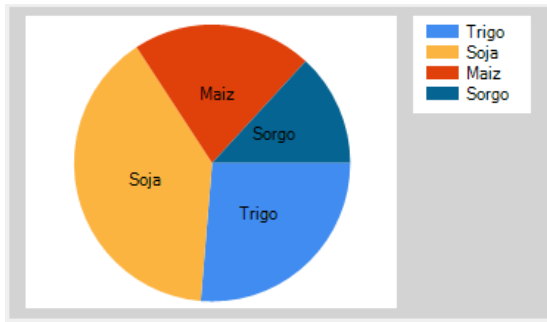
```
chart1.Series[0].ChartType = SeriesChartType.Point;
```



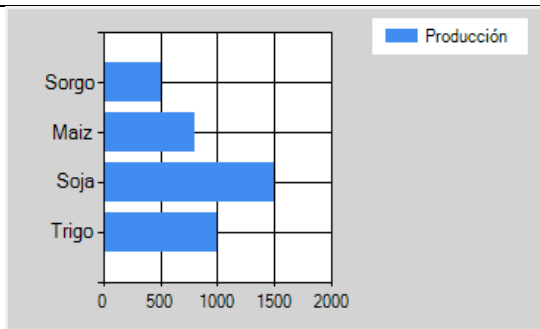
```
chart1.Series[0].ChartType = SeriesChartType.Line;
```



```
chart1.Series[0].ChartType = SeriesChartType.Pie;
```



```
chart1.Series[0].ChartType = SeriesChartType.Bar;
```



Quedan muchas propiedades y funcionalidades del control Chart para profundizar, puede consultarlas en este enlace:

<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8.1>

Recomendamos investigar principalmente sobre estas propiedades:

- ChartAreas
- Series
- Legends
- Titles

1. Indique la opción correcta

En un gráfico, una serie es un conjunto de puntos de datos relacionados.

Verdadero X

Falso

2. Indique la opción correcta

Solo se pueden crear gráficos de una sola serie.

Verdadero

Falso X

3. Indique la opción correcta

Vea el siguiente código e indique a qué corresponde:

```
COMANDO.Connection = CONECTOR;  
COMANDO.CommandType = CommandType.TableDirect;  
COMANDO.CommandText = "Produccion";  
ADAPTADOR.Fill(TABLA);
```

Conexión a BD

Inicializa variable de columna

Cargar la tabla en el DataSet X

Limpia Chart y agrega serie

4. Indique la opción correcta

Vea el siguiente código e indique a qué corresponde:

```
foreach( DataRow D in TABLA.Rows) {  
    Chart1.Series[0].Points.Add((D["toneladas"]));  
}
```

Conexión a BD

Inicializa variable de columna

Repetitiva para cargar datos X

Limpia Chart y agrega serie

5. Indique la opción correcta

Vea el siguiente código e indique a qué corresponde:

```
Chart1.Series.Clear();  
Chart1.Series.Add("Produccion");
```

Conexión a BD

Inicializa variable de columna

Repetitiva para cargar datos

Limpia Chart y agrega serie X

6. Indique la opción correcta

El método **Series[0].Points.Add()** permite:

Agregar un punto de datos al gráfico X

Agregar un arreglo de puntos en un solo paso.

Agregar una serie.

Ninguna de las anteriores

SP4/H2: Control ProgressBar

El control ProgressBar representa una barra de progreso y nos será de mucha utilidad para informar al usuario de una aplicación, el progreso de alguna actividad que deba llevar a cabo. Por ejemplo, una consulta intensiva en una base de datos.



En nuestra Situación profesional la podemos utilizar para mostrar la barra de progreso antes de presentar el grafico en el control Chart (en este caso la consulta es muy rápida por el poco contenido de datos, pero simbólicamente lo agregaremos para conocer su uso).

Uso de las propiedades Value, Minimum y Maximum para indicar la evolución

Para mostrar el progreso de una operación, la propiedad Value se incrementa continuamente hasta llegar a un máximo (definido por la propiedad Maximum). Por tanto, el número de intervalos que muestra el control será siempre un porcentaje de la propiedad Value, relativo a las propiedades Minimum y Maximum.

Por ejemplo, si el valor de Minimum es 1 y el de Maximum 100, un valor 50 en la propiedad Value hará que se muestre el 50 por ciento de los intervalos, de esta forma:



Establecimiento de la propiedad Maximum a un límite conocido

Para programar el control ProgressBar debe tener un límite que va a alcanzar la propiedad Value. Por ejemplo, si va a transferir un archivo y la aplicación puede determinar su tamaño en kilobytes, puede establecer ese número en la propiedad Maximum. Al transferir el archivo, la aplicación debe tener alguna forma de determinar cuántos kilobytes se han descargado y establecer la propiedad Value a ese valor.

En los casos donde no pueda determinar por adelantado el valor de la propiedad Maximum, podrá usar un control que permita mostrar continuamente una animación.

Uso de la propiedad Step y el método PerformStep

Otra forma de indicar el proceso de una tarea con el control ProgressBar es usando el valor asignado a la propiedad Step, que por defecto es 10, en conjunto con el método PerformStep(),

cada vez que se ejecute este método la propiedad Value será incrementada automáticamente en el valor que tenga la propiedad Step, es decir que para los valores por defecto que presenta el control, Minimum=0, Value=0, Maximum=100 y Step=10, el uso del método PerformStep repetidas veces hará que el progreso se modifique de 10 en 10 desde 0 hasta 100.

Ocultar el control ProgressBar con la propiedad Visible

Normalmente, la barra de progreso no se muestra hasta que empieza la operación y desaparece cuando termina. Puede establecer la propiedad Visible a True para mostrar el control al comienzo de la operación y volver a establecerla a False para ocultarlo cuando termine.

Con esta herramienta podremos dar una indicación de actividad en la aplicación que alerte a los usuarios para esperar un cierto tiempo por el resultado de las operaciones, esto es especialmente importante cuando se trate de operaciones con grandes volúmenes de datos, procesos con muchos archivos, cálculos intensivos, procesos de backups, etc. Sin esta indicación visual el usuario podría pensar que la aplicación se ha bloqueado y no responde.

Más información sobre el control ProgressBar:

<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/progressbar-control-windows-forms?view=netframeworkdesktop-4.8>

1. Indique la opción correcta

La barra de progreso es automática, debemos establecer solamente las propiedades Minimum y Maximum.

Verdadero

Falso X

2. Indique la opción correcta

La propiedad Minimum determina el valor máximo que puede adoptar el progressbar.

Verdadero

Falso X

3. Indique la opción correcta

No es posible utilizar el control ProgressBar si no se conoce cuánto tiempo va a durar la transacción.

Verdadero

Falso X

4. Indique la opción correcta

Normalmente, cuando termina la operación, la barra de progreso desaparece.

Verdadero

Falso X

5. Indique las opciones

¿Cuáles son las propiedades de una barra de progreso?:

☐ Minimum X

☐ MaxVal

☐ Maximum X

☐ Step X

☐ StepPerform

☐ Value X

☐ Valor

☐ PerformStep

6. Indique la opción correcta

La propiedad Step representa el valor en el cual la barra de progreso ira incrementando su valor.

