

Laboratorio de Programación III

Carlos Gonzalez

```
each: function(e, t, n) {
    var r, i = 0,
        o = e.length,
        s = M(e);
    if (n) {
        if (a) {
            for (; o > i; i++)
                if (r = t.apply(e[i], n), r === !1) break;
        } else
            for (i in e)
                if (r = t.apply(e[i], n), r === !1) break;
        } else if (a) {
            for (; o > i; i++)
                if (r = t.call(e[i], i, e[i]), r === !1) break;
        } else
            for (i in e)
                if (r = t.call(e[i], i, e[i]), r === !1) break;
    return e
},
trim: b && !b.call("\ufeff\u00a0") ? function(e) {
    return null == e ? "" : b.call(e)
} : function(e) {
    return null == e ? "" : (e + "").replace(c, "")
},
makeArray: function(e, t) {
    var n = t || [];
    return null != e && (M(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : b.call(n, e)), n
},
inArray: function(e, t, n) {
    var r;
    if (t) {
        if (n) return b.call(t, e, n);
        for (r = t.length, n = n ? 0 : n ? Math.max(0, r - n) : n ; n ; r > n; n++)
            if (n in t && t[n] === e) return n
    }
}

```

Fuente: Imagen de Elchinator extraída de Pixabay

Editorial

Agradecimientos

Agradecemos a todos aquellos que han aportado su investigación, su experiencia y su tiempo para la elaboración de este material de estudio.

Autor



Carlos Arnaldo Gonzalez

- Analista de Sistemas de Computación. Colegio Universitario IES Siglo 21.
- Desarrollador de software para diversas empresas locales y extranjeras.
- Miembro del Consejo Profesional de Ciencias Informáticas de la Provincia de Córdoba (CPCIPC).
- Docente de las materias Laboratorio de Programación 1, 2 y 3. Colegio Universitario IES Siglo 21.
- Docente de la materia Programación 2-IA. Colegio Universitario IES Siglo 21
- Docente de las materias Programación 1 y 2-RB. Colegio Universitario IES Siglo 21.
- Desarrollador en Internet de las Cosas (IoT).

Equipo de producción

Producción y dirección general

- Director general: Alberto Rabbat
- Directora Académica: María Fernanda Sin
- Vicedirectora Académica: Érica Bongiovanni

Planificación y coordinación general

- Coordinador de estudios a distancia: Érica Bongiovanni
- Dirección de carrera: Fernando Frías / Informática

Producción Multimedial

- Sebastian Benito
- Nicolás Irusta
- Diego Oliva

Producción Académica

- Ana Giró
- Telmo Torres

Coordinación de sistemas

- Marcela Giraudo

Uso de marcas

Cláusulas de uso de marcas y derechos de autor de terceros.

- Uso atípico de marca ajena: Exclusión de los usos no comerciales del Derecho Marcario. PERSPECTIVAS S.A. en su carácter de titular de los derechos intelectuales sobre la presente obra declara por esta vía que el uso que realiza de marcas comerciales de terceros lo es sólo a los fines informativos y didácticos, para mejor comprensión de los lectores y alumnos del contenido de la obra, siendo el mismo de carácter atípico (uso atípico de marca ajena) y lícito. Este uso, a tenor de la jurisprudencia vigente queda fuera del ius prohibendi, que detenta el titular de cada marca registrada, atento no ser el mismo de carácter comercial en relación al producto que distinguen las referidas marcas, y por ende de índole marcario.-
- Uso de derechos de autor en videos, diskettes, imágenes y audio: Libre utilización -Uso privado- de obras protegidas. PERSPECTIVAS S.A. en su carácter de titular de los derechos intelectuales sobre la presente obra declara por esta vía que el uso que realiza de determinadas grabaciones (audio y video), e imágenes (fotografías) de terceros, lo es a los fines informativos y didácticos, para mejor comprensión de los alumnos del contenido de la obra, siendo el mismo de carácter privado y no comercial, y desde ya respetando el derecho de cita, esto es declarando en toda ocasión la cita o fuente (obra y autor) de la cual se toman los fragmentos de obras de terceros para incorporarlos a la presente (Convenio de Berna, Acta de París, 1971 – Art. 10, § 2 y § 3).-
- Modificación de obras literarias por el titular de los derechos patrimoniales. PERSPECTIVAS S.A. en su carácter de titular de los derechos intelectuales (patrimoniales) sobre la presente obra, aclara que ha autorizado a Carlos Gonzales a la modificación respecto de la obra original publicada con el título de LABORATORIO DE PROGRAMACIÓN III (02/2012) con N° de ISBN 978-987-600-201-1, cuyos autores son Miguel Carrizo y Juan Carlos Casale, constituyéndose en autores morales de la obra referida y modificada. Se declara a todo efecto que, los derechos intelectuales se ceden y mantienen a favor de su titular, PERSPECTIVAS S.A.

Copyright

Gonzalez, Carlos

Laboratorio de Programación III / Carlos Gonzalez ; coordinación general de Erica Bongiovanni ; dirigido por María Fernanda Sin. - 1a ed adaptada. - Córdoba : IES Siglo 21, 2023.

Libro digital, Otros

Archivo Digital: descarga y online

ISBN 978-987-600-412-1

1. Aplicaciones Informáticas. I. Bongiovanni, Erica, coord. II. Sin, María Fernanda , dir. III. Título.

CDD 005.102

1ra Edición

© 2023 - Editorial IES Siglo 21

Buenos Aires 563

TE: 54-351-4211717

5000 - Córdoba

Queda hecho el depósito que establece la Ley 11723

Libro de edición argentina

No se permite la reproducción parcial o total, el almacenamiento, el alquiler, la transmisión o la transformación de este libro, en cualquier forma o por cualquier medio, sea electrónico o mecánico, mediante fotocopias, digitalización u otros métodos, sin el permiso previo y escrito del editor. Su infracción está penada por las leyes 11723 y 25446.

Se terminó de replicar durante el mes de Abril de 2023 en el departamento de Logística en Editorial IES Siglo 21.

Índice

Índice	5
Cómo está organizado este texto	7
Introducción	9
Esquema	10
Situación profesional 1: Farmacia “La Salud”	11
SP1/H1:Componentes de ADO .NET	13
SP1/Autoevaluación 1	22
SP1/H2: Las clases Streams	23
SP1/Autoevaluación 2	28
SP1/H3: Bloque Try – Catch, Excepciones	29
SP1/Autoevaluación 3	33
SP1/H4: Cuadro de diálogo común	34
SP1/Autoevaluación 4	38
SP1/H5 Trace, EventLogs	39
SP1/Autoevaluación 5	49
SP1/Ejercicio resuelto	50
SP1/Ejercicio por resolver	68
SP1/Evaluación de paso	69
Situación profesional 2:Backup Programado	71
SP2/H1: Control DateTimePicker	74
SP2/Autoevaluación 1	77
SP2/H2: Control Timer	79
SP2/Autoevaluación 2	82
SP2/H3: Transacciones en ADO .NET	84
SP2/Autoevaluación 3	88
SP2/Ejercicio resuelto	90
SP2/Ejercicio por resolver	108
SP2/Evaluación de paso	110
Situación profesional 3: Incendios	112
SP3/H1: Control ImageList	114
SP3/Autoevaluación 1	117
SP3/H2: Controles de presentación jerárquica de datos	119
SP3/Autoevaluación 2	128

SP3/H3: Controles para visualización de información en forma de lista (ListView)	129
SP3/Autoevaluación 3	136
SP3/H4: Control StatusStrip, barras de estado	138
SP3/Autoevaluación 4	144
SP3/H5: Control ToolStrip, barras de herramientas	146
SP3/Autoevaluación 5	150
SP3/Ejercicio resuelto	152
SP3/Ejercicio por resolver	168
SP3/Evaluación de paso	170
Situación profesional 4: Agricultura	172
SP4/H1: Control MS Chart	175
SP4/Autoevaluación 1	186
SP4/H2: Control ProgressBar	188
SP4/Autoevaluación 2	190
SP4/H3: Control TabControl	192
SP4/Autoevaluación 3	197
SP4/Ejercicio resuelto	198
SP4/Ejercicio por resolver	218
SP4/Evaluación de paso	220
Situación profesional 5: Reserva de pasajes	222
SP5/H1: Control PictureBox	228
SP5/Autoevaluación 1	233
SP5/H2: GDI+. Gráficos	235
SP5/Autoevaluación 2	245
SP5/H3: Controles Contenedores de tipo Panel	248
SP5/Autoevaluación 3	256
SP5/H4: Impresión	258
SP5/Autoevaluación 4	272
SP5/Ejercicio resuelto	274
SP5/Ejercicio por resolver	312
SP5/Evaluación de paso	314
Cierre	316
Bibliografía	317

Cómo está organizado este texto

Usted está en presencia de este texto que los autores proponen para la comprensión y estudio de la asignatura. Ha sido preparado y diseñado para facilitar el acceso al conocimiento, a partir de una secuencia cuyo punto de partida es la práctica profesional cotidiana y no la teoría alejada de la realidad.

Está organizado de la siguiente manera:

- **Introducción.** Indica qué papel desempeña la asignatura dentro de la carrera y los conceptos básicos que usted conocerá.
- **Esquema.** Muestra los enlaces que unen los conceptos centrales de la asignatura entre sí.
- **Situación profesional.** Lo ubica frente a un problema de la práctica profesional cotidiana que puede ser resuelto, ya que existe al menos una solución para ello, a través de conocimientos específicos que en cada caso se aportan.
- **Herramientas.** Son los conocimientos necesarios para resolver la Situación Profesional planteada.
- **Autoevaluación.** Para que usted compruebe si ha comprendido correctamente lo que se explicó en una herramienta, los autores proponen la resolución de actividades y le ofrecen las respuestas.
- **Ejercicio resuelto.** Bajo este título encontrará una manera de resolver los problemas de práctica profesional planteados, con la selección de las herramientas pertinentes.
- **Ejercicio por resolver.** Ahora le toca a usted. Es el momento de aplicar las herramientas a una situación profesional nueva o similar a la ya expuesta. Todas las dudas que le aparezcan podrán ser planteadas a su docente.
- **Evaluación de paso.** Para que usted compruebe si ha comprendido correctamente lo que se explicó en las distintas herramientas que hasta el momento se han presentado, los autores proponen la resolución de actividades y le ofrecen las respuestas.

- **Bibliografía.** Se indican los textos, revistas y links de consulta a los que podrá recurrir para complementar o ampliar algunos temas.

Introducción

En el camino recorrido como estudiante de informática, aprendió distintos contenidos y fundamentos de programación que le permiten desarrollar algunas aplicaciones totalmente funcionales. En esta materia lo acompañaremos en el aprendizaje de nuevas herramientas que le permitirán afianzar el dominio del lenguaje de programación, aprender nuevas técnicas y formas de aplicarlo.

En este material de estudio conocerá herramientas avanzadas para el desarrollo de aplicaciones con mayor complejidad, utilizando herramientas de consulta de datos conectadas a bases de datos, herramientas gráficas, controles para representar estadísticas, generadores de reportes para mostrar en pantalla o enviar a un dispositivo de impresión.

La manipulación de información y el análisis de la misma en las organizaciones es muy importante para la toma de decisiones o para la representación real del estado de la misma.

No olvide que manipular correctamente los datos, y representar de la mejor manera los mismos, ayudará a interpretarlos fácilmente y a la correcta toma de decisiones por parte de los usuarios.

Esperamos encuentre en este material nuevos conocimientos y que le sean útiles para aplicarlos en su camino como desarrollador de aplicaciones con la tecnología de .NET, mejorar sus habilidades como informático debe ser un desafío constante.

El autor

Esquema

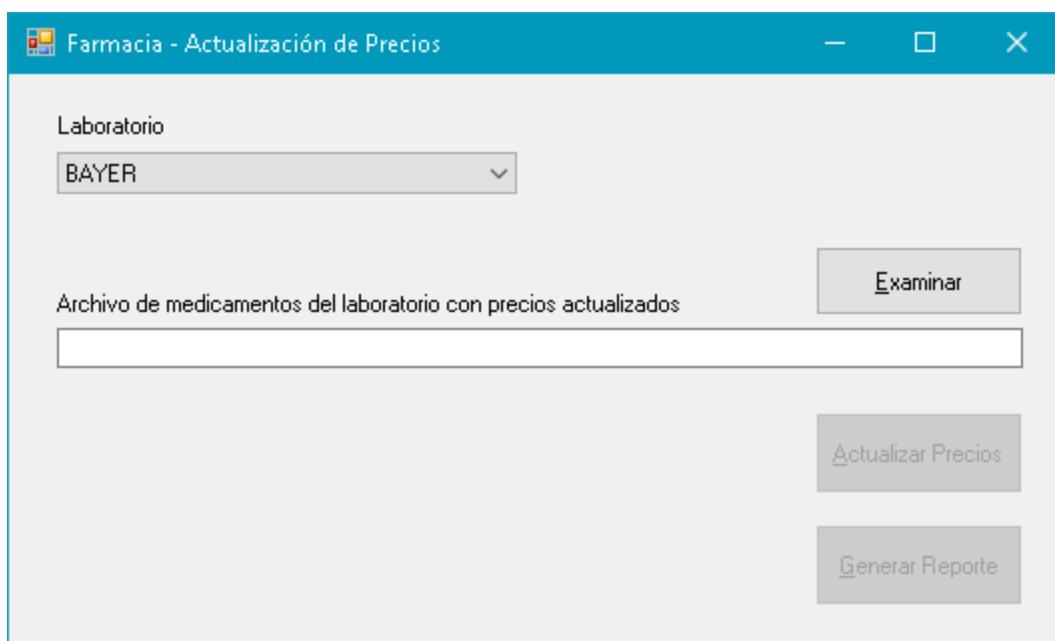


Situación profesional 1: Farmacia “La Salud”

Usted está haciendo una pasantía en la farmacia “La Salud”. El profesional farmacéutico que la gestiona, sabiendo que usted está estudiando informática en el Colegio Universitario IES, le ha solicitado el desarrollo de una aplicación para actualizar los precios de sus medicamentos a partir de archivos de texto (txt) enviados por los laboratorios que los producen.

Los archivos con las actualizaciones de los precios se tienen que elegir de cualquier medio y

Ubicación, por ejemplo, de una carpeta del disco duro, de una unidad extraíble, de una lectora de CD/DVD, etc.