Verdadero X

Falso

7. Indique la opción correcta

¿Cuál es la aplicación posible de la barra de progreso?:

Progreso de una operación X

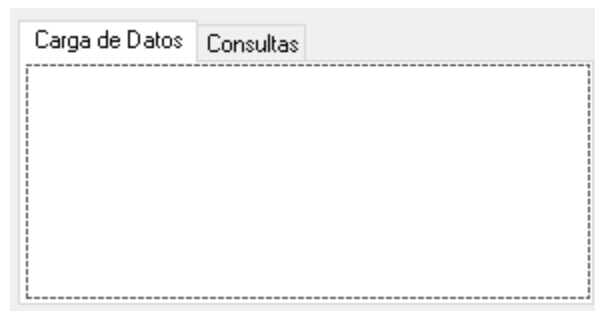
Copiar Archivos

Mostrar un gráfico al usuario

SP4/H3: Control TabControl

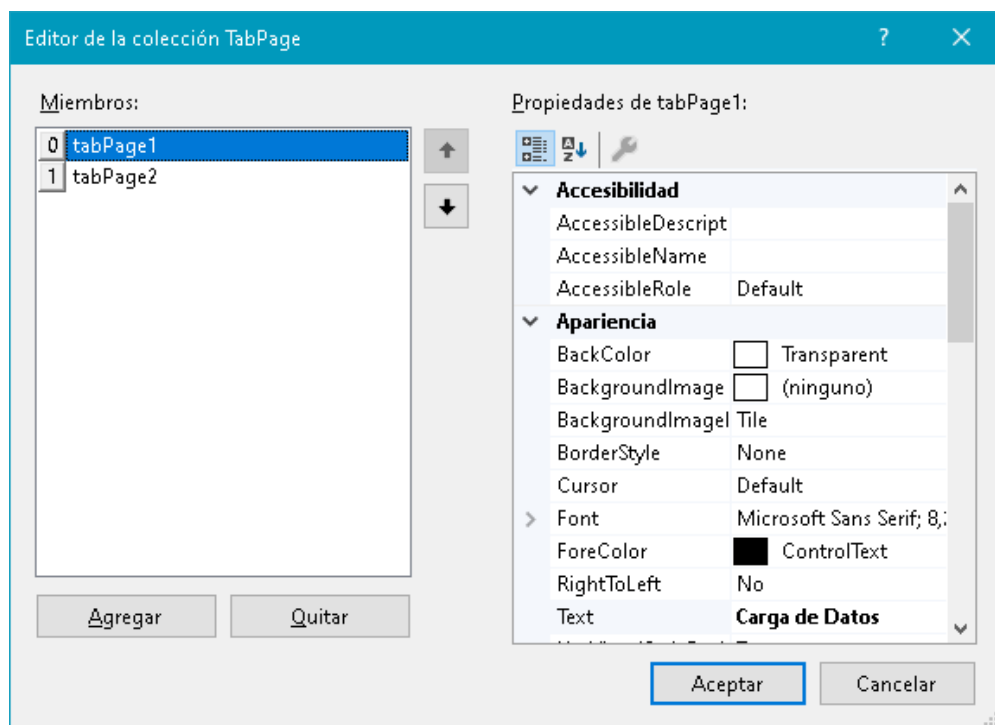
El control TabControl de formularios Windows Forms muestra múltiples fichas, similares a los divisores de un cuaderno o a las etiquetas de las carpetas de un archivador.

Las fichas pueden contener imágenes y otros controles. Puede utilizar el control de fichas para crear cuadros de diálogo con varias páginas como los que suelen aparecer en el sistema operativo Windows, por ejemplo:



La propiedad más importante de TabControl es TabPages, que contiene las fichas individuales. Cada ficha individual es un objeto TabPage.

De forma predeterminada, un control TabControl contiene dos controles TabPage. Puede tener acceso a estas fichas a través de la propiedad TabPages, como también puede agregar nuevas fichas, configurar propiedades de cada una y eliminarlas de ser necesario.



Para Agregar una Ficha mediante Programación

Utilice el método **Add** de la propiedad **TabPages**

```
TabPage NuevaPagina = new TabPage();  
NuevaPagina.Text = "TabPage" + (TabControl1.TabPages.Count + 1).ToString();  
TabControl1.TabPages.Add(NuevaPagina);
```

Para Quitar una Ficha mediante Programación

Utilice el método **Remove** de la propiedad **TabPages**.

Para quitar todas las fichas, utilice el método **Clear** de la propiedad **TabPages**.

Cambiar la Apariencia de un Control TabControl

Puede cambiar la apariencia de las fichas utilizando las propiedades del control **TabControl** y los objetos **TabPage** que constituyen las fichas individuales en el control. Al establecer estas propiedades, podrá mostrar imágenes en las fichas, mostrar fichas en posición vertical en lugar de hacerlo horizontalmente, mostrar varias filas de fichas y habilitar o deshabilitar las mismas mediante programación.

Para Mostrar un icono en la Parte de Etiqueta de una Ficha

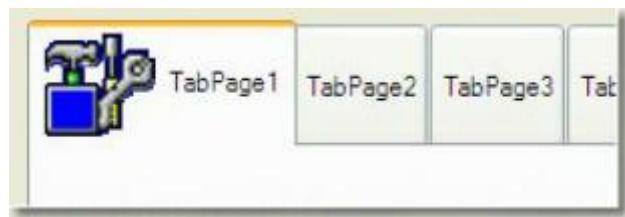
Paso 1 - Agregue un control **ImageList** al formulario.

Paso 2 - Agregue imágenes a la lista de imágenes.

Paso 3 - Establezca la propiedad **ImageList** del **TabControl** en el control **ImageList**.

Paso 4 - Establezca la propiedad **ImageIndex** del objeto **TabPage** en el índice de una imagen de la lista.

Resultado



Para Crear Varias Filas de Fichas

Paso 1 - Agregue el número de páginas de fichas que desea.

Paso 2 - Establezca la propiedad Multiline del control TabControl en true.

Paso 3 - Si las fichas no aparecen ya en varias filas, establezca la propiedad Width del control TabControl para que sea más estrecha que el total de las fichas.

Resultado



Para Organizar Fichas a un Lado del Control

Establezca la propiedad Alignment del control TabControl en Left, Right, Top o Bottom.



Para Mostrar las Fichas con Forma de Botones

Establezca la propiedad Appearance del control TabControl en Buttons o FlatButtons.



Para Ajustar el Alto y Ancho de las Fichas

La propiedad **ItemSize** obtiene o establece el tamaño de las fichas del control. La propiedad **Width** de la propiedad ItemSize permite modificar el ancho de las fichas, si la

propiedad **SizeMode** está establecida en Fixed. La propiedad **Height** permite modificar el alto de las fichas.

Para Habilitar o Deshabilitar Fichas mediante Programación

Establezca la propiedad Enabled del control TabPage en true o false.

```
tabControl1.TabPages[0].Enabled = false;
```

Seleccionar una Ficha

Cuando se hace clic en una ficha, se produce el evento Click correspondiente al objeto TabPage. El usuario puede cambiar el objeto TabPage actual haciendo click en una de las fichas del control, o también mediante programación, utilizando una de las propiedades de TabControl siguientes:

SelectedIndex obtiene o establece el índice de la página de fichas seleccionada actualmente

```
tabControl1.SelectedIndex = 1;
```

SelectedTab obtiene o establece la página de fichas seleccionada actualmente

```
tabControl1.SelectedTab = tabPage1;
```

Puede consultar más propiedades y métodos del control TabControl en este enlace:

<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.tabcontrol?view=netframework-4.8.1>

1. Indique la opción correcta

Para modificar el tamaño de las fichas debe utilizar la propiedad ItemSize del control.