En la interfaz, el usuario selecciona de una lista desplegable el nombre del laboratorio cuyos medicamentos desea actualizar. Seguidamente con un botón de comando deberá buscar y seleccionar de la ubicación adecuada el archivo con los datos de los medicamentos. Para comenzar el proceso de actualización deberá pulsar el botón “Actualizar”.

Cuando se concluye el proceso de actualización de precios deberá mostrarse un mensaje indicativo, si se detecta algún error también deberá informarse por pantalla.

La aplicación debe controlar que no se actualicen datos de listas con fecha anterior a la que tiene cada medicamento.

También deberá controlar si no coincide el laboratorio y el archivo elegido cuando se actualizan los precios. El archivo de texto para la actualización contiene en cada línea el código del medicamento, el nombre del medicamento y el precio del medicamento.

El botón “Generar Reporte” deberá generar un archivo “.txt” con el nombre y ubicación que el usuario elija, el reporte mostrará el detalle de la tabla Medicamentos con los precios actualizados. El formato del reporte debe tener esta estructura:

Reporte de Medicamentos Actualizados				
Código	Nombre	Lab	Precio	Fecha
1010002	GENIOL X 3 TABLETAS	101	29,50	10/01/2022
1010001	ASPIRINA X 6 TABLETAS	101	29,70	25/03/2022
1020003	PASTILLAS IBUPROFENO ADULTOS	102	52,00	01/04/2022
1020002	JARABE PARA LA TOS NIÑOS	102	85,00	01/04/2022
1010005	NOVALGINA X 4 TABLETAS	101	32,00	25/03/2022
1020005	JARABE PARA LA FIEBRE NIÑOS	102	90,00	19/02/2022

Fecha del reporte: 05/03/2023

La base de datos se llama Farmacia.mdb y contiene dos tablas denominadas laboratorios y medicamentos:

- La tabla “laboratorios” contiene un número para identificar al laboratorio y el nombre del laboratorio. La clave principal es el número de laboratorio.
- La tabla “medicamentos” contiene el código de medicamento del laboratorio, el nombre del medicamento, el número de laboratorio al que pertenece el medicamento, el precio del medicamento y la fecha de la última actualización de precios. La clave principal es el código del medicamento.

SP1/H1:Componentes de ADO .NET

Para resolver nuestra situación profesional necesitaremos tener acceso a la base de datos y a sus tablas para realizar lecturas y escrituras de sus datos, para lograr esto trabajamos con las clases de ADO .NET pensadas para resolver de manera eficiente la interacción desde el código de C# con las bases de datos. Este tema se trató con profundidad y detalle en el texto de Laboratorio de Programación 2, por lo que sugerimos una revisión del mismo para despejar cualquier duda. Seguidamente presentamos, a modo de resumen, los puntos más importantes centrados en la resolución de la SP.

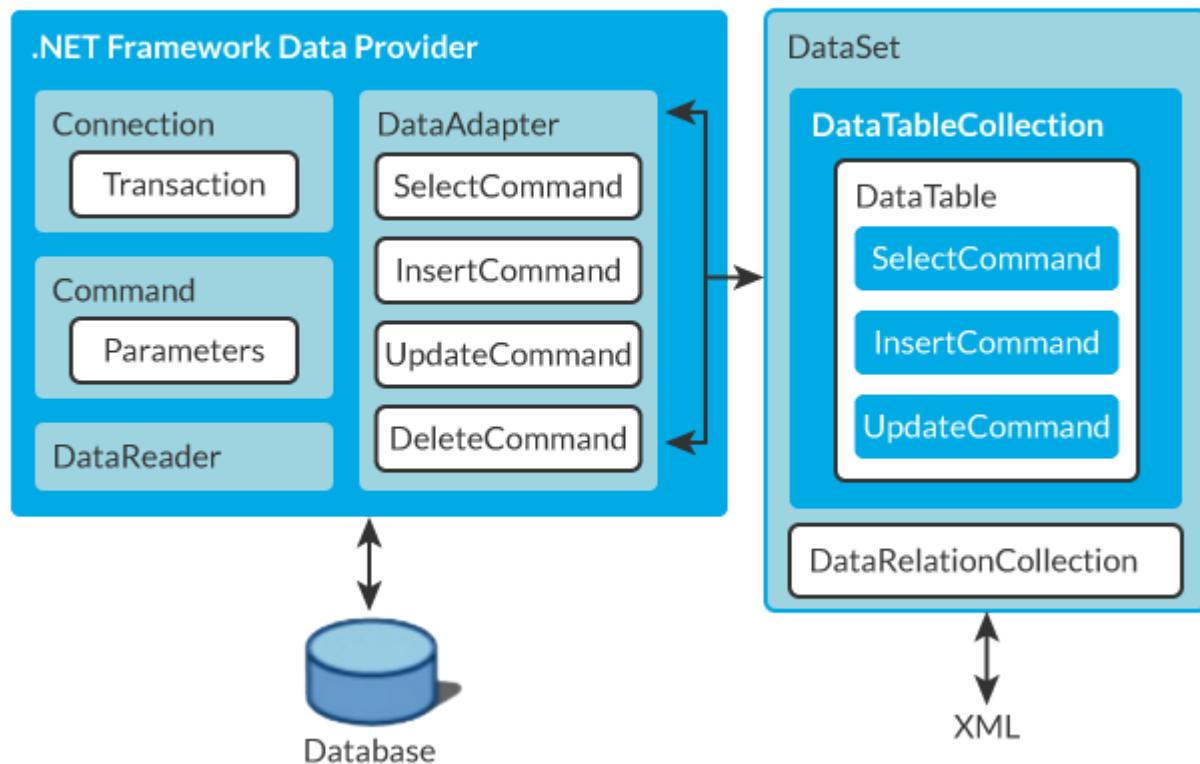
ADO .NET

ADO .NET es el modelo de objetos implementado en la plataforma .NET (ActiveX Data Objects), usado para el acceso a todo tipo de datos. ADO .NET está diseñado para trabajar con conjuntos de datos desconectados, lo que permite reducir el tráfico de datos en la red y lograr así un mejor rendimiento en las aplicaciones que lo utilizan. ADO .NET también permite trabajar en modo conectado.

La estructura de ADO .NET es la siguiente:

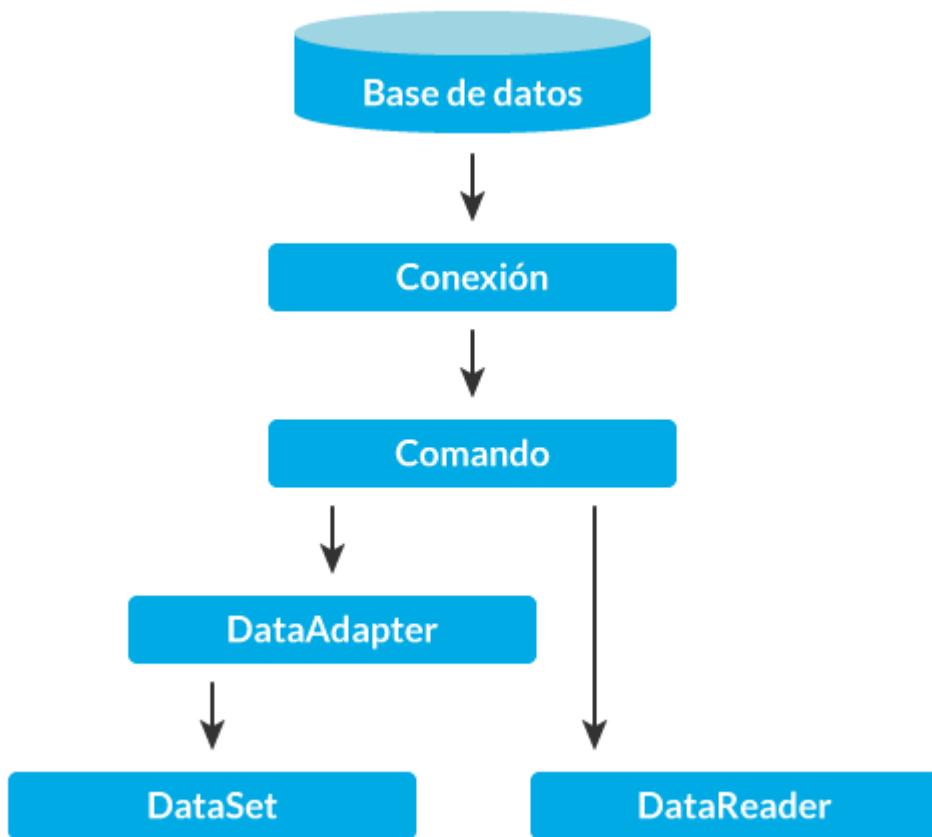
Diagrama de componentes de ADO .NET

Un conjunto de interfaces, clases, estructuras y enumeraciones que permiten el acceso a datos desde la plataforma .NET de Microsoft. En la imagen vemos sus principales herramientas y relaciones.



Resumiendo, ADO.NET nos ofrece las herramientas para acceder a una base de datos, encontraremos mucha información sobre sus características y usos, aquí exponemos algunas de las más comunes para luego dar lugar a la práctica.

Componentes principales de ADO .NET



Los proveedores de datos de .NET y el espacio de nombres System.Data proporcionan los objetos ADO.NET que utilizaremos en un escenario desconectado. ADO.NET proporciona un modelo de objetos común para proveedores de datos de .NET. La siguiente lista describe los principales objetos ADO.NET que utilizaremos en un escenario desconectado:

- Connection: establece y gestiona una conexión a una fuente de datos específica. Por ejemplo, la clase OleDbConnection se conecta a fuentes de datos OLE DB.
- Command: ejecuta un comando en una fuente de datos. Por ejemplo, la clase OleCommand puede ejecutar instrucciones SQL en una fuente de datos OLE DB.
- DataSet: diseñado para acceder a datos con independencia de la fuente de datos. En consecuencia, podemos utilizarlo con varias y diferentes fuentes de datos, con datos XML, o para gestionar datos locales a la aplicación. El objeto DataSet contiene una colección de uno o más objetos DataTable formados por

filas y columnas de datos, además de clave principal, clave foránea, restricciones e información de la relación sobre los datos en los objetos DataTable.

- DataReader: proporciona un flujo de datos eficaz, sólo-reenvío y de sólo-lectura desde una fuente de datos.
- DataAdapter: utiliza los objetos Connection, Command y DataReader implícitamente para poblar un objeto DataSet, y para actualizar la fuente de datos central con los cambios efectuados en el DataSet. Por ejemplo, OleDbDataAdapter puede gestionar la interacción entre un DataSet y una base de datos Access.

En la práctica repasamos lo necesario para:

1. Realizar la conexión con la base de datos.
2. Leer el contenido de una tabla de la base de datos.
3. Modificar y agregar registros a una tabla de la base de datos.

Conexión:

1. Importar los nombres de espacio de System.Data.OleDb:

```
Imports System.Data.OleDb
```

2. Establecer el valor de la cadena de conexión. Podemos obtener la sentencia de las cadenas de conexión a diferentes bases de datos en esta página: Cadena de conexión.
3. Por ejemplo, para una base de datos de tipo Access cuyo nombre sea "database.mdb", la cadena de conexión tendrá este contenido:

```
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=database.mdb".
```

1. Crear la conexión y realizar su apertura; se usa un objeto de tipo OleDbConnection, su propiedad ConnectionString y el método Open(). Por ejemplo:

```
OleDbConnection cnn = new OleDbConnection();  
cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data  
Source=database.mdb";  
cnn.Open();
```

Lectura de una tabla

1. Ejemplo de lectura de la tabla ‘Cantantes’:

Cantantes	
IdCantante	Nombre
1	CantanteA
2	CantanteB
3	CantanteC
*	0

Lectura de la tabla “Cantantes” usando un DataSet:

```
OleDbConnection objcnn = new OleDbConnection();
OleDbCommand objcmd = new OleDbCommand();
OleDbDataAdapter objda;
DataSet objds = new DataSet(); // objeto DataSet as usar
// definir la cadena de conexión
string strcnn = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=Cantantes.mdb";
// asignar la cadena al objeto conexión
objcnn.ConnectionString = strcnn;
// abrir la conexión con la base de datos
objcnn.Open();
// establecer las propiedades al objeto comando
objcmd.Connection = objcnn;
objcmd.CommandType = CommandType.TableDirect;
objcmd.CommandText = "Cantantes"; // nombre de la tabla a leer
// crear el objeto DataAdapter pasando como parámetro el objeto comando que queremos vincular
objda = new OleDbDataAdapter(objcmd);
// ejecutar la lectura de la tabla y almacenar su contenido en el dataAdapter
objda.Fill(objds, "Cantantes");
// controlar si hay registros disponibles
if(objds.Tables["Cantantes"].Rows.Count > 0)
{
    string datos = ""; // variable auxiliar para almacenar los datos de la tabla
    // recorrer los registros
    foreach (DataRow dr in objds.Tables["Cantantes"].Rows)
    {
        // concatenar los campos de la tabla 'Cantantes': 'IdCantante' y 'Nombre'
        datos += dr["IdCantante"].ToString() + ", " + dr["Nombre"] + "\r\n";
    }
    MessageBox.Show(datos, "Tabla de Cantantes - DataSet");
}
// cerrar la conexión
objcnn.Close();
```

Algunos comentarios sobre el código anterior:

1. El método “Fill” se encarga de ‘rellenar’ el DataSet con todos los registros de la tabla definida en el objeto comando y al mismo tiempo se le asigna un nombre

dentro del DataSet, en este caso el nombre coincide con el que tiene la tabla en la base de datos, esto facilita el control del contenido del DataSet. El proceso puede repetirse usando otros objetos comando y obtener en el DataSet varias tablas, cada una con su propio nombre y contenido. Para cada tabla se necesita un objeto comando y su correspondiente objeto dataAdapter.

2. Las tablas en el DataSet forman una colección cuyos elementos (las tablas) pueden identificarse con su nombre o con un índice que indique su posición dentro de la colección:
objds.Tables["Cantantes"] sería equivalente a: objds.Tables[0] si esa fuera la primera tabla agregada al DataSet.
3. A su vez la tabla posee una colección de objetos DataRow que puede ser recorrida con un ciclo foreach() y para acceder a los campos de la tabla se usan los nombres de los campos, así dr["IdCantante"] devuelve el valor del campo "IdCantante" en el registro actual.

Agregar nuevos registros a una tabla

Para agregar registros nuevos a la tabla debemos primero crear un objeto de tipo DataRow con la misma estructura de la tabla y luego asignar los valores a cada uno de sus campos, una vez que el registro está completo lo agregaremos a la tabla que tenemos en el DataSet. Finalmente deberemos sincronizar el cambio con la base de datos real.

Necesitamos entonces crear un nuevo **DataRow** y también un objeto de tipo **OleDbCommandBuilder** para que el DataAdapter sepa cómo debe actualizar la base de datos con los cambios realizados en el DataSet.

Retomamos nuestro ejemplo con la tabla de Cantantes para agregar un nuevo registro.

```

// obtenemos una referencia a la tabla de Cantantes
DataTable dt = objds.Tables["Cantantes"];
// creamos el nuevo DataRow con la estructura de campos de la tabla Cantantes
DataRow dr = dt.NewRow();
// asignamos los valores a todos los campos del DataRow
dr["IdCantante"] = 4;
dr["Nombre"] = "CantanteD";
// agregamos el DataRow a la tabla de Cantantes
dt.Rows.Add(dr);
// creamos el objeto OleDbCommandBuilder pasando como parámetro el DataAdapter
OleDbCommandBuilder cb = new OleDbCommandBuilder(objda);
// actualizamos la base con los cambios realizados
objda.Update(objds, "Cantantes");

```

El método `NewRow()` del objeto `DataTable` crea un nuevo `DataRow` con los campos de la tabla pero con su contenido vacío, es decir sin valores iniciales.

El objeto `OleDbCommandBuilder` es necesario para que el `DataAdapter` tenga la información sobre cómo ejecutar las actualizaciones sobre la base de datos.

Modificar valores en registros ya existentes

Para modificar los valores en algunos campos de un registro determinado debemos obtener una referencia al registro con un objeto `DataRow`, luego iniciar el modo de edición del `DataRow`, hacer los cambios de los valores en sus campos y finalizar el modo edición. Finalmente se actualizarán los cambios sobre la base de datos como en el caso de agregar un nuevo registro.

El modo edición se controla con los métodos `"BeginEdit()"` y `"EndEdit()"` del objeto `DataRow`, mientras esté iniciado el modo de edición se podrán hacer cambios en los valores del `DataRow`, de otra forma los cambios serán descartados.

Por ejemplo, podemos modificar el nombre del cantante con identificador igual a 1:

```

// obtenemos una referencia a la tabla de Cantantes
DataTable dt = objds.Tables["Cantantes"];

// recorrer los registros de la tabla
foreach (DataRow dr in dt.Rows)
{
    // buscar el cantante con ID = 1
    if((int)dr["IdCantante"] == 1)
    {
        // establecer el modo de edición del DataRow

```

```

        dr.BeginEdit();
        // asignamos el nuevo valor al nombre del cantante
        dr["Nombre"] = "NuevoCantanteA";
        // finalizamos el modo edición sobre del DataRow
        dr.EndEdit();
        break; // salir del foreach
    }
}
// creamos el objeto OleDbCommandBuilder pasando como parámetro el DataAdapter
OleDbCommandBuilder cb = new OleDbCommandBuilder(objda);
// actualizamos la base con los cambios realizados
objda.Update(objds, "Cantantes");

```

Uso de la Clave Primaria (*Primary Key*) y el método '*Find*'

La clave primaria de una tabla es el conjunto de campos cuyos valores no pueden repetirse y que por lo tanto permiten identificar a cualquier registro como único en la tabla. Este concepto es propio de las bases de datos relacionales y nos permite gestionar el contenido de una tabla de forma más eficiente.

Podemos agregar la clave primaria a una tabla que ya tenemos cargada en el DataSet de esta forma:

```

// crear el arreglo para las columnas de la clave primaria
DataColumn[] objdc; // arreglo de tipo DataColumn para la clave primaria
objdc = new DataColumn[1]; // la cantidad de elementos del arreglo depende de los
                           // campos que definen la clave primaria en la tabla
// asignar las columnas de la tabla que forman la clave primaria
objdc[0] = objds.Tables[0].Columns["IdCantante"];
// establecer la propiedad PrimaryKey con las columnas correspondiente
objds.Tables[0].PrimaryKey = objdc;

```

Necesitamos un arreglo de elementos de tipo DataColumn cuya cantidad de posiciones sea igual a la cantidad de columnas que forman la clave primaria de la tabla, en la mayoría de los casos será una sola posición, pero hay tablas cuya clave primaria está compuesta por 2 o más campos también. Cada elemento del arreglo recibirá el valor de la columna correspondiente y finalmente el arreglo completo se asigna a la propiedad '*PrimaryKey*' de la tabla que tenemos en el DataSet.

El uso de la clave primaria está muy relacionado con las búsquedas ya que en lugar de recorrer los registros uno por uno hasta encontrar el que buscamos podemos aplicar el método '*Find*' pasando como parámetro el valor del campo clave y obtener como

resultado el registro buscado, todo en un solo paso, por ejemplo, si repetimos el proceso para modificar un registro:

```
// obtenemos una referencia a la tabla de Cantantes
DataTable dt = objds.Tables["Cantantes"];
// creamos el arreglo para los valores que forman la clave a buscar
object[] valor = new object[1];
valor[0] = 4; // asignamos el valor de IdCantante que queremos buscar
// ejecutar la búsqueda sobre la tabla en el DataSet
DataRow dr = objds.Tables[0].Rows.Find(valor);
// controlar el resultado de la búsqueda
if(dr is null == false) // si no es nulo
{
    // establecer el modo de edición del DataRow
    dr.BeginEdit();
    // asignamos el nuevo valor al nombre del cantante
    dr["Nombre"] = "NuevoCantanteA";
    // finalizamos el modo edición sobre del DataRow
    dr.EndEdit();
    break; // salir del foreach
}
// creamos el objeto OleDbCommandBuilder pasando como parámetro el DataAdapter
OleDbCommandBuilder cb = new OleDbCommandBuilder(objda);
// actualizamos la base con los cambios realizados
objda.Update(objds, "Cantantes");
```

Si comparamos con el ejemplo anterior de modificar, vemos que ya no necesitamos implementar un recorrido de los registros de la tabla hasta encontrar el valor a borrar, ahora la búsqueda es directa.

Otro punto a tener muy en cuenta es el de controlar siempre el resultado de la búsqueda ya que el método 'Find' devolverá nulo (*null*) cuando el valor buscado no existe en la tabla.

SP1/Autoevaluación 1

1. Indique la opción correcta:

El objeto OleDbConnection usa la propiedad ConnectionString para especificar el tipo y nombre de la base de datos.

- Verdadero
- Falso

2. Indique la opción correcta:

El objeto OleDbCommand es el que establece la conexión con la base de datos.

- Verdadero
- Falso

3. Marque la/s opción/es correcta/s:

Un DataSet representa un conjunto completo de datos y se puede usar de distintas formas:

- a. Crear por programa tablas y llenarlas con datos.
- b. Obtener las tablas de un origen de datos ya existente.
- c. Crear el DataSet a partir de un archivo de texto (.txt)

4. Indique la opción correcta:

El método "Fill" se usa para llenar el DataSet con los datos de una tabla.

- Verdadero
- Falso

5. Marque la/s opción/es correcta/s:

Para acceder a una tabla de la colección de tablas en un DataSet se usa esta sintaxis:

- dataset.Tables[1]
- dataset.Tables["NombreTabla"]

Respuestas correctas¹

¹1) Verdadero. 2) Falso. 3) a, b. 4) Verdadero 5) a, b.

SP1/H2: Las clases Streams

Como ya estudiamos en la materia Laboratorio de Programación 2, en C# un “Stream” representa una corriente o flujo de datos, flujos que son muy usados en el manejo de archivos ya que implementan una capa de código extra que facilita al programador el acceso a los datos del archivo. Recomendamos revisar este tema en el texto de la materia Laboratorio de Programación 2 para recordar sus características y aplicaciones. De todas formas a continuación exponemos los puntos más importantes que necesitamos conocer.

Para los procesos de escritura y lectura disponemos de tipos de stream especializados: “StreamWriter” para realizar las escrituras en un archivo y “StreamReader” para las lecturas de archivos.

Repasamos brevemente cómo trabajar con el StreamWriter que necesitaremos para generar los reportes de nuestra aplicación.

Debemos agregar a nuestro código el espacio de nombres para IO (Input – Output):
using System.IO;

StreamWriter

Como dijimos antes el StreamWriter nos permite grabar datos. Necesitamos crear un objeto de la clase StreamWriter usando como parámetro la ruta incluyendo el nombre del archivo que queremos escribir. También debemos tener en cuenta el modo de trabajo con el archivo, puede ser escritura simplemente y en ese caso el archivo será creado como un nuevo archivo vacío o podemos trabajar con el modo “append” para continuar grabando datos en un archivo que ya existe sin perder la información previamente grabada.

Ejemplos:

1. Abrir y crear un archivo para escribir en él:

```
StreamWriter sw = new StreamWriter(NombreArchivo, false);
```

El primer parámetro: NombreArchivo, contiene la ruta y nombre del archivo, por ejemplo, “C:\\\\Datos\\\\Reporte.txt”.

El segundo parámetro: “false” indica que el archivo será creado desde cero, sin importar que ya exista. Si no existe se crea, si ya existe se borra y se crea de nuevo.

2. Abrir un archivo que ya existe para agregar más datos (modo ‘append’):

```
StreamWriter sw = new StreamWriter(NombreArchivo, true);
```

Para que nos permita seguir grabando sobre un archivo que ya existe sin perder su contenido el segundo parámetro debe ser “true” (modo append). En este modo si el archivo no existe será creado, pero si ya existe solamente se abre sin borrar su contenido y nos permitirá seguir grabando más datos.

Después de crear el stream ya podemos grabar datos sobre el archivo, la clase StreamWriter posee varios métodos para ello:

- Write: graba un string en el archivo
- WriteLine: graba un string en el archivo y agrega un salto de línea al final.

Ejemplo: grabar el contenido de dos variables en una línea del archivo:

```
StreamWriter sw = new StreamWriter(NombreArchivo, true);
int Código = 1000;
string Materia = "Lenguaje Java";
sw.WriteLine(Código.ToString() + "," + Materia);
```

Usamos el método “WriteLine” y como parámetro se concatena el valor de la variable “Número” convertida a string con una coma y con el contenido de la variable “Nombre”. Todo se grabará en una única línea en el archivo y se termina con un salto de línea. Resultaría:

1000,Lenguaje Java

Finalmente se debe cerrar el stream, es decir cerrar la conexión con el archivo físico y luego liberar los recursos del stream, usamos el método “Close” para lo primero y “Dispose” para liberar los recursos:

```
sw.Close();  
sw.Dispose();
```

El método “Close” nos asegura que todos los datos enviados al stream son grabados en el archivo y el uso del método “Dispose” es particularmente importante para que no queden objetos sin usar en la memoria del sistema.

StreamReader

El StreamReader se usa para leer datos desde un archivo, el esquema de uso es muy similar al StreamWriter, creamos el objeto indicando por parámetro la ruta y nombre del archivo que deseamos leer, el archivo debe existir, de lo contrario se genera un error.

Métodos principales de **StreamReader**:

- **Read**: lee un carácter de la secuencia
- **ReadLine**: lee caracteres hasta el fin de línea
- **ReadBlock**: lee un bloque de caracteres, se especifica la cantidad a leer y el buffer destino.
- **ReadToEnd**: lee desde la posición actual hasta el fin de la secuencia.

Ejemplo de lectura con el método “ReadLine”:

```
String Linea;  
StreamReader sr = new StreamReader(NombreArchivo);  
while (sr.EndOfStream == false)  
{  
    Linea = sr.ReadLine();  
    // mostrar la linea  
    MessageBox.Show(Linea);  
}  
sr.Close();  
sr.Dispose();
```

Se crea el StreamReader con el nombre del archivo, si no se incluye la ruta, el archivo se tratará de abrir en el mismo directorio de la aplicación. Seguidamente, se realiza un ciclo repetitivo con la condición de fin de stream usando la propiedad **EndOfStream** del streamReader. Mientras esta propiedad tenga el valor falso podemos continuar realizando lecturas, cuando “EndOfStream” es verdadero nos indica que la secuencia llegó al final.

Dentro del ciclo “while” realizamos la lectura de una línea completa del archivo y la almacenamos en la variable “Linea”, luego mostramos el contenido de la variable. Al salir del ciclo “while” debemos cerrar la secuencia usando el método “Close” y finalmente liberar los recursos del stream con el método “Dispose”.

Proceso de los datos obtenidos de cada línea del archivo

Retomando el ejemplo visto para grabar datos con StreamWriter, supongamos que además de leer el archivo necesitamos tener cada valor en forma individual para realizar otro proceso, en este caso sería el código y el nombre de la materia grabados en cada línea. Una alternativa es usar el separador entre valores, el carácter coma (”,”) para descomponer la línea en varias partes y asignar cada parte a una variable. Dicha descomposición o partición se puede hacer con el método “Split” de la clase string, ejemplos:

```
string Linea = "1000,Lenguaje Java";
string[] partes = Linea.Split(',');
int codigo = int.Parse(partes[0]);
string materia = partes[1];
```

Con el método Split con el carácter ” como parámetro, obtenemos un arreglo de tipo string con varios elementos, tantos como comas existan en el string más uno. En este ejemplo hay una sola coma, se obtiene un arreglo de dos elementos.

Cada elemento del arreglo se podrá entonces convertir al tipo de dato correcto y asignar a una variable. En nuestro ejemplo, el elemento de la posición cero se convierte a entero y se asigna a la variable “codigo”, el elemento de la posición 1 se asigna a la variable “materia” y no necesita conversión ya que es de tipo string.

También podemos obtener el mismo resultado sin usar la variable “partes”, simplemente agregamos a la llamada del método Split el índice del elemento que nos interesa acceder:

```
string Linea = "1000,Juan Castro";
int numero = int.Parse(Linea.Split(',')[0]);
string nombre = Linea.Split(',')[1];
```

Para evitar errores en la conversión de números con parte decimal se recomienda especificar el valor del parámetro para la configuración regional o “cultura” deseada: Español - Latinoamérica, o Inglés - EEUU, etc.