Verdadero X

Falso

2. Indique la opción correcta

¿Para qué se usa el control TabControl?:

contener botones de comando

contener controles de imagen

contener controles e información X

3. Indique la opción correcta

Se puede determinar la ficha seleccionada actualmente utilizando la propiedad SelectedIndex o SelectedTab.

Verdadero X

Falso

4. Indique la opción correcta

Las Fichas solo se pueden organizar en la parte superior del control.

Verdadero

Falso X

5. Indique la opción correcta

Solo se puede seleccionar una ficha realizando un clic sobre el objeto tabPage.

Verdadero X

Falso

6. Indique la opción correcta

Los objetos tabPage solo se pueden crear y eliminar en tiempo de diseño utilizando la propiedad TabPages.

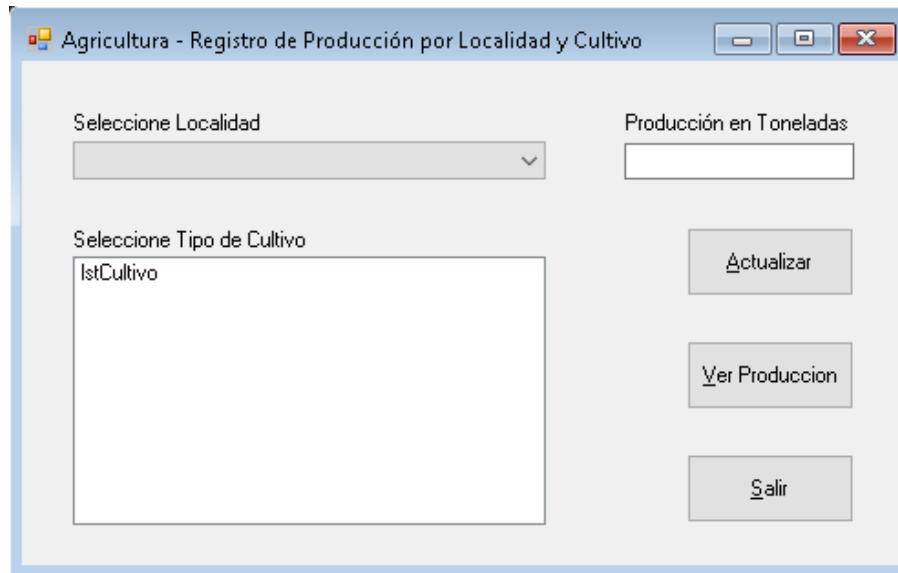
Verdadero X

Falso

SP4/Ejercicio resuelto

Comenzamos el desarrollo de la solución creando un nuevo proyecto con Visual Studio y diseñando los formularios necesarios.

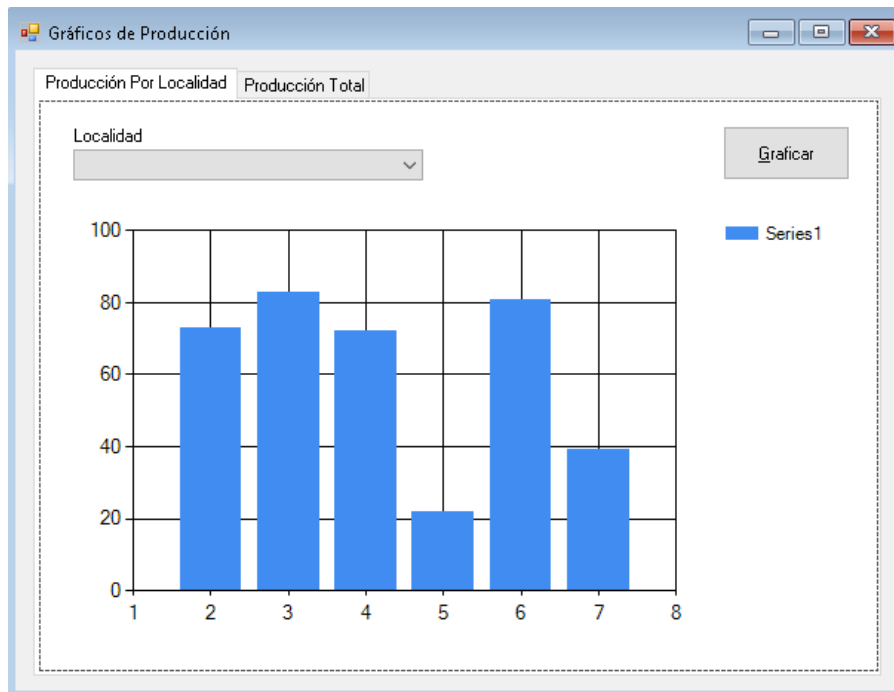
Form1:

The image shows a Windows application window titled "Agricultura - Registro de Producción por Localidad y Cultivo". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains several controls. On the left, there is a label "Seleccione Localidad" above a dropdown menu. Below that is a label "Seleccione Tipo de Cultivo" above a list box. The list box has a header "lstCultivo" and is currently empty. On the right side, there is a label "Producción en Toneladas" above a text box. Below the text box are three buttons stacked vertically: "Actualizar", "Ver Produccion", and "Salir".

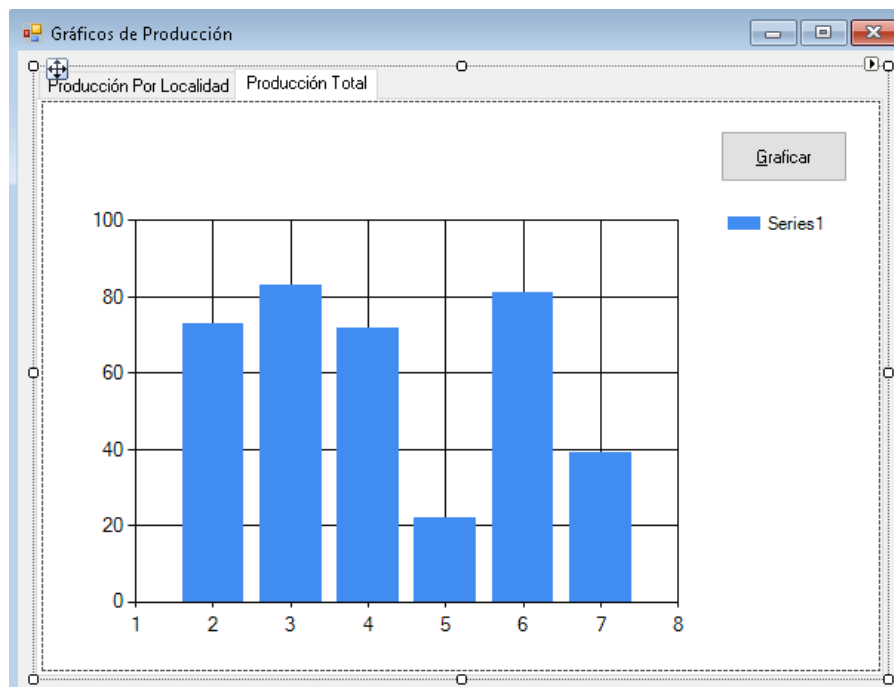
Contiene además de las etiquetas, un control ComboBox (cmbLocalidad), un ListBox (lstCultivo), un TextBox (txtProducción), y 3 controles de tipo Button (btnActualizar, btnVerProduccion y btnSalir).

Form2:

Este formulario contiene un control TabControl con 2 pestañas, en la primera colocamos un control ComboBox cmbLocalidad), un Button (btnGraficar) y un control Chart (chtGrafico)



En la segunda pestaña colocamos un control Button (btnGraficarTotal) y un control Chart (chtGraficoTotal)



Una vez completado el diseño de los formularios y configuradas las propiedades básicas de todos los controles vamos a continuar con la implementación de las clases para el acceso a los datos de la base.

Clase “CLocalidad”:

En esa clase necesitamos acceder a la tabla Localidades, tendrá el método constructor, un método para cargar los nombres de las localidades en un control ComboBox y el método Dispose.

```
CargarLocalidades(System.Windows.Forms.ComboBox)
CLocalidad()
Dispose()
DS
```

Implementación de CLocalidad:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.OleDb;
using System.Windows.Forms;

namespace SP4
{
    public class CLocalidad
    {
        private DataSet DS;

        public CLocalidad()
        {
            try
            {
                DS = new DataSet(); // creación del DataSet
                // conexión con la base de datos
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Agricultura.mdb";
                cnn.Open();

                // Proceso de la tabla Localidades
                OleDbCommand cmdLoc = new OleDbCommand();
                cmdLoc.Connection = cnn;
                cmdLoc.CommandType = CommandType.TableDirect;
                cmdLoc.CommandText = "Localidades";

                OleDbDataAdapter daLoc = new OleDbDataAdapter(cmdLoc);
                daLoc.Fill(DS, "Localidades");
                // se agrega la clave primaria
                DataColumn[] dcL = new DataColumn[1];
                dcL[0] = DS.Tables["Localidades"].Columns["Localidad"];
                DS.Tables["Localidades"].PrimaryKey = dcL;

                cnn.Close();
            }
            catch (Exception ex)
            {
                throw new Exception("CLocalidad " + ex.Message);
            }
        }
    }
}
```

```

public void CargarLocalidades(ComboBox cmb)
{
    // rellena un ComboBox con los nombres de las localidades
    cmb.Items.Clear();
    cmb.DisplayMember = "Nombre";
    cmb.ValueMember = "Localidad";
    cmb.DataSource = DS.Tables["Localidades"];
}

public void Dispose()
{
    DS.Dispose();
}
}

```

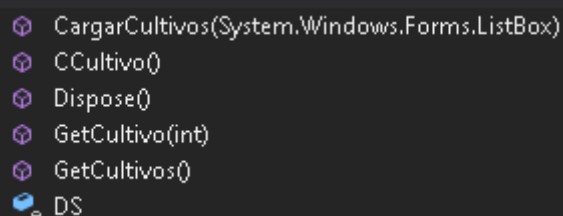
El **constructor** realiza la conexión con la base de datos y obtiene los datos de la tabla Localidades para agregar al DataSet, define además la clave primaria de la tabla.

El método “**CargarLocalidades**” recibe por parámetro un objeto de tipo ComboBox al cuál asignará las propiedades **DisplayMember**, **ValueMember** y **DataSource** para enlazar la tabla de Localidades con el ComboBox.

El método “**Dispose**” libera los recursos del DataSet.

Clase “CCultivo”:

Similar a la anterior clase, “CCultivo” trabajará con la tabla Cultivos y pose el método constructor, un método para cargar un control ListBox con los nombres de los cultivos, un método para obtener la tabla completa, un método para obtener el nombre de un determinado cultivo y el método Dispose.



```

CargarCultivos(System.Windows.Forms.ListBox)
CCultivo()
Dispose()
GetCultivo(int)
GetCultivos()
DS

```

Implementación de CCultivo:

```

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.OleDb;
using System.Windows.Forms;

namespace SP4
{
    public class CCultivo
    {
        private DataSet DS;

        public CCultivo()
        {
            try
            {
                DS = new DataSet(); // creación del DataSet
                // conexión con la base de datos
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Agricultura.mdb";
                cnn.Open();

                // Proceso de la tabla Localidades
                OleDbCommand cmdC = new OleDbCommand();
                cmdC.Connection = cnn;
                cmdC.CommandType = CommandType.TableDirect;
                cmdC.CommandText = "Cultivos";

                OleDbDataAdapter daC = new OleDbDataAdapter(cmdC);
                daC.Fill(DS, "Cultivos");
                // se agrega la clave primaria
                DataColumn[] dcC = new DataColumn[1];
                dcC[0] = DS.Tables["Cultivos"].Columns["Cultivo"];
                DS.Tables["Cultivos"].PrimaryKey = dcC;

                cnn.Close();
            }
            catch (Exception ex)
            {
                throw new Exception("CCultivo " + ex.Message);
            }
        }

        public void CargarCultivos(ListBox lst)
        {
            // rellena un ListBox con los nombres de los cultivos
            lst.Items.Clear();
            lst.DisplayMember = "Nombre";
            lst.ValueMember = "Cultivo";
            lst.DataSource = DS.Tables["Cultivos"];
        }

        public DataTable GetCultivos()
        {
            // devuelve la tabla completa de Cultivos
            return DS.Tables["Cultivos"];
        }

        public String GetCultivo(int cultivo)
        {
            // devuelve el nombre de un cultivo, si su número existe

```

```

        String Nombre = ""; // si no existe, devuelve vacío
        DataRow dr = DS.Tables["Cultivos"].Rows.Find(cultivo);
        if(dr != null)
        {
            Nombre = dr["Nombre"].ToString();
        }
        return Nombre;
    }

    public void Dispose()
    {
        DS.Dispose();
    }
}

```

El **constructor** realiza la conexión con la base de datos y obtiene los datos de la tabla Cultivos para agregar al DataSet, define además la clave primaria de la tabla.

El método “**CargarCultivos**” recibe por parámetro un objeto de tipo ListBox al cuál enlazará la tabla de Cultivos para visualizar los nombres de todos los cultivos que tenga la tabla.

El método “**GetCultivos**” devuelve la tabla obtenida por el constructor, es decir la tabla completa.

El método “**GetCultivo**” recibe por parámetro el número de un cultivo, Hace uso del método “Find” para buscar en la tabla un cultivo con ese número, si obtiene el registro buscado devuelve el nombre del cultivo, en caso contrario devuelve un string vacío.

El método “**Dispose**” libera los recursos del DataSet.

Clase “CProduccion”

Esta clase necesita algo más de código ya que se usará en los dos formularios de la aplicación, para el primero se requiere un método que permita actualizar los datos de producción para cada localidad y cultivo y para el segundo formulario de necesita poder crear los gráficos de producción por localidad y de producción total. Tenemos entonces el método constructor, un método para buscar si hay registro de producción para determinada localidad y cultivo, un

método para actualizar la producción, un método para graficar por localidad y cultivo, un método para graficar la producción total y el método Dispose.