Ejemplo: suponemos un string que contiene dos valores decimales separados por coma:

```
string Linea = "123.45,234.56";
string[] partes = Linea.Split(',');
decimal numero1 = decimal.Parse(partes[0], CultureInfo.InvariantCulture);
decimal numero2 = decimal.Parse(partes[1], CultureInfo.InvariantCulture);
```

El segundo parámetro del método Parse se denomina “CultureInfo” y representa la configuración regional aplicada para los formatos de fecha, idioma, y de números que varían de país o región. Como el punto se usa para separar la parte decimal de un número real en la región de EEUU entonces debemos aplicar ese formato al método Parse, hay dos formas de hacerlo, una es con el valor “CultureInfo.InvariantCulture” que usa de modo predeterminado los formatos independientes de la cultura o configuración regional del sistema donde se ejecute la aplicación, es decir toma las configuraciones básicas del lenguaje C# sin importar la región configurada en el sistema.

Para poder trabajar con los valores de “CultureInfo” se debe agregar al código la cláusula “using”:

```
using System.Globalization;
```

Más detalles del uso de “CultureInfo”:

[CultureInfo Class \(System.Globalization\) | Microsoft Docs](#)

1. Indique la opción correcta:

La clase “StreamReader” posee propiedades y métodos para realizar lecturas de archivos.

- Verdadero
- Falso

2. Indique la opción correcta:

Con la clase “StreamReader” solamente se pueden grabar datos en archivos que ya existen.

- Verdadero
- Falso

3. Indique la opción correcta:

Al leer datos de un archivo la lectura se puede realizar de un único carácter, una línea completa o de un bloque de caracteres.

- Verdadero
- Falso

4. Indique la opción correcta:

Para cerrar un archivo que ya no se necesita se usa el método “FileClose”.

- Verdadero
- Falso

5. Indique la opción correcta:

Al realizar lecturas de un archivo se puede usar la propiedad “EndOfStream” para determinar si todos los datos han sido leídos.

- Verdadero
- Falso

Respuestas correctas²

²1) Verdadero. 2) Falso. 3) Verdadero. 4) Falso. 5) Verdadero.

SP1/H3: Bloque Try – Catch, Excepciones

El manejo de las excepciones es un tema conocido cuyo tratamiento podemos implementar con la estructura try-catch, pero ahora veremos cómo implementar nuestras propias excepciones para agregar mejores prestaciones a nuestros desarrollos. Se trata de crear nuevas clases que hereden de la clase “Exception” y redefinir algunas propiedades o métodos para obtener la funcionalidad deseada.

Desde el punto de vista técnico del lenguaje: “*En .NET, una excepción es un objeto que hereda de la clase System.Exception. Una excepción se inicia desde un área del código en la que se ha producido un problema. La excepción se pasa hacia arriba en la pila hasta que la aplicación la controla o el programa finaliza.*”

Recordemos el esquema de uso de la estructura try-catch:

El formato general del bloque es este:

```
try
{
    // bloque de sentencias a ejecutar
    // que pueden provocar excepciones
}
catch (tipo1_Exception ex)
{
    // código para tratar la excepción de tipo1
}
catch (tipo2_Exception ex)
{
    // código para tratar la excepción de tipo2
}
finally
{
    // bloque de código que se ejecutará siempre
    // ocurran o no excepciones
}
```

En el bloque “try” se colocan todas las instrucciones que debemos ejecutar normalmente pero que de alguna forma pueden generar una excepción, por ejemplo, la lectura de un archivo o realizar una división de 2 variables, etc.

A continuación, colocamos los bloques “catch” que son los encargados de atrapar las excepciones cuando se presentan. Cada “catch” puede procesar un tipo específico de excepción, por ejemplo: DivideByZeroException, IndexOutOfRangeException, NullReferenceException, InvalidOperationException, FileNotFoundException, etc. O también podemos dejar un único bloque “catch” con la clase base de todas las excepciones: “Exception” por lo que cualquier tipo de excepción que ocurra será atrapado por el bloque “catch”.

En tercer lugar, tenemos el bloque “finally” cuyo contenido será ejecutado siempre, sin importar si en el bloque “try” se dispararon o no excepciones. Este bloque es opcional si existe al menos un bloque “catch”.

Lanzando Excepciones desde nuestro código

Estos bloques try-catch nos permiten capturar las excepciones y darles el tratamiento adecuado para cada caso, pero también es posible relanzar la excepción original o crear un nuevo tipo de excepción por medio la sentencia “throw” (lanzar).

Sintaxis de la sentencia throw:

```
throw e;
```

Donde ‘e’ es un objeto de la clase “System.Exception” o de alguna otra clase derivada de ella.

Ejemplo de uso:

```
int[] numbers = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };
public int GetNumber(int index)
{
    if (index < 0 || index >= numbers.Length)
    {
        throw new IndexOutOfRangeException();
    }
    return numbers[index];
}
```

El método “GetNumber” debe devolver un valor del arreglo “number” de acuerdo al índice que recibe por parámetro, pero si ese índice está fuera del rango permitido se dispara una excepción del tipo “IndexOutOfRangeException” (índice fuera de rango), se usa la sentencia ‘throw’ para lanzar la excepción que deberá ser controlada desde el bloque de código que llame al método “GetNumber”.

El uso de “throw” se puede ver en muchas funciones de validación de datos, disparando excepciones ante situaciones que detectan valores incorrectos o no permitidos por las reglas de negocio de la aplicación. Las excepciones que se disparan pueden ser excepciones ya definidas en el lenguaje o excepciones nuevas, creadas por el programador para personalizar su tratamiento.

Creando excepciones propias

Para crear una excepción propia se debe definir una nueva clase que herede de la clase “Exception” y que implemente al menos 3 constructores, uno sin parámetros, otro con el parámetro para establecer el mensaje de la excepción y otro constructor que reciba el mensaje y la excepción interna (inner exception) que podría contener.

Ejemplo:

```
public class InvalidDepartmentException : Exception
{
    public InvalidDepartmentException() : base() { }

    public InvalidDepartmentException(string message) : base(message) { }

    public InvalidDepartmentException(string message, Exception inner) :
base(message, inner) { }
}
```

En el ejemplo se observa el uso de “base()”, es la forma por medio de la cual se ejecuta el código del constructor de la clase base, de la cual hereda la nueva clase, en este caso sería la clase “Exception”.

Esta nueva clase creada se podrá usar en los bloques catch o para lanzar excepciones con la sentencia throw.

Ejemplo:

```
public void SetDepartment(String name)
{
    if(name.Length == 0)
    {
        throw new InvalidDepartmentException("El nombre no puede estar
vacío");
    }
    // continuar el proceso del nombre
}
```

En el código del ejemplo anterior se crea la nueva excepción invocando al constructor que recibe el mensaje por parámetro.

Todo este contenido desarrollado necesitarás a fin de resolver la actual situación profesional.

1. Indique la opción correcta:

Una excepción es un objeto que contiene información sobre el último error ocurrido.

- Verdadero
- Falso

2. Indique la opción correcta:

La clase “Exception” es la base para crear nuevas excepciones.

- Verdadero
- Falso

3. Indique la opción correcta:

La sentencia “throw” permite lanzar una excepción desde cualquier punto del código.

- Verdadero
- Falso

4. Indique la opción correcta:

El bloque try-catch-finally no permite implementar un control de errores estructurado.

- Verdadero
- Falso

5. Indique la opción correcta:

La sentencia “throw” permite lanzar excepciones sólo si estas son creadas por el programador.

- Verdadero
- Falso

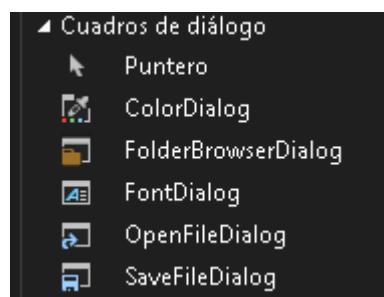
Respuestas correctas³

³1) Verdadero. 2) Verdadero. 3) Verdadero. 4) Falso. 5) Falso.

SP1/H4: Cuadro de diálogo común

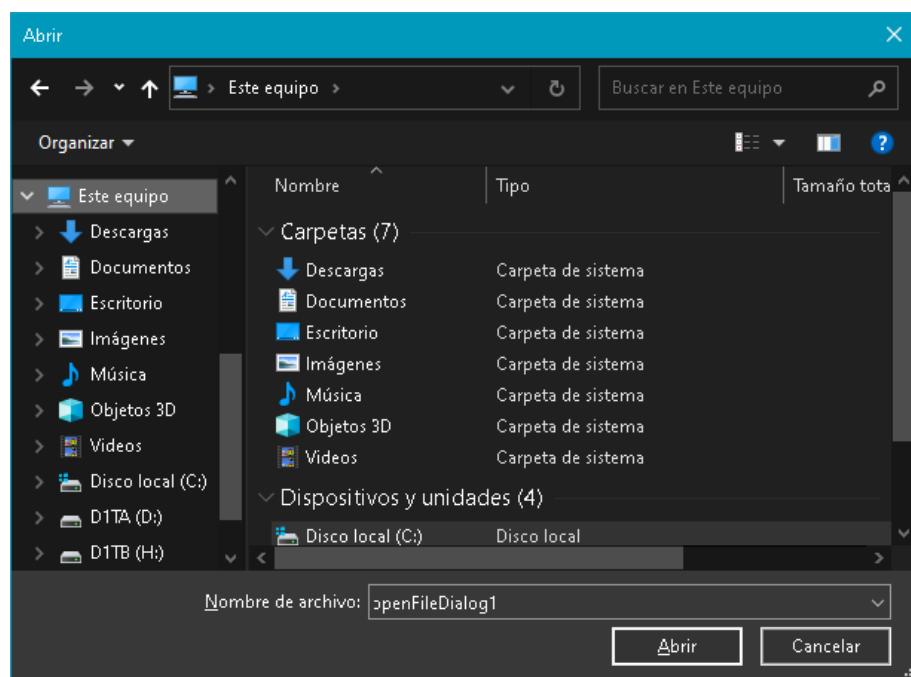
Trabajando con Cuadros de diálogo

Estos cuadros de diálogo nos van a permitir realizar actividades que son repetitivas sin necesidad de implementar su funcionalidad una y otra vez. Nos referimos por ejemplo, a los cuadros de diálogo para la selección de un tipo de letra, para seleccionar un archivo para leer, o seleccionar una carpeta para guardar un archivo, etc. Estos componentes están disponibles en el cuadro de herramientas de *Visual Studio*:

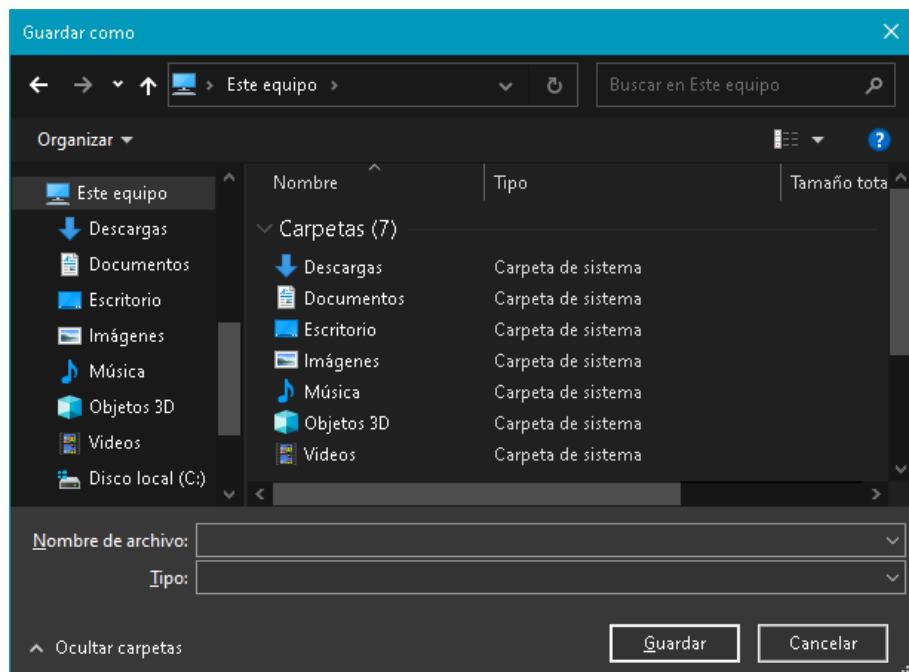


En este caso nos interesa trabajar con los diálogos para seleccionar un archivo y para definir el nombre y ubicación de un archivo a guardar.

- `OpenFileDialog`: permite seleccionar un archivo para abrirlo.



- SaveFileDialog: permite seleccionar una carpeta y especificar el nombre de un archivo para grabarlo.



El uso de todos estos diálogos es bastante simple, para lograr que el diálogo sea visualizado en la pantalla debemos ejecutar su método “ShowDialog()” y luego, cuando el usuario cierre el diálogo podremos acceder a sus propiedades para obtener los valores que fueron seleccionados. Debemos tener en cuenta también que en todos los diálogos tenemos disponible el botón “Cancelar” de manera que el usuario puede cerrar el diálogo con ese botón indicando que se deben descartar las posibles selecciones o modificaciones que hubiere realizado en el diálogo, por lo tanto, siempre debemos controlar de qué forma se cierra el diálogo. Eso se logra consultando el valor de la propiedad “DialogResult” que nos devuelve el método “ShowDialog()”.

El valor ‘OK’ corresponde al botón “Aceptar”, “Abrir” o “Guardar”, según el caso. Si el usuario cierra el diálogo con el botón “Cancelar” no tendremos en cuenta los valores que esperamos obtener del diálogo.