```
Actualizar(int, int, long)
BuscarProduccion(int, int)
CProduccion()
Dispose()
GraficarPorLocaldad(int, string, System.Windows.Forms.DataVisualization.Charting.Chart)
GraficarTotal(System.Windows.Forms.DataVisualization.Charting.Chart)
DAP
DS
```

Implementación de la clase CProduccion:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.OleDb;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace SP4
{
    public class CProduccion
    {
        private DataSet DS;
        private OleDbDataAdapter DAP;
        public CProduccion()
        {
            try
            {
                DS = new DataSet(); // creación del DataSet
                // conexión con la base de datos
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Agricultura.mdb";
                cnn.Open();

                // tabla de Producción
                OleDbCommand cmdPro = new OleDbCommand();
                cmdPro.Connection = cnn;
                cmdPro.CommandType = CommandType.TableDirect;
                cmdPro.CommandText = "Produccion";
                // este dataAdapter está declarado como privado, acá solamente se
instancia
                DAP = new OleDbDataAdapter(cmdPro);
                DAP.Fill(DS, "Produccion");
                // se agrega la clave primaria
                DataColumn[] dcP = new DataColumn[2];
                dcP[0] = DS.Tables["Produccion"].Columns["Localidad"];
                dcP[1] = DS.Tables["Produccion"].Columns["Cultivo"];
                DS.Tables["Produccion"].PrimaryKey = dcP;
            }
            catch { }
        }
    }
}
```

```

        // agregar el objeto commandBulider para poder hacer cambios en la
tabla
        OleDbCommandBuilder cb = new OleDbCommandBuilder(DAP);

        cnn.Close();
    }
    catch (Exception ex)
    {
        throw new Exception("CProduccion " + ex.Message);
    }
}

public long BuscarProduccion(int localidad, int cultivo)
{
    // devuelve el valor del campo producción si existe para
    // el número de localidad y cultivo pasados por parámetro
    long produccion = 0; // si no existe devuelve cero

    Object[] valores = new Object[2];
    valores[0] = localidad;
    valores[1] = cultivo;
    // buscar el registro en producción
    DataRow prod = DS.Tables["Produccion"].Rows.Find(valores);
    if (prod != null)
    {
        // si existe obtener el valor de producción
        produccion = (int)prod["produccion"];
    }

    return produccion;
}

public void Actualizar(int localidad, int cultivo, long produccion)
{
    // actualiza el valor de producción para una localidad y cultivo
    // si el registro ya existe lo actualiza, si no existe agrega
    // un registro nuevo

    Object[] valores = new Object[2];
    valores[0] = localidad;
    valores[1] = cultivo;
    // buscar el registro en producción
    DataRow prod = DS.Tables["Produccion"].Rows.Find(valores);
    if (prod != null)
    {
        // ya existe el registro, se deberá editar
        prod.BeginEdit();
        prod["Produccion"] = produccion; // se actualiza
        prod.EndEdit();
    }
    else
    {
        // no existe el registro, se debe agregar uno nuevo
        DataRow nuevo = DS.Tables["Produccion"].NewRow();
        nuevo["Localidad"] = localidad;
        nuevo["Cultivo"] = cultivo;
        nuevo["Produccion"] = produccion;
        DS.Tables["Produccion"].Rows.Add(nuevo);
    }
    DAP.Update(DS, "Produccion"); // actualiza los cambios en la tabla física
}

public void GraficarPorLocalidad(int localidad, String Nombre, Chart cht)

```

```

{
    // crea un gráfico para una determinada localidad
    cht.Series.Clear();
    // crear una tabla temporal
    DataTable prod = new DataTable();
    prod.Columns.Add("Produccion");
    prod.Columns.Add("Nombre");
    //
    CCultivo cultivo = new CCultivo(); // datos de los cultivos

    foreach (DataRow dr in DS.Tables["Produccion"].Rows)
    {
        // si la producción es de la localidad consultada
        if (localidad == (int)dr["Localidad"])
        {
            String NombreCultivo = cultivo.GetCultivo((int)dr["Cultivo"]);
            // se agrega el registro nuevo a la tabla temporal
            DataRow nuevo = prod.NewRow();
            nuevo["Produccion"] = (int)dr["Produccion"];
            nuevo["Nombre"] = NombreCultivo;
            prod.Rows.Add(nuevo);
        }
    }
    // enlazar la tabla temporal al control Chart
    cht.DataBindTable(prod.DefaultView, "Nombre");
    cht.Series[0].ChartType = SeriesChartType.Column; // tipo de gráfico
    cht.Series[0].Name = Nombre; // nombre de la serie
    cultivo.Dispose();
}

public void GraficarTotal(Chart cht)
{
    cht.Series.Clear();
    // crear una tabla temporal
    DataTable prod = new DataTable();
    prod.Columns.Add("Produccion");
    prod.Columns.Add("Nombre");
    //
    CCultivo cultivo = new CCultivo();
    DataTable cultivos = cultivo.GetCultivos();
    // recorrer todos los cultivos existentes
    foreach (DataRow cult in cultivos.Rows)
    {
        long prod_total = 0; // para cada cultivo acumula la producción
        // recorre la tabla de producción
        foreach (DataRow pro in DS.Tables["Produccion"].Rows)
        {
            if ((int)cult["Cultivo"] == (int)pro["Cultivo"])
            {
                // si es el mismo cultivo, acumula
                prod_total += (int)pro["Produccion"];
            }
        }
        // por cada cultivo se agrega un registro nuevo a la tabla temporal
        DataRow nuevo = prod.NewRow();
        nuevo["Produccion"] = prod_total; // cantidad total de producción
        nuevo["Nombre"] = cult["Nombre"].ToString(); // nombre del cultivo
        prod.Rows.Add(nuevo);
    }
    // enlazar la tabla temporal al control Chart
    cht.DataBindTable(prod.DefaultView, "Nombre");
    cht.Series[0].ChartType = SeriesChartType.Column; // tipo de gráfico
}

```



```

        cht.Series[0].Name = "Totales"; // nombre de la serie
        cultivo.Dispose();
    }

    public void Dispose()
    {
        DS.Dispose();
    }
}

```

El **constructor** realiza la conexión con la base de datos y obtiene los datos de la tabla Produccion para agregar al DataSet, define además la clave primaria de la tabla, observe que se trata de una **clave compuesta** por dos campos, también crea el objeto **OleDbCommandBuilder** para poder hacer las actualizaciones sobre la tabla.

El método “**BuscarProduccion**” recibe por parámetro un número de localidad y un número de cultivo, con ese par de valores busca si existe algún registro con esos mismos valores y si lo encuentra devuelve el valor de producción registrado, en caso de no existir el registro buscado devuelve el valor cero. Este método ayudará a colocar en el control TextBox de producción ese valor cada vez que el usuario modifique o la localidad del ComboBox o el cultivo seleccionado en el control ListBox.

El método “**Actualizar**” recibe por parámetro un número de localidad, un número de cultivo y el valor de producción, usará los valores de la localidad y el cultivo para determinar si en la tabla Produccion existe o no un registro con esos valores, si el registro ya existe se modificará la producción con el nuevo valor y si no existe ningún registro para esa localidad y cultivo se agregará uno nuevo con los mismos valores de los parámetros recibidos.