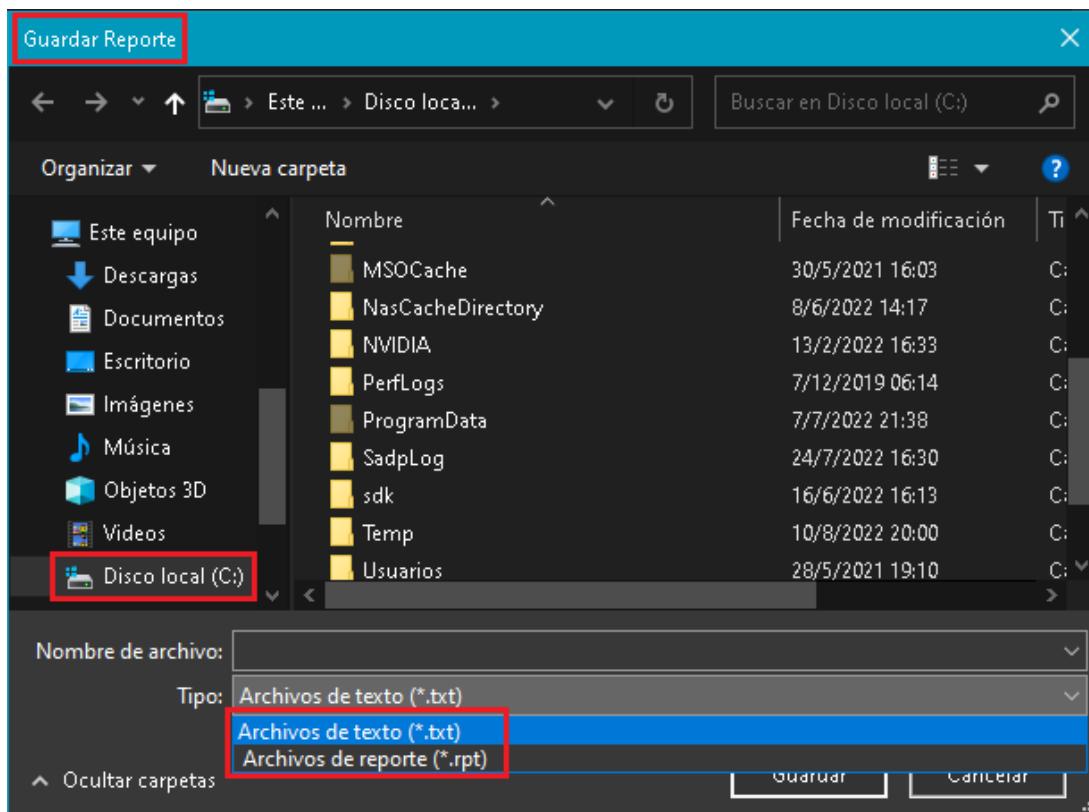
Revisaremos con un poco más de detalle el diálogo “saveFileDialog” más conocido como “Guardar cómo”. Como mencionamos antes cada diálogo tiene una serie de propiedades que pueden ser configuradas por el usuario en el momento de usar el

diálogo, para “saveFileDialog” hay varias propiedades interesantes que tienen bastante uso:

- Filter: los filtros nos permiten especificar la extensión del archivo que vamos a usar, por ejemplo, *.txt, o *.doc, etc. El formato del texto que se asigna a la propiedad Filter es: “descripción del filtro1 | máscara del filtro1 | descripción del filtro2 | máscara del filtro2” y así con todos los filtros que se quieran agregar, siempre se usa el carácter “|” como separador de cada parte. Ejemplo:
`saveFileDialog.Filter = “Archivos de texto (*.txt) | *.txt | Archivos de Reporte (*.rpt) | *.rpt”;`
- FilterIndex: indica qué filtro es el que está seleccionado, el primero tiene el índice cero.
- Title: especifica el título que mostrará el diálogo.
- RestoreDirectory: (booleano) si tiene el valor true el último directorio seleccionado será el que se muestre cuando el diálogo se abra nuevamente.
- InitialDirectory: indica cuál es el directorio inicial al mostrar el diálogo.

Todas estas propiedades se deben establecer antes de ejecutar el método “ShowDialog()” para que en el momento de visualizar el diálogo todos esos valores ya estén asignados.

En la siguiente imagen se puede ver el resultado de asignar “Guardar Reporte” a la propiedad “Title”, el valor “Archivos de texto (*.txt) | *.txt | Archivos de Reporte (*.rpt) | *.rpt” a la propiedad “Filter” y el valor “C:” a la propiedad “InitialDirectory”:



Finalmente, cuando el usuario escriba el nombre del archivo y presione el botón “Guardar” el diálogo se cerrará y podremos recuperar la ruta y el nombre del archivo desde la propiedad “FileName”, por ejemplo:

```
saveFileDialog1.Filter = "Archivos de texto (*.txt) |*.txt| Archivos de reporte (*.rpt)|*.rpt";
saveFileDialog1.FilterIndex = 0;
saveFileDialog1.Title = "Guardar Reporte";
saveFileDialog1.RestoreDirectory = true;
saveFileDialog1.InitialDirectory = "C:";
if(saveFileDialog1.ShowDialog() == DialogResult.OK )
{
    string ruta = saveFileDialog1.FileName;
    MessageBox.Show("Archivo de reporte: " + ruta);
}
```

Guardamos en la variable “ruta” el valor de la propiedad “FileName” con la ubicación y el nombre del archivo ingresado en el diálogo, podría ser, por ejemplo, “C:/Datos/Reporte.txt”, donde “C:/Datos/” es la ruta elegida y “Reporte.txt” es el nombre ingresado.

De forma similar podemos trabajar con el diálogo “OpenFileDialog” que posee las mismas propiedades para establecer el título, los filtros, el directorio inicial, etc.

1. Indique la opción correcta:

Con OpenFileDialog podemos establecer la ubicación y el nombre del archivo a grabar.

- Verdadero
- Falso

2. Indique la opción correcta:

Con la propiedad “FileName” obtenemos el nombre del archivo seleccionado en un cuadro de tipo OpenFileDialog.

- Verdadero
- Falso

3. Indique la opción correcta.

La propiedad .ShowHelp=true nos permite ver el botón “Ayuda” además de los botones “Aceptar” y “Cancelar”.

- Verdadero
- Falso

4. Indique la opción correcta:

Indique la opción correcta. La propiedad Filter = "Archivos gráficos (*.jpg)|*.jpg" nos listará en la ventana todos los archivos de gráficos, sin importar su extensión.

- Verdadero
- Falso

5. Indique la opción correcta:

Al cerrar el diálogo de tipo SaveFileDialog con el botón “Guardar” el valor devuelto es:

- a. DialogResult.OK
- b. DialogResult.Cancel
- c. DialogResult.Save

Respuestas correctas⁴

⁴1) Falso. 2) Verdadero, 3) Verdadero. 4) Falso 5) a.

SP1/H5 Trace, EventLogs

El término “Trace” se refiere al rastreo o seguimiento que podemos hacer del funcionamiento de nuestra aplicación mientras se está ejecutando, “EventLogs” por su parte nos permite interactuar con los logs de eventos del sistema operativo Windows. Así podemos hacer un seguimiento de comportamiento de la aplicación, detectar errores y/o problemas de performance.

Para acceder a la clase “Trace” y sus métodos debemos agregar a nuestro código la directiva:

```
using System.Diagnostics;
```

También podemos configurar las propiedades “autoflush” y “indentsize” desde el archivo XML de configuración de la aplicación (App.config). “autoflush” define si los datos se escriben automáticamente en la salida definida de Trace o se espera hasta que se escriban en forma explícita, “indentsize” por su parte, indica la cantidad de espacios de tabulación que se agregaron a cada salida del trace. Esto se define así:

```
<configuration>
  <system.diagnostics>
    <trace autoflush="false" indentsize="3" />
  </system.diagnostics>
</configuration>
```

Y se agregan al archivo App.config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
  <system.diagnostics>
    <trace autoflush="false" indentsize="3" />
  </system.diagnostics>
</configuration>
```

Para escribir información en la salida de Trace disponemos de varios métodos:

- **Write**
- **WriteLine**
- **Writelf**
- **WriteLinelf**

Write escribe el texto pasado por parámetro.

WriteLine escribe el texto y agrega un salto de línea.

Writelf escribe el texto sólo si se cumple la condición establecida como primer parámetro.

WriteLinelf igual que el anterior, pero agrega un salto de línea al final del texto.

Ejemplo:

```
using System;
using System.Diagnostics;

class Test
{
    static void Main()
    {
        Trace.AutoFlush = true;
        Trace.Indent();
        Trace.WriteLine("Iniciando Main");
        Trace.Unindent();
    }
}
```

Al ejecutar este ejemplo veremos en la consola de salida de Visual Studio el texto “Iniciando Main” separado del margen izquierdo con una tabulación.

Trace Listener

Un “Listener” representa un objeto que estará escuchando permanentemente los mensajes que envía el objeto Trace para procesarlos de acuerdo a su funcionalidad. Por defecto los mensajes escritos por Trace se dirigen a la consola de salida (DefaultTraceListener), pero es posible agregar otros listeners:

- **FileLogTraceListener**
- **EventProviderTraceListener**
- **EventLogTraceListener**
- **TextWriterTraceListener**
- **IisTraceListener**
- **WebPageTraceListener**

Uno o más de estos Listeners se pueden agregar como salida del Trace:

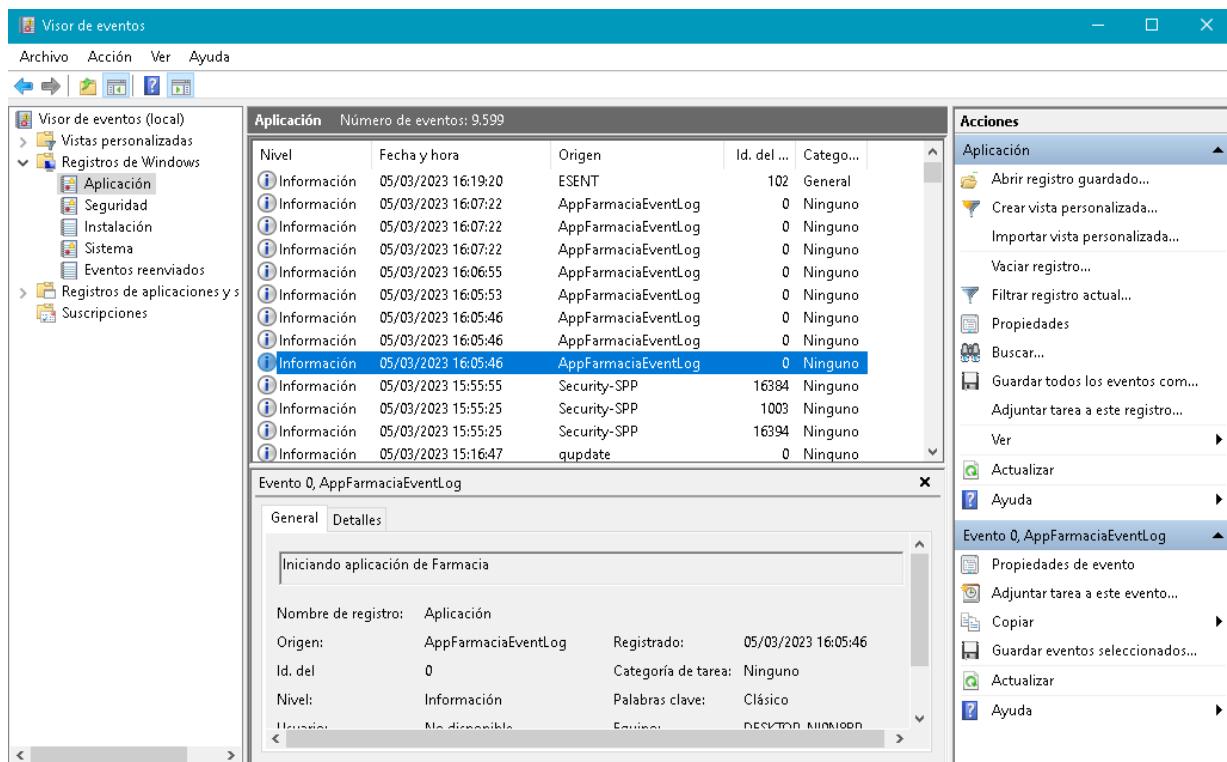
```
Trace.Listeners.Add(new EventLogTraceListener("Nombre_Aplicacion"));
```

“Nombre_Aplicacion” se usará para identificar los mensajes escritos por Trace.

Trace y EventLog

Si se configura el listener “EventLogTraceListener” como salida de Trace entonces se podrán registrar todos los eventos que genere nuestra aplicación en el gestor de eventos de Windows para poder consultarlos posteriormente.

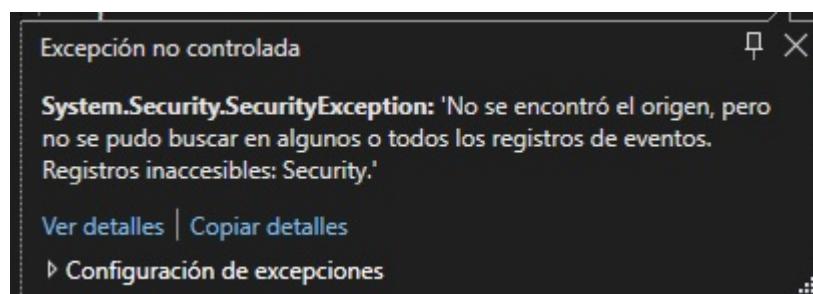
Los registros de todos los eventos de Windows se pueden visualizar con la aplicación “Visor de Eventos”:



Esto resulta particularmente interesante para mantener un registro de lo que ocurre durante la ejecución de una aplicación, revisar fechas y horas de cada evento y determinar posibles errores o problemas que presenta la aplicación. Por ejemplo, si un procedimiento demora mucho tiempo en completarse podremos controlarlo fácilmente desde el visor de eventos y así buscar las posibles causas de la demora.

Permisos de uso de EventLog

El gestor de eventos de Windows requiere permisos de administrador para poder ser ejecutado sin errores, por lo que si la aplicación es ejecutada por un usuario no administrador se obtendrá un error indicando un problema de seguridad:



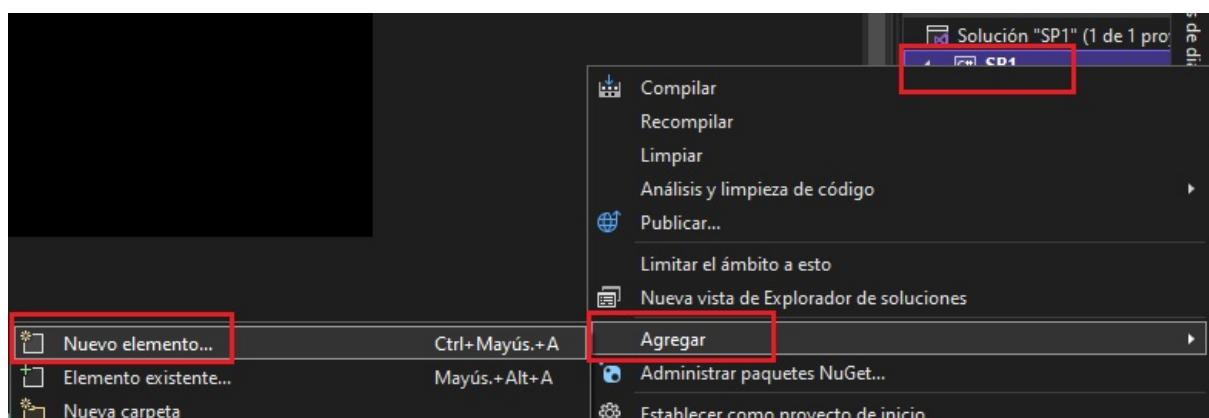
Sólo un usuario administrador puede hacer uso de EventLog

Para solucionar este problema hay que agregar al proyecto un archivo de manifiesto que permita ejecutar la aplicación a un usuario común con permisos de administrador. Los pasos a seguir son:

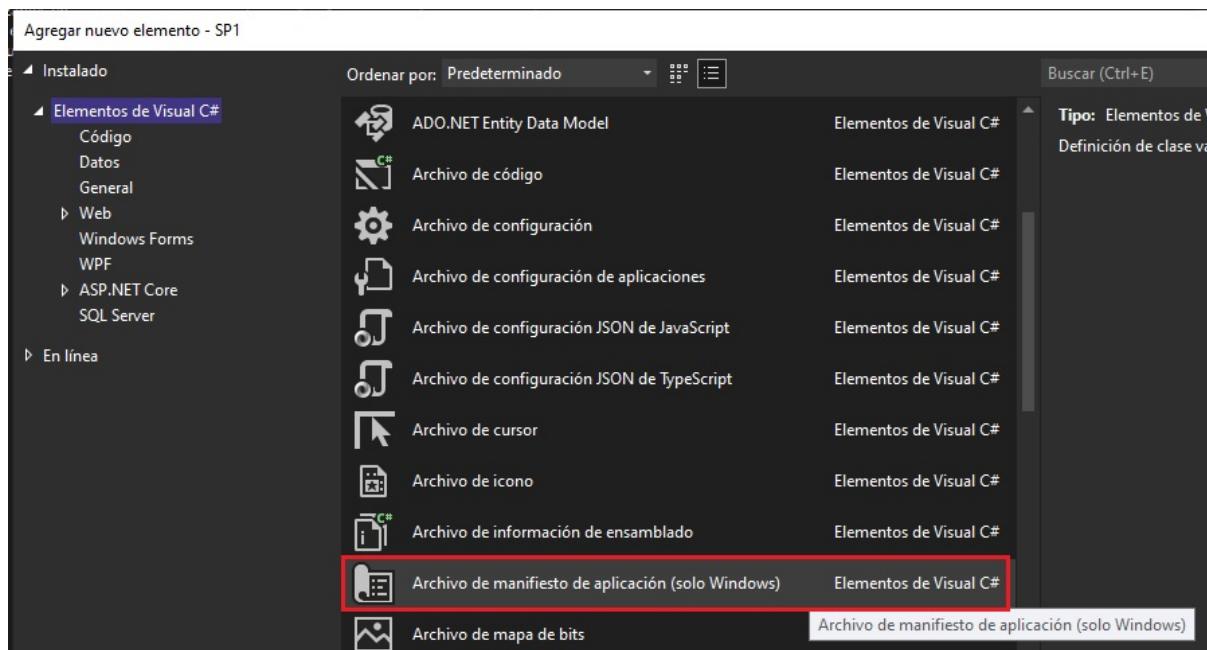
- 1- Agregar el archivo de manifiesto al proyecto.
- 2- Editar el archivo de manifiesto y modificar el permiso
- 3- Actualizar las propiedades del proyecto para que use el archivo de manifiesto creado

Agregar el archivo de Manifiesto al proyecto de Visual Studio:

Desde la ventana del Explorador de la Solución, sobre el nombre del proyecto, clic derecho del mouse y seleccionamos “Aregar” – “Nuevo elemento”:



Luego, en la ventana que se abre buscamos la opción “Archivo de manifiesto de aplicación (sólo Windows)” y lo agregamos al proyecto:



El archivo agregado lleva por nombre “app.manifest”.

Nos mostrará una ventana con su contenido (código xml):