El método “**GraficarPorLocalidad**” recibe por parámetro el número de la localidad, el nombre de la localidad y objeto Chart sobre el que se desea graficar.

Como los datos para el gráfico no están en ninguna tabla de la base de datos con la estructura que se necesita, vamos a crear una tabla temporal:

```

// crear una tabla temporal
DataTable prod = new DataTable();
prod.Columns.Add("Produccion");
prod.Columns.Add("Nombre");

```

En esa tabla se definen dos columnas, corresponden a los datos que se quieren graficar, en este caso un valor de producción y un nombre de cultivo. La tabla temporal entonces se usará para enlazar con el control Chart y generar así el gráfico deseado.

A continuación, agregamos un objeto de tipo CCultivo ya que será necesario conocer sus nombres:

```
CCultivo cultivo = new CCultivo(); // datos de los cultivos
```

Seguidamente debemos realizar un recorrido de los registros que tenga la tabla Produccion para ir procesando registro por registro y para cada uno de ellos determinar si corresponde a la localidad consultada, el recorrido de los registros se implementa con la estructura “foreach”:

```
foreach (DataRow dr in DS.Tables["Produccion"].Rows)
```

En el interior del ciclo “foreach” debemos comparar si la localidad consultada, es el valor del parámetro “localidad” es igual al valor del campo “localidad” del objeto DataRow “dr” usado para recorrer Produccion, si la condición es verdadera entonces los datos de ese registro los debemos agregar a la tabla temporal:

```
if (localidad == (int)dr["Localidad"])
{
    String NombreCultivo = cultivo.GetCultivo((int)dr["Cultivo"]);
    // se agrega el registro nuevo a la tabla temporal
    DataRow nuevo = prod.NewRow();
    nuevo["Produccion"] = (int)dr["Produccion"];
    nuevo["Nombre"] = NombreCultivo;
    prod.Rows.Add(nuevo);
}
```

El nombre del cultivo lo obtenemos con el método “GetCultivo” pasando por parámetro el número de cultivo que tenemos en la tabla de Produccion.

Al finalizar el ciclo “foreach”, es decir cuando se hayan procesado todos los registros de Producción existentes, ya tenemos la tabla temporal completa con todos los datos necesarios para crear el gráfico en el control Chart, resta entonces enlazar esa tabla al control Chart:

```
// enlazar la tabla temporal al control Chart
cht.DataBindTable(prod.DefaultView, "Nombre");
cht.Series[0].ChartType = SeriesChartType.Column; // tipo de gráfico
cht.Series[0].Name = Nombre; // nombre de la serie
```

Como ya vimos el enlace se realiza con el método “DataBindTable” pasando la propiedad “DefaultView” de la tabla temporal como primer parámetro y el nombre del campo que vamos

a usar para las etiquetas del eje X en el gráfico como segundo parámetro, en este caso el campo se llama “Nombre”.

Después de realizarse el enlace el gráfico ya está creado y lo hace en una serie nueva, que será la serie de la posición cero en la colección de series del control, a esa serie debemos definir el tipo de gráfico que deseamos crear, en este caso se usa “SeriesChartType.Column”, y luego asignamos en nombre de la serie, acá es donde usamos el nombre de la localidad que recibimos en el segundo parámetro del este método.

El método “**GraficarTotal**” recibe por parámetro solamente el control Chart para realizar el gráfico. El proceso es similar al método anterior pero ahora debemos obtener el total de producción para cada cultivo, sin importar de qué localidad se trate, como el proceso es para cada cultivo vamos a necesitar recorrer la tabla de producción N veces, una vez para cada cultivo. Lo que significa implementar dos recorridos anidados, el primero recorre los cultivos y el ciclo interno recorre la producción.

Comenzamos creando la tabla temporal:

```
// crear una tabla temporal
DataTable prod = new DataTable();
prod.Columns.Add("Produccion");
prod.Columns.Add("Nombre");
```

Seguidamente creamos el objeto de la clase CCultivo y con el método “GetCultivos” obtenemos la tabla que vamos a recorrer:

```
CCultivo cultivo = new CCultivo();
DataTable cultivos = cultivo.GetCultivos();
```

Como dijimos el primer ciclo repetitivo será para recorrer los registros de los cultivos:

```
foreach (DataRow cult in cultivos.Rows)
```

En cada iteración del ciclo establecemos un acumulador para el total de producción en cero:

```
long prod_total = 0; // para cada cultivo acumula la producción
```

A continuación, se implementa el segundo ciclo repetitivo para recorrer los registros de producción y en él acumular los valores de producción del cultivo que se está procesando:

```
foreach (DataRow pro in DS.Tables["Produccion"].Rows)
{
    if ((int)cult["Cultivo"] == (int)pro["Cultivo"])
```

```

        {
            // si es el mismo cultivo, acumula
            prod_total += (int)pro["Produccion"];
        }
    }
}

```

De esa forma se van obteniendo los totales de producción para cada cultivo, al finalizar el “foreach” interno tenemos el total que vamos a graficar y ya lo podemos agregar a la tabla temporal:

```

// por cada cultivo se agrega un registro nuevo a la tabla temporal
DataRow nuevo = prod.NewRow();
nuevo["Produccion"] = prod_total; // cantidad total de producción
nuevo["Nombre"] = cult["Nombre"].ToString(); // nombre del cultivo
prod.Rows.Add(nuevo);

```

Cuando ya se hayan procesado todos los cultivos, esto es cuando finaliza el primer “foreach” tenemos la tabla temporal completa y lista para enlazar con el control Chart:

```

// enlazar la tabla temporal al control Chart
cht.DataBindTable(prod.DefaultView, "Nombre");
cht.Series[0].ChartType = SeriesChartType.Column; // tipo de gráfico
cht.Series[0].Name = "Totales"; // nombre de la serie

```

Usamos el mismo tipo de gráfico: “SeriesChartType.Column” y el nombre de la serie será en este caso la palabra “Totales”.

Para finalizar con los comentarios sobre la clase “CProduccion” digamos que el método “Dispose” se encarga de liberar los recursos del DataSet usado

Continuamos ahora con el código de los formularios

Implementaciones en el formulario de registro de producción: Form1

```

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.Data.OleDb;
using System.Windows.Forms.DataVisualization.Charting;

namespace SP4
{
    public partial class Form1 : Form

```

```

{
    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        CLocalidad localidad = new CLocalidad();
        localidad.CargarLocalidades(cmbLocalidad);
        localidad.Dispose();
        CCultivo cultivo = new CCultivo();
        cultivo.CargarCultivos(lstCultivo);
        cultivo.Dispose();
    }

    private void btnActualizar_Click(object sender, EventArgs e)
    {
        if (cmbLocalidad.Items.Count > 0 &&
            lstCultivo.Items.Count > 0 &&
            txtProduccion.Text != "")
        {
            try
            {
                int localidad = (int)cmbLocalidad.SelectedValue;
                int cultivo = (int)lstCultivo.SelectedValue;
                long produccion = long.Parse(txtProduccion.Text);
                //
                CProduccion prod = new CProduccion();
                prod.Actualizar(localidad, cultivo, produccion);
                prod.Dispose();
                MessageBox.Show("Registro actualizado", "Actualizar Producción",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }

    private void cmbLocalidad_SelectedIndexChanged(object sender, EventArgs e)
    {
        BuscarProducción();
    }

    private void lstCultivo_SelectedIndexChanged(object sender, EventArgs e)
    {
        BuscarProducción();
    }

    private void BuscarProducción()
    {
        if (cmbLocalidad.Items.Count > 0 && lstCultivo.Items.Count > 0)
        {
            try
            {
                int localidad = (int)cmbLocalidad.SelectedValue;
                int cultivo = (int)lstCultivo.SelectedValue;
                CProduccion prod = new CProduccion();
                long produccion = prod.BuscarProduccion(localidad, cultivo);
                prod.Dispose();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
}

```

```

        if (produccion == 0) // no existe el textBox queda vacío
        {
            txtProduccion.Text = "";
        }
        else // si existe, muestra el valor de producción para ese cultivo
y localidad
        {
            txtProduccion.Text = produccion.ToString();
        }
    }
    catch (Exception ex)
    {
        txtProduccion.Text = "";
        MessageBox.Show("Error: " + ex.Message);
    }
}

private void btnVerProduccion_Click(object sender, EventArgs e)
{
    Form2 frm = new Form2();
    frm.ShowDialog();
}

private void btnSalir_Click(object sender, EventArgs e)
{
    Close();
}
}

```

El evento “**Load**” se encarga de cargar el control ComboBox con los datos de las localidades y el ListBox con los datos de los cultivos, de esa forma el formulario queda listo para ser usado.

Se programaron los eventos “**SelectedIndexChanged**” del ComboBox y del ListBox para que cada vez que el usuario seleccione un elemento en alguno de ellos se actualice el contenido del textBox de producción, en esos eventos se invoca al método “**BuscarProduccion**”.

“**BuscarProduccion**” realiza las validaciones sobre el contenido del comboBox y el ListBox ya que si alguno de ellos no tiene datos entonces no hay nada para buscar y el TextBox quedará vacío. Por el contrario, si ambos poseen datos se tomarán los valores de los ítems seleccionados en cada uno de ellos, una localidad y un cultivo y se buscarán en la tabla de Producción haciendo uso del objeto de la clase “CProduccion” y el método “BuscarProduccion”, de acuerdo al resultado de esa búsqueda se podrá visualizar en el control TextBox el valor de producción almacenado para esa localidad y cultivo o si no existe ningún valor registrado el control TextBox quedará vacío.

En el evento “**Click**” del botón “**btnActualizar**” se validan los datos necesarios para la localidad, el cultivo y la producción y con un objeto de la clase “CProduccion” se ejecuta el método “**Actualizar**”, agregando un registro nuevo o modificando uno ya existente según el caso. También se muestra un mensaje para confirmar la operación al usuario.

En el evento “**Click**” del botón “**btnVerProduccion**” se crea un objeto de la clase “Form2” y se ejecuta el método “ShowDialog” para pasar el control del programa al formulario con los gráficos:

```
Form2 frm = new Form2();  
frm.ShowDialog();
```

Finalmente, en el evento “**Click**” del botón “**btnSalir**” simplemente se ejecuta el método “Close” para cerrar el formulario y la aplicación:

```
private void btnSalir_Click(object sender, EventArgs e)  
{  
    Close();  
}
```

Implementación de código en el formulario de gráficos: Form2

```
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 SP4  
{  
    public partial class Form2 : Form  
    {  
        public Form2()  
        {  
            InitializeComponent();  
        }  
  
        private void Form2_Load(object sender, EventArgs e)  
        {  
            // cargar las localidades en el ComboBox  
            CLocalidad loc = new CLocalidad();  
            loc.CargarLocalidades(cmbLocalidad);  
            loc.Dispose();  
            // seleccionar el primer tab  
            tabGraficos.SelectedTab = tabProduccionPorLocalidad;  
        }  
    }  
}
```

```

        // inicializar los gráficos
        chtGrafico.Titles.Add("Producción por Cultivo y Localidad");
        chtGrafico.Series.Clear();
        chtGraficoTotal.Titles.Add("Producción por Cultivo Total");
        chtGraficoTotal.Series.Clear();
    }

    private void btnGraficar_Click(object sender, EventArgs e)
    {
        CProduccion prod = new CProduccion();
        prod.GraficarPorLocalidad((int)cmbLocalidad.SelectedValue,
        cmbLocalidad.Text, chtGrafico);
        prod.Dispose();
    }

    private void btnGraficarTodo_Click(object sender, EventArgs e)
    {
        CProduccion prod = new CProduccion();
        prod.GraficarTotal(chtGraficoTotal);
        prod.Dispose();
    }
}

```

El código de este formulario es bastante breve, ya que posee pocos componentes.

En el evento “**Load**” se carga el comboBox con los datos de las localidades, se selecciona el primer tab para visualizar la pestaña del gráfico por localidad, seguidamente, se inicializan los controles Chart de ambos gráficos.

En el evento “**Click**” del botón “**btnGraficar**” se crea un objeto de la clase “CProduccion” y se ejecuta el método “GraficarPorLocalidad” pasando como parámetros el número de localidad seleccionada, el nombre de la localidad y el control Chart, de esa forma se generará el gráfico sobre el formulario.

En el evento “**Click**” del botón “**btnGraficarTodo**” también se crea un objeto de la clase “CProduccion” pero en este caso se ejecuta el método “GraficarTotal” pasando como parámetro el control Chart, así obtenemos el gráfico de totales.

Ejecución: ejemplo al ejecutar la aplicación, para la localidad de Arroyito, el cultivo de Maíz registra una producción de 2200 toneladas.

Registro de Producción por Localidad y Cultivo

Seleccione Localidad
ARROYITO

Producción en Toneladas
2200

Seleccione Tipo de Cultivo
Maiz
Trigo
Soja
Maní

Actualizar

Ver Produccion

Salir

Gráfico de producción por localidad: se observan los valores de producción para los cultivos de Maíz, Trigo, Soja y Maní de la localidad de Arroyito.