```

app.manifest ✘ X Form1.cs*
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- Opciones del manifiesto UAC
            Si quiere cambiar el nivel del Control de cuentas de usuario de Windows reemplace
            nodo requestedExecutionLevel por uno de los siguientes.

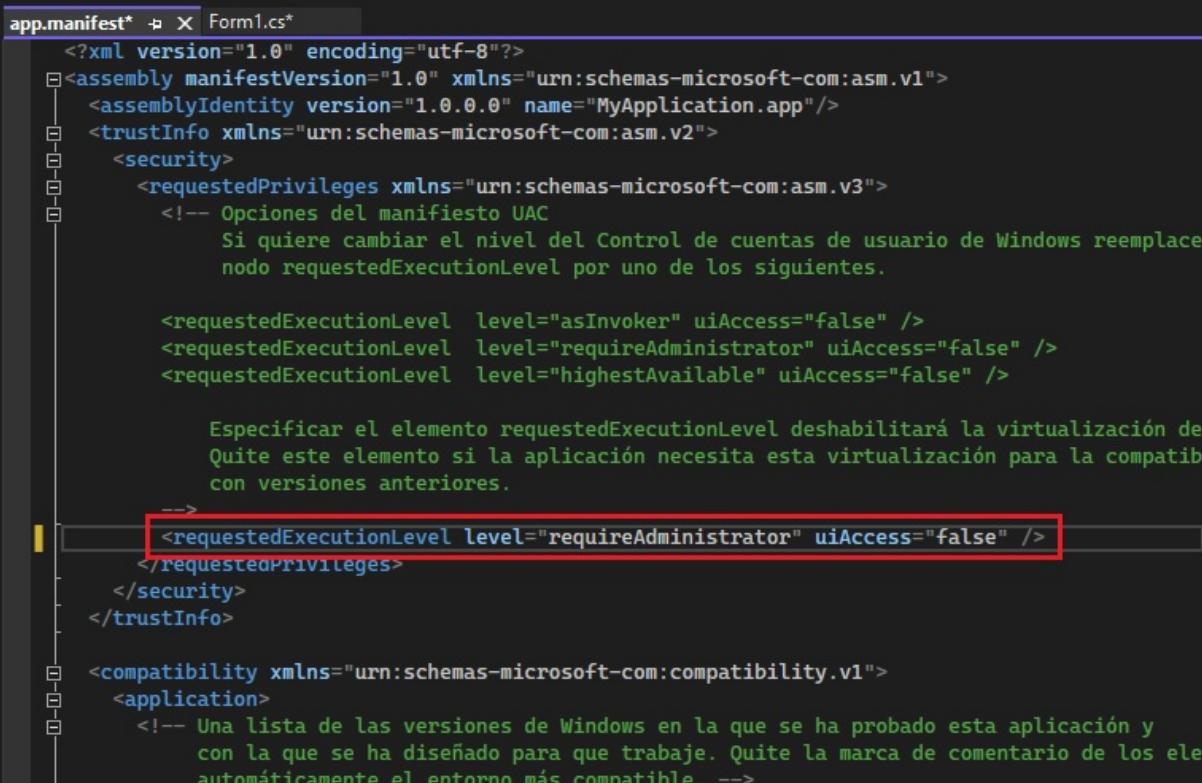
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
        <requestedExecutionLevel level="highestAvailable" uiAccess="false" />

        Especificar el elemento requestedExecutionLevel deshabilitará la virtualización de
        Quite este elemento si la aplicación necesita esta virtualización para la compatibilidad
        con versiones anteriores.
        -->
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>

  <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
    <application>
      <!-- Una lista de las versiones de Windows en la que se ha probado esta aplicación y
          con la que se ha diseñado para que trabaje. Quite la marca de comentario de los elementos
          automáticamente el entorno más compatible. -->
    </application>
  </compatibility>
</assembly>
```

La línea marcada es la que define el nivel de permisos con que se ejecuta la aplicación y el valor “asInvoker” es el que deberemos modificar.

Cambiamos “asInvoker” por “RequireAdministrator”, y resultará así:



```
app.manifest* ▾ X Form1.cs*
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
    <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
        <security>
            <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
                <!-- Opciones del manifiesto UAC
                    Si quiere cambiar el nivel del Control de cuentas de usuario de Windows reemplace
                    nodo requestedExecutionLevel por uno de los siguientes.

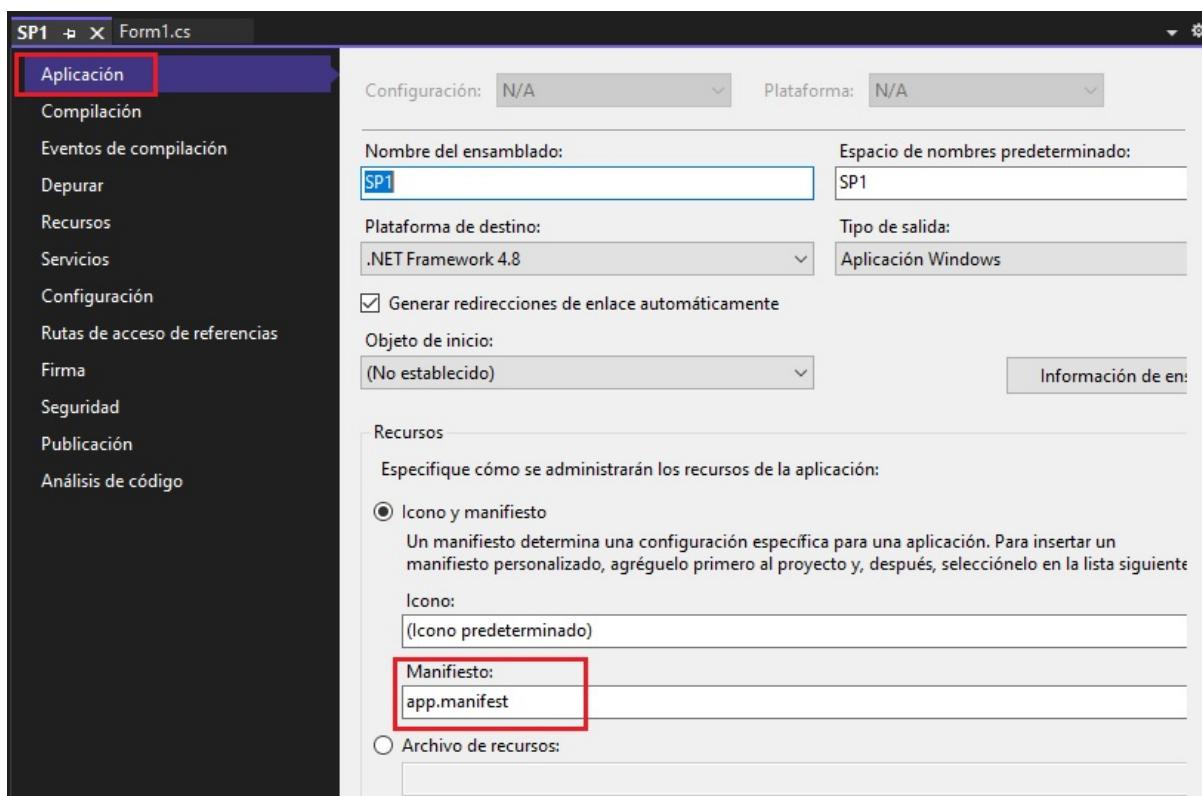
                <requestedExecutionLevel level="asInvoker" uiAccess="false" />
                <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
                <requestedExecutionLevel level="highestAvailable" uiAccess="false" />

                    Especificar el elemento requestedExecutionLevel deshabilitará la virtualización de
                    Quite este elemento si la aplicación necesita esta virtualización para la compatib
                    con versiones anteriores.
                -->
                <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
            