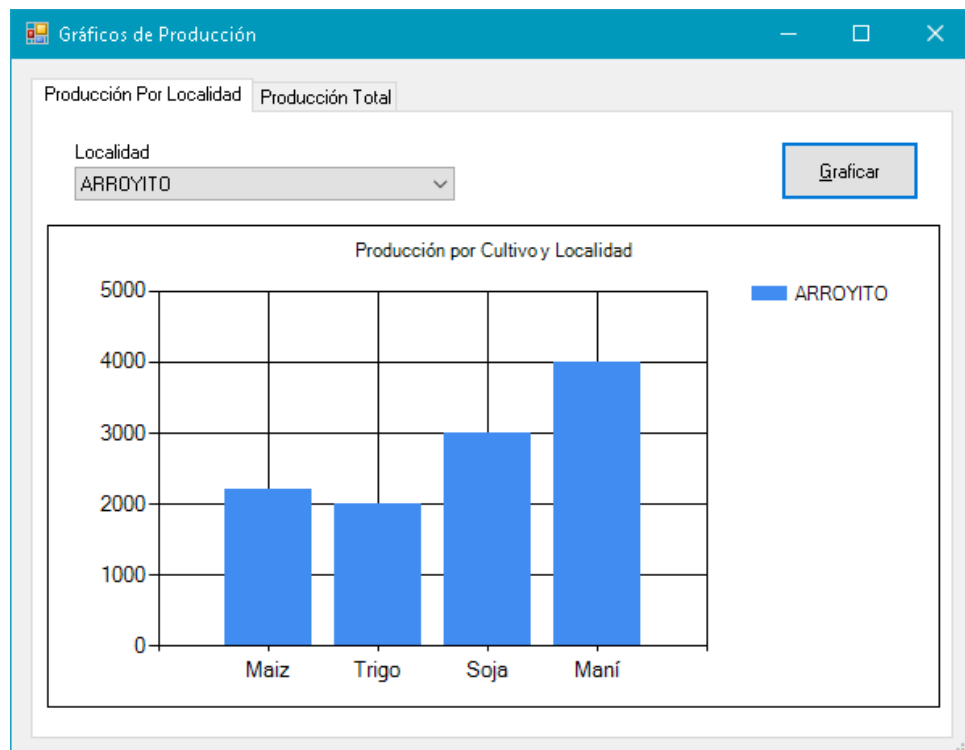
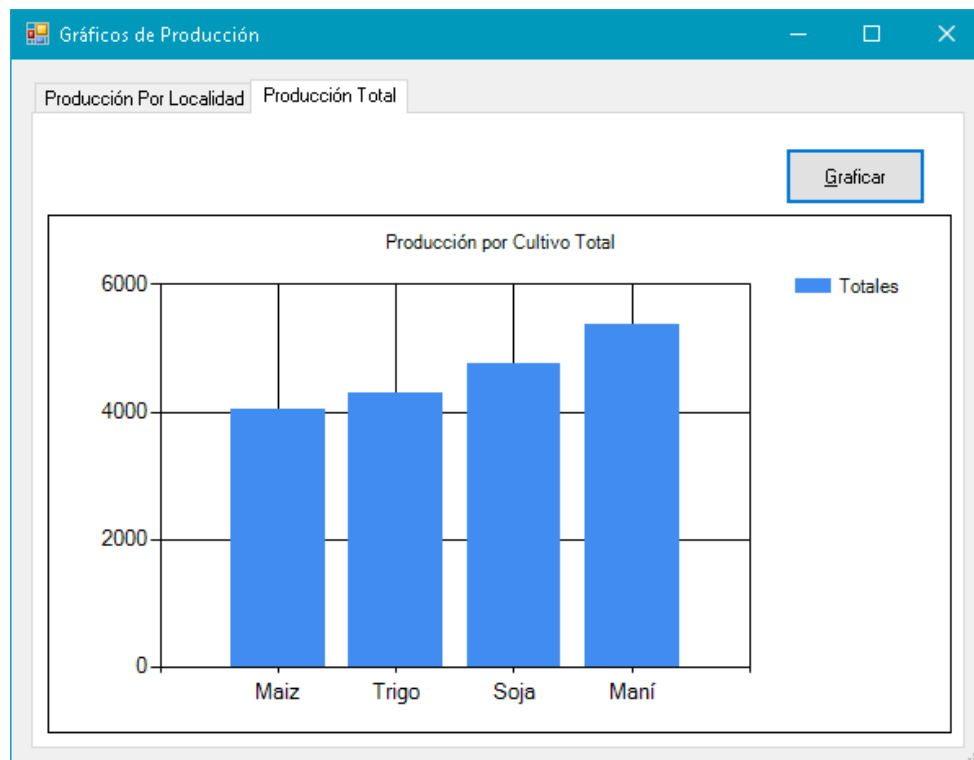


Gráfico de producción total: se observan los valores totales de producción para los cuatro cultivos registrados.



Finalizamos así el desarrollo y pruebas de la aplicación solicitada en la situación profesional.

SP4/Ejercicio por resolver

La empresa de transporte "Cargas Rápidas", dedicada a la logística y distribución de materias primas para empresas del rubro alimenticio, cuenta con una flota de 10 camiones propios para el transporte y distribución de los materiales. Lleva un registro manual de los kilómetros recorridos, el consumo, costo del combustible y gastos por camión, guardando dicha información en una planilla de cálculo.

Debido a que desea llevar un control sobre estos datos, le solicita a usted, estudiante avanzado de la carrera Analista de Sistemas que se encuentra realizando una pasantía en la empresa, que confeccione una aplicación que realice los siguientes gráficos estadísticos:

- Kilómetros por camión: se muestran los kilómetros recorridos por cada camión.
- Kilogramos por camión: por cada uno de los camiones se muestran los kilogramos transportados.
- Total por camión: por cada camión se muestra el importe gastado en concepto de combustible (se calcula multiplicando los litros de combustible consumidos por el precio por litro de combustible).
- Total y Viáticos por camión: por cada camión se muestra el importe gastado en concepto de combustible (se calcula multiplicando los litros de combustible consumidos, por el precio, por litro de combustible) más los viáticos generados.

La planilla de cálculo registra la siguiente información:

	A	B	C	D	E	F	G	H
1	Camión	Kilometros	Litros	Precio	kmperLitros	Total	kg	Viaticos
2	Camión 1	3000	300.00	1.25	10	375.00	22000	330
3	Camión 2	3500	233.33	1.25	15	291.67	35500	300
4	Camión 3	2235	139.69	1.38	16	192.77	18000	250
5	Camión 4	1200	92.31	1.12	13	103.38	48050	280
6	Camión 5	4500	375.00	1.16	12	435.00	18900	500
7	Camión 6	5000	333.33	1.1	15	366.67	70000	400
8	Camión 7	2350	146.88	1	16	146.88	69900	240
9	Camión 8	1879	156.58	1.35	12	211.39	35987	250
10	Camión 9	4500	346.15	1.22	13	422.31	49870	480
11	Camión 10	3900	276.57	1.3	14	362.14	29875	400

La aplicación debe presentar la siguiente interfaz gráfica:



En el control ComboBox titulado “Tipo de Gráfico” deberá incluir las opciones:

- Gráfico de columna
- Gráfico de línea
- Gráfico de barra
- Otra opción a su elección.

El control ProgressBar deberá indicar el tiempo del proceso para generar cada uno de los gráficos solicitados.

1. Indique la opción correcta

El control ProgressBar es utilizado para marcar el progreso en una operación.

Verdadero X

Falso

2. Indique la opción correcta

El control TabControl es utilizado para contener controles y ordenar información.

Verdadero X

Falso

3. Indique la opción correcta

El control ProgressBar, para definir el valor del incremento que se hará de forma predeterminada, debe codificar:

ProgressBar1.PerformStep(1);

ProgressBar1.Step = 1; X

ProgressBar1.PerformStep();

4. Indique la opción correcta

El control Chart se utiliza solamente para graficar información que viene vinculada a una base de datos.

Verdadero

Falso X

5. Indique la opción correcta

El control Chart no se puede utilizar para crear gráficos en formato 3D.

Verdadero

Falso X

6. Indique la opción correcta

El control TabControl, para borrar una pestaña, debe codificar:

```
TabControl1.TabPages.Remove(TabControl1.SelectedTab);
```

 X

```
TabControl1.TabPages.Remove(TabControl1);
```

```
TabControl1.TabPages.Remove(TabControl1.SelectedTab)=1;
```

7. Indique la opción correcta

El control ImageList es un contenedor de imágenes que debemos utilizar asociado a otro control.

Verdadero X

Falso