```

Guardamos los cambios en el archivo y lo cerramos.

A continuación, debemos abrir la ventana de propiedades del proyecto, desde el menú principal de Visual Studio: opción “Proyecto”, luego “Propiedades de ...” y se mostrará esta ventana:

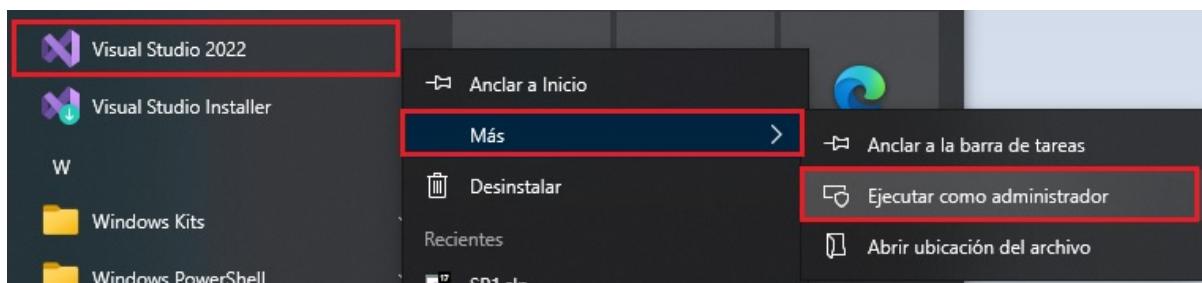
En la solapa “Aplicación” tenemos:



Debemos verificar que se muestre el archivo "app.manifest" en la casilla correspondiente al manifiesto, si no está asignado debemos agregarlo en forma manual.

Guardamos los cambios y cerramos la ventana.

Con esos cambios el usuario podrá ejecutar la aplicación sin errores. Para verificar esto desde el mismo Visual Studio, se deberá iniciar su ejecución como administrador:



Clic con el botón derecho del mouse sobre "Visual Studio 2022", luego elegimos "Más" y luego "Ejecutar como administrador". Así Visual Studio tendrá los permisos necesarios para acceder al uso de EventLog.

Tipos de registros con Trace

Además de escribir información simple con los métodos “Write”, la clase Trace posee métodos para definir el nivel del mensaje generado:

- TraceInformation: usado para información general de la aplicación
- TraceWarning: usado para generar mensajes de advertencia.
- TraceError: usado para mensajes de tipo error.

Ejemplo:

Aplicación Número de eventos: 9.621					
Nivel	Fecha y hora	Origen	Id. del ...	Catego...	
⚠ Advertencia	02/03/2023 19:04:46	Security-SPP	8233	Ninguno	
ℹ Información	02/03/2023 19:04:46	SecurityCenter	15	Ninguno	
⚠ Advertencia	02/03/2023 19:04:42	Security-SPP	8233	Ninguno	
ℹ Información	02/03/2023 19:04:38	Windows Error Reporting	1001	Ninguno	
ℹ Información	02/03/2023 19:04:37	Windows Error Reporting	1001	Ninguno	
ℹ Información	02/03/2023 19:04:37	Windows Error Reporting	1001	Ninguno	
⚠ Advertencia	02/03/2023 19:04:36	Security-SPP	8233	Ninguno	
❗ Error	02/03/2023 19:04:36	Application Error	1000 (100)		
❗ Error	02/03/2023 19:04:36	.NET Runtime	1026	Ninguno	
ℹ Información	02/03/2023 19:04:36	gupdate	0	Ninguno	
ℹ Información	02/03/2023 19:04:32	Winlogon	6000	Ninguno	
ℹ Información	02/03/2023 19:04:32	Windows Error Reporting	1001	Ninguno	
ℹ Información	02/03/2023 19:04:31	NvStreamSvc	2003	Ninguno	

En el recuadro se observan los distintos tipos de Nivel generados: Información, Advertencia y Error.

También podemos agregar un Listener para que la salida sea un archivo de texto:

```
Trace.Listeners.Add(new TextWriterTraceListener("Nombre_Archivo"));
```

Donde “Nombre_Archivo” puede incluir la ruta absoluta donde se ubicará el archivo creado, si sólo se especifica el nombre del archivo, éste se ubicará en la misma carpeta que contiene el archivo ejecutable (.exe) de la aplicación.

Para más detalles de las propiedades y métodos de la clase Trace puede consultar este enlace:

[https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.trace?view=netframewor](https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.trace?view=netframework-4.8.1)k-4.8.1

También puede revisar la información sobre la clase EventLog en este enlace:

[https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.eventlog?view=netframewor](https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.eventlog?view=netframework-4.8.1)k-4.8.1

SP1/Autoevaluación 5

1. Indique la opción correcta:

Para incluir la clase Trace en el código de la aplicación se debe agregar la cláusula “using system.Diagnostics”.

- Verdadero
- Falso

2. Indique la opción correcta:

La clase Trace dispone de métodos para grabar y leer información, por ejemplo, Write() y Read().

- Verdadero
- Falso

3. Indique la opción correcta:

En la clase Trace, el método WriteLine() permite escribir un texto y agrega un salto de línea al final.

- Verdadero
- Falso

4. Indique la opción correcta:

El listener por defecto de Trace es la consola de salida.

- Verdadero
- Falso

5. Indique la opción correcta:

Para visualizar los registros de Trace en el Visor de Eventos de Windows se debe usar el listener **EventProviderTraceListener**.

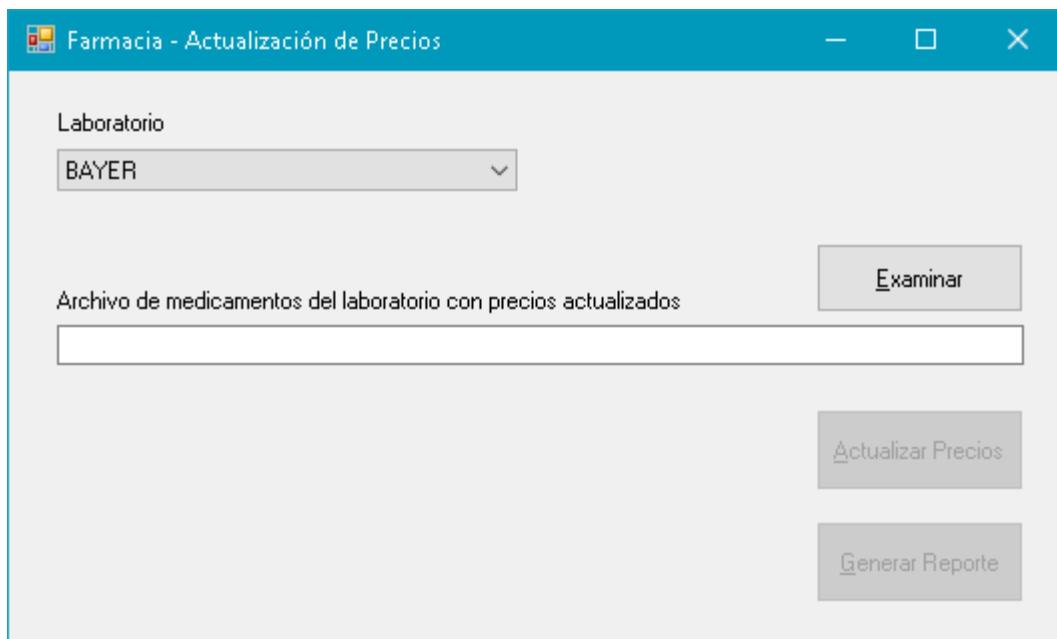
- Verdadero
- Falso

Respuestas correctas⁵

⁵1) Verdadero. 2) Falso. 3) Verdadero. 4) Verdadero. 5) Falso.

SP1/Ejercicio resuelto

El diseño del formulario es realmente simple:



Tenemos un control comboBox, un textBox, Labels y botones de comando. Una vez completado el diseño del formulario pasamos a trabajar con el código de la aplicación. Trataremos de aplicar los conceptos de POO ya vistos anteriormente para crear las clases necesarias con sus correspondientes propiedades y métodos.

Como queremos tener excepciones personalizadas para el tratamiento de posibles errores en las clases que manejan los datos de las tablas vamos a definir las clases “LaboratorioException” y “ActualizacionException”, cuyos contenidos serán muy simples, nos limitaremos a definir los 3 constructores recomendados en la documentación:

Así la clase “LaboratorioException” tendrá este contenido:

```
• LaboratorioException()
• LaboratorioException(string)
• LaboratorioException(string, System.Exception)
```

El código completo de la clase es este:

```

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

namespace SP1
{
    public class LaboratorioException: Exception
    {
        //constructores
        public LaboratorioException() : base() {}
        public LaboratorioException(String message): base(message) {}
        public LaboratorioException(String message, Exception inner) :
base(message, inner) { }
    }
}

```

De forma similar, la clase “ActualizacionException” tendrá también los 3 constructores:

- ⊕ ActualizacionException()
- ⊕ ActualizacionException(string)
- ⊕ ActualizacionException(string, System.Exception)

Su código:

```

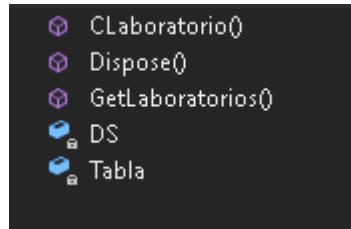
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SP1
{
    public class ActualizacionException: Exception
    {
        public ActualizacionException() : base() { }
        public ActualizacionException(String message): base(message) { }
        public ActualizacionException(String message, Exception ex) : base(message,
ex) { }
    }
}

```

Clase “CLaboratorio”: permitirá conectarse a la base de datos y acceder a la tabla “Laboratorios”, se necesitan solamente operaciones de lectura sobre la tabla.

El contenido de la clase será:



Tendrá el método constructor, los métodos Dispose() y GetLaboratorios(), además de dos elementos privados : el DataSet “DS” y un string “Tabla” para mantener el nombre de la tabla.

El código completo de “CLaboratorio” es:

```

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.IO;

namespace SP1
{
    public class CLaboratorio
    {
        DataSet DS;
        String Tabla = "Laboratorios";

        public CLaboratorio()
        {
            try
            {
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Farmacia.mdb";
                cnn.Open();
                DS = new DataSet();
                OleDbCommand cmd = new OleDbCommand();
                cmd.Connection = cnn;
                cmd.CommandType = CommandType.TableDirect;
                cmd.CommandText = Tabla;
                OleDbDataAdapter DA = new OleDbDataAdapter(cmd);
                DA.Fill(DS, Tabla);
                DataColumn[] pk = new DataColumn[1];
                pk[0] = DS.Tables[Tabla].Columns["Laboratorio"];
                DS.Tables[Tabla].PrimaryKey = pk;
                OleDbCommandBuilder cb = new OleDbCommandBuilder(DA);
                cnn.Close();
            }
            catch(Exception ex)
            {
                String MsgErr = "CLaboratorio: " + ex.Message;
                throw new LaboratorioException(MsgErr);
            }
        }
    }
}

```

```

    }

    public DataTable GetLaboratorios()
    {
        if(DS != null && DS.Tables.Count == 1)
        {
            return DS.Tables["Laboratorios"];
        }
        return null;
    }

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

```

En el constructor de la clase se realiza la conexión con la base de datos y se carga el contenido de la tabla “Laboratorios” en el objeto data set “DS”, también se define la propiedad de la clave primaria. En el bloque try-catch se capturan las excepciones y si algo falla se disparará una nueva excepción de tipo “LaboratorioException” con el comando “throw”. El texto que se genera en esta excepción lleva como prefijo el nombre de la clase “CLaboratorio” seguido del texto original de la excepción capturada. De esa forma será más fácil de reconocer en los logs a qué clase pertenecen las excepciones generadas.

El método “GetLaboratorios” devuelve el contenido completo de la tabla “Laboratorios” cargada previamente en el dataset “DS”.

El método “Dispose” se encarga de liberar los recursos del dataset “DS” una vez que la instancia de esta clase ya no se necesita más.

Continuamos el desarrollo del código con la clase “CActualización”, cuyo contenido queda definido así:

```
    ↗ Actualizar(string, string, int)
    ↗ ActualizarMedicamento(string, string, float, int, System.DateTime)
    ↗ CActualizacion()
    ↗ Dispose()
    ↗ GenerarReporte(string)
    ↘ DALab
    ↘ DAMed
    ↘ DS
    ↘ TablaLab
    ↘ TablaMed
```

Esta clase será responsable, por medio de su constructor, de leer las tablas de Laboratorios y Medicamentos, definir las claves primarias y crear los objetos de tipo CommandBuilder para poder grabar los cambios necesarios.

El método “Actualizar” es el proceso principal de actualización que se encargará de leer los datos del archivo seleccionado en la interfaz, cada medicamento que forme parte del archivo se procesa con el método “ActualizarMedicamento”, el que se encargará de determinar si es un nuevo medicamento o uno ya existente.

El método “GenerarReporte” tiene por finalidad crear el archivo de texto con el reporte de todos los medicamentos existentes en la tabla de la base de datos.

Finalmente, el método “Dispose” libera los recursos empleados por los objetos DataSet y DataAdapter empleados en cada instancia de la clase.

El código completo de la clase resulta así:

```
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.IO;
using System.Globalization;
using System.Diagnostics;

namespace SP1
{
    public class CActualizacion
    {
        DataSet DS;
```

```

OleDbDataAdapter DALab;
OleDbDataAdapter DAMed;
String TablaLab = "Laboratorios";
String TablaMed = "Medicamentos";

public CActualizacion()
{
    try{
        OleDbConnection cnn = new OleDbConnection();
        cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Farmacia.mdb";
        cnn.Open();
        DS = new DataSet();
        OleDbCommand cmd = new OleDbCommand();
        cmd.Connection = cnn;
        cmd.CommandType = CommandType.TableDirect;
        cmd.CommandText = TablaLab;
        DALab = new OleDbDataAdapter(cmd);
        DALab.Fill(DS, TablaLab);
        DataColumn[] pk = new DataColumn[1];
        pk[0] = DS.Tables[TablaLab].Columns["Laboratorio"];
        DS.Tables[TablaLab].PrimaryKey = pk;
        OleDbCommandBuilder cbLab = new OleDbCommandBuilder(DALab);
        //
        cmd = new OleDbCommand();
        cmd.Connection = cnn;
        cmd.CommandType = CommandType.TableDirect;
        cmd.CommandText = TablaMed;
        DAMed = new OleDbDataAdapter(cmd);
        DAMed.Fill(DS, TablaMed);
        DataColumn[] pkM = new DataColumn[1];
        pkM[0] = DS.Tables[TablaMed].Columns["Codigo"];
        DS.Tables[TablaMed].PrimaryKey = pkM;
        OleDbCommandBuilder cbMed = new OleDbCommandBuilder(DAMed);
        cnn.Close();
    }
    catch (Exception ex)
    {
        String MsgErr = "CActualizacion: " + ex.Message ;
        throw new ActualizacionException(MsgErr);
    }
}

public bool Actualizar(String path, String NombreLaboratorio, int
laboratorio)
{
    bool resultado = false;
    String linea = "";
    try
    {
        // abrir el archivo de medicamentos
        StreamReader sr = new StreamReader(path);
        String encabezado = sr.ReadLine();
        String Lab = encabezado.Split('(')[0];
        if(Lab.CompareTo(NombreLaboratorio) != 0)
        {
            throw new ActualizacionException("El archivo seleccionado no
corresponde al Laboratorio " + NombreLaboratorio);
    }
}

```

```

        }
        String Fecha = encabezado.Split('(')[1].Split(')')[0];

        while (sr.EndOfStream == false)
        {
            linea = sr.ReadLine();
            //
            String[] campos = linea.Split(',');
            if(campos.Length == 3)
            {
                String codigo = campos[0];
                String nombre = campos[1].Trim("'");
                Single precio = Single.Parse(campos[2],
CultureInfo.InvariantCulture);
                //
                ActualizarMedicamento(codigo, nombre, precio, laboratorio,
DateTime.Parse(Fecha));
            }
        }
        sr.Close();
        sr.Dispose();
        resultado = true;
    }
    catch(ActualizacionException aex)
    {
        Trace.WriteLineIf(linea != "", "Ultima lectura: " + linea);
        throw new ActualizacionException("Error actualizando archivo de
Medicamentos: " + aex.Message);
    }
    return resultado;
}

public void ActualizarMedicamento(String codigo, String nombre, Single
precio, int laboratorio, DateTime fecha)
{
    try
    {
        DataRow drM = DS.Tables[TablaMed].Rows.Find(codigo);
        if (drM != null)
        {
            // contolar la fecha
            if (fecha.CompareTo(drM["Fecha"]) >= 0 )
            {
                // ya existe, se actualiza
                drM.BeginEdit();
                drM["Nombre"] = nombre;
                drM["Precio"] = precio;
                drM["Fecha"] = fecha;
                drM.EndEdit();
                DAMed.Update(DS, TablaMed);
                Trace.WriteLine("Medicamento actualizado " + nombre);
            }
            else
            {
                Trace.WriteLine("Medicamento con fecha anterior " +
nombre);
            }
        }
        else
    }
}

```

```

        {
            // no existe, se agrega
            DataRow drNuevo = DS.Tables[TablaMed].NewRow();
            drNuevo[ "Codigo" ] = long.Parse(codigo);
            drNuevo[ "Nombre" ] = nombre;
            drNuevo[ "Precio" ] = precio;
            drNuevo[ "Laboratorio" ] = laboratorio;
            drNuevo[ "Fecha" ] = fecha;
            DS.Tables[ "Medicamentos" ].Rows.Add(drNuevo);
            DAMed.Update(DS, TablaMed);
            Trace.WriteLine("Medicamento agregado " + nombre);
        }
    }
    catch(Exception ex)
    {
        String MsgErr = "CActualizacion: " + ex.Message;
        throw new ActualizacionException(MsgErr);
    }
}

public void GenerarReporte(String NombreArchivo)
{
    // Lab
    try
    {
        StreamWriter sw = new StreamWriter(NombreArchivo);
        sw.WriteLine("Reporte de Medicamentos Actualizados");
        sw.WriteLine("-----");
        sw.WriteLine("    Código Nombre");
        foreach (DataRow dr in DS.Tables[TablaMed].Rows)
        {
            String linea = dr[ "Codigo" ].ToString().PadLeft(10) + " ";
            linea += dr[ "Nombre" ].ToString().PadRight(30);
            linea += dr[ "Laboratorio" ].ToString().PadLeft(6);
            linea += String.Format("{0,12:F2}", dr[ "Precio" ]);
            linea += string.Format("{0:d}", dr[ "Fecha" ]).PadLeft(15);
            sw.WriteLine(linea);
        }
        sw.WriteLine("");
        sw.WriteLine("Fecha del reporte: " +
DateTime.Now.ToString("dd/MM/yyyy"));
        sw.Close();
        sw.Dispose();
        Trace.WriteLine("Reporte generado");
    }
    catch(Exception ex)
    {
        String MsgErr = "CActualizacion: " + ex.Message;
        throw new ActualizacionException(MsgErr);
    }
}

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

```

```
}
```

Algunas observaciones para tener en cuenta

En el constructor, el manejo de las posibles excepciones se realiza lanzando una excepción de tipo “ActualizacionException”. Algo similar ocurre en los restantes métodos de la clase.

En los métodos “Actualizar”, “ActualizarMedicamento” y “GenerarReporte” se incluye el uso de la clase “Trace”, escribiendo mensajes informativos sobre la ejecución de cada uno de los métodos. Por ejemplo:

```
Trace.WriteLine("Medicamento actualizado " + nombre);
```

Para poder usar la clase “Trace” se incluye la directiva using:

```
using System.Diagnostics;
```

Y para poder hacer uso de las clases Stream se agrega la directiva:

```
using System.IO;
```

En el método “Actualizar”, al leer los datos del archivo de medicamentos, y obtener el valor del campo “precio”, hay que especificar el valor de “CultureInfo” para lograr una conversión correcta de los valores con parte decimal:

```
Single precio = Single.Parse(campos[2], CultureInfo.InvariantCulture);
```

En el método “GenerarReporte”, para conformar el contenido de la línea de texto que se debe grabar en el reporte, se trabaja con los métodos “PadLeft”, “PadRight” y “String.Format”. Los métodos “Pad...”, permiten establecer un ancho fijo para cada campo, alineando el valor a mostrar a la derecha o a la izquierda según se trate de campos de tipo numérico o de texto. El método “String.Format” se usa para asignar formatos específicos como, por ejemplo:

```
String.Format("{0,12:F2}", dr["Precio"]); // float con 2 decimales
```

O para formatos de fecha:

```
string.Format("{0:d}", dr["Fecha"]) // formato fecha: dd/mm/yyyy
```

Para más detalles de los formatos admitidos por el método “Format” puede consultar este enlace:

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

Una vez finalizada la implementación de estas 4 clases resta hacer uso de las mismas desde el código del formulario, código que resultará bastante simple ya que toda la lógica de acceso a datos y manejo de archivos está implementada en estas clases.

Veamos entonces el código del formulario:

En primer lugar, debemos agregar la directiva using para poder trabajar con la clase Trace:

```
using System.Diagnostics; // permite usar Trace
```

Seguidamente revisamos el código del evento “Load”:

```
private void Form1_Load(object sender, EventArgs e)
{
    // controlar si la aplicación ya está registrada en EventLog
    if (!EventLog.SourceExists("AppFarmaciaEventLog"))
    {
        // si no está registrada, se agrega como nuevo Listener para Trace
        Trace.Listeners.Add(new
EventLogTraceListener("AppFarmaciaEventLog"));
    }
    Trace.AutoFlush = true;
    Trace.Indent();
    Trace.WriteLine("Iniciando aplicación de Farmacia");
    try
    {
        CLaboratorio lab = new CLaboratorio();
        cmbLaboratorio.DisplayMember = "Nombre";
        cmbLaboratorio.ValueMember = "Laboratorio";
        cmbLaboratorio.DataSource = lab.GetLaboratorios();
        lab.Dispose();
        Trace.WriteLine("Datos de Laboratorios obtenidos " +
cmbLaboratorio.Items.Count.ToString());
    }
    catch(LaboratorioException lex)
    {
        MessageBox.Show(lex.Message);
    }
}
```

Acá se agrega el listener “EventLogTraceListener” para obtener las salidas de Trace en el visor de eventos de Windows, luego se crea una instancia de la clase “CLaboratorio” y se ejecuta el método “GetLaboratorios” para vincular la tabla al control ComboBox de Laboratorio. La captura de excepciones se realiza con la clase propia “LaboratorioException”.

Continuamos con el código del botón “Examinar”:

```
private void btnExaminar_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();
    dlg.Title = "Archivo de Medicamentos";
    dlg.Filter = "Archivos txt|*.txt";
    dlg.FilterIndex = 0;
    dlg.InitialDirectory = Application.StartupPath;
    dlg.RestoreDirectory = true;
    if(dlg.ShowDialog() == DialogResult.OK)
    {
        txtArchivo.Text = dlg.FileName;
        btnActualizar.Enabled = true;
        btnReporte.Enabled = true;
        Trace.WriteLine("Se seleccionó el archivo " + dlg.FileName);
    }
    else
    {
        txtArchivo.Text = "";
        btnActualizar.Enabled = false;
        btnReporte.Enabled = false;
    }
}
```

Este botón de comando permite buscar y seleccionar el archivo de medicamentos que se usará para actualizar los medicamentos, se crea por código el objeto “OpenFileDialog”, se establecen sus principales propiedades y se visualiza ejecutando el método “ShowDialog”, si la respuesta es OK el nombre del archivo seleccionado se asigna al control TextBox y se habilitan los demás botones de comando. Con Trace se informa el nombre del archivo seleccionado.

Código del botón “Actualizar”:

```
private void btnActualizar_Click(object sender, EventArgs e)
{
    try
    {
        CActualizacion actualizacion = new CActualizacion();
        bool res = actualizacion.Actualizar(txtArchivo.Text,
            cmbLaboratorio.Text,
```

```

        int.Parse(cmbLaboratorio.SelectedValue.ToString())));
    if (res)
    {
        MessageBox.Show("Actualización finalizada correctamente");
    }
    actualizacion.Dispose();
}
catch(ActualizacionException aex)
{
    MessageBox.Show(aex.Message);
}
}
}

```

En este procedimiento se crea una nueva instancia de la clase “CActualizacion”, luego se ejecuta el método “Actualizar” y se pasa por parámetro el nombre del archivo que contiene los medicamentos, el nombre del laboratorio seleccionado para la actualización, y el número del laboratorio seleccionado. El método “Actualizar” realizará su tarea y devolverá verdadero si finaliza correctamente, en caso contrario generará una excepción.

Código del botón “GenerarReporte”:

```

private void btnReporte_Click(object sender, EventArgs e)
{
    SaveFileDialog dlg = new SaveFileDialog();
    dlg.Title = "Guardar Reporte";
    dlg.Filter = "Archivos de Reporte (.txt)|*.txt";
    dlg.FilterIndex = 0;
    dlg.InitialDirectory = Application.StartupPath;
    if(dlg.ShowDialog() == DialogResult.OK)
    {
        try
        {
            CActualizacion actualizacion = new CActualizacion();
            actualizacion.GenerarReporte(dlg.FileName);
            MessageBox.Show("Reporte Generado. (" + dlg.FileName + ")");
            actualizacion.Dispose();
            Trace.WriteLine("Reporte generado: " + dlg.FileName);
        }
        catch(ActualizacionException aex)
        {
            MessageBox.Show(aex.Message);
        }
        finally
        {
            dlg.Dispose();
        }
    }
}

```

Para generar el reporte solicitado usamos un objeto de la clase “SaveFileDialog” con el que el usuario seleccionará la ubicación y el nombre del archivo que contendrá el reporte, seguidamente se crea una instancia de la clase “CActualizacion” y se invoca el método “GenerarReporte” pasando por parámetro el nombre del archivo a crear.

Código del evento “SelectedIndexChanged” del control ComboBox:

```
private void cmbLaboratorio_SelectedIndexChanged(object sender, EventArgs e)
{
    Trace.WriteLine("Se seleccionó el laboratorio " +
cmbLaboratorio.Text);
    txtArchivo.Text = "";
    btnActualizar.Enabled = false;
    btnReporte.Enabled = false;
}
```

En este evento, que se ejecuta cada vez que el usuario selecciona otro laboratorio de la lista desplegable, se actualiza el estado del TextBox y los botones de comando, también se hace uso de Trace para informar el laboratorio seleccionado.

El código completo del formulario queda de esta forma:

```
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.Diagnostics; // permite usar Trace

namespace SP1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // controlar si la aplicación ya está registrada en EventLog
            if (!EventLog.SourceExists("AppFarmaciaEventLog"))
            {
                // si no está registrada, se agrega como nuevo Listener para Trace
                Trace.Listeners.Add(new EventLogTraceListener("AppFarmaciaEventLog"));
            }
        }
    }
}
```

```

        }
        Trace.AutoFlush = true;
        Trace.Indent();
        Trace.WriteLine("Iniciando aplicación de Farmacia");
        try
        {
            CLaboratorio lab = new CLaboratorio();
            cmbLaboratorio.DisplayMember = "Nombre";
            cmbLaboratorio.ValueMember = "Laboratorio";
            cmbLaboratorio.DataSource = lab.GetLaboratorios();
            lab.Dispose();
            Trace.WriteLine("Datos de Laboratorios obtenidos " +
cmbLaboratorio.Items.Count.ToString());
        }
        catch(LaboratorioException lex)
        {
            MessageBox.Show(lex.Message);
        }
    }

    private void btnActualizar_Click(object sender, EventArgs e)
    {
        try
        {
            CActualizacion actualizacion = new CActualizacion();
            bool res = actualizacion.Actualizar(txtArchivo.Text,
                cmbLaboratorio.Text,
                int.Parse(cmbLaboratorio.SelectedValue.ToString()));
            if (res)
            {
                MessageBox.Show("Actualización finalizada correctamente");
            }
            actualizacion.Dispose();
        }
        catch(ActualizacionException aex)
        {
            MessageBox.Show(aex.Message);
        }
    }

    private void btnExaminar_Click(object sender, EventArgs e)
    {
        OpenFileDialog dlg = new OpenFileDialog();
        dlg.Title = "Archivo de Medicamentos";
        dlg.Filter = "Archivos txt|*.txt";
        dlg.FilterIndex = 0;
        dlg.InitialDirectory = Application.StartupPath;
        dlg.RestoreDirectory = true;
        if(dlg.ShowDialog()== DialogResult.OK)
        {
            txtArchivo.Text = dlg.FileName;
            btnActualizar.Enabled = true;
            btnReporte.Enabled = true;
            Trace.WriteLine("Se seleccionó el archivo " + dlg.FileName);
        }
        else
        {
            txtArchivo.Text = "";
            btnActualizar.Enabled = false;
        }
    }
}

```

```

        btnReporte.Enabled = false;
    }

private void btnReporte_Click(object sender, EventArgs e)
{
    SaveFileDialog dlg = new SaveFileDialog();
    dlg.Title = "Guardar Reporte";
    dlg.Filter = "Archivos de Reporte (.txt)|*.txt";
    dlg.FilterIndex = 0;
    dlg.InitialDirectory = Application.StartupPath;
    if(dlg.ShowDialog() == DialogResult.OK)
    {
        try
        {
            CActualizacion actualizacion = new CActualizacion();
            actualizacion.GenerarReporte(dlg.FileName);
            MessageBox.Show("Reporte Generado. (" + dlg.FileName + ")");
            actualizacion.Dispose();
            Trace.WriteLine("Reporte generado: " + dlg.FileName);
        }
        catch(ActualizacionException aex)
        {
            MessageBox.Show(aex.Message);
        }
        finally
        {
            dlg.Dispose();
        }
    }
}

private void cmbLaboratorio_SelectedIndexChanged(object sender, EventArgs e)
{
    Trace.WriteLine("Se seleccionó el laboratorio " +
cmbLaboratorio.Text);
    txtArchivo.Text = "";
    btnActualizar.Enabled = false;
    btnReporte.Enabled = false;
}
}
}

```

De esa forma se completa todo el desarrollo de la aplicación solicitada, podemos ejecutarla y verificar su funcionamiento y resultados.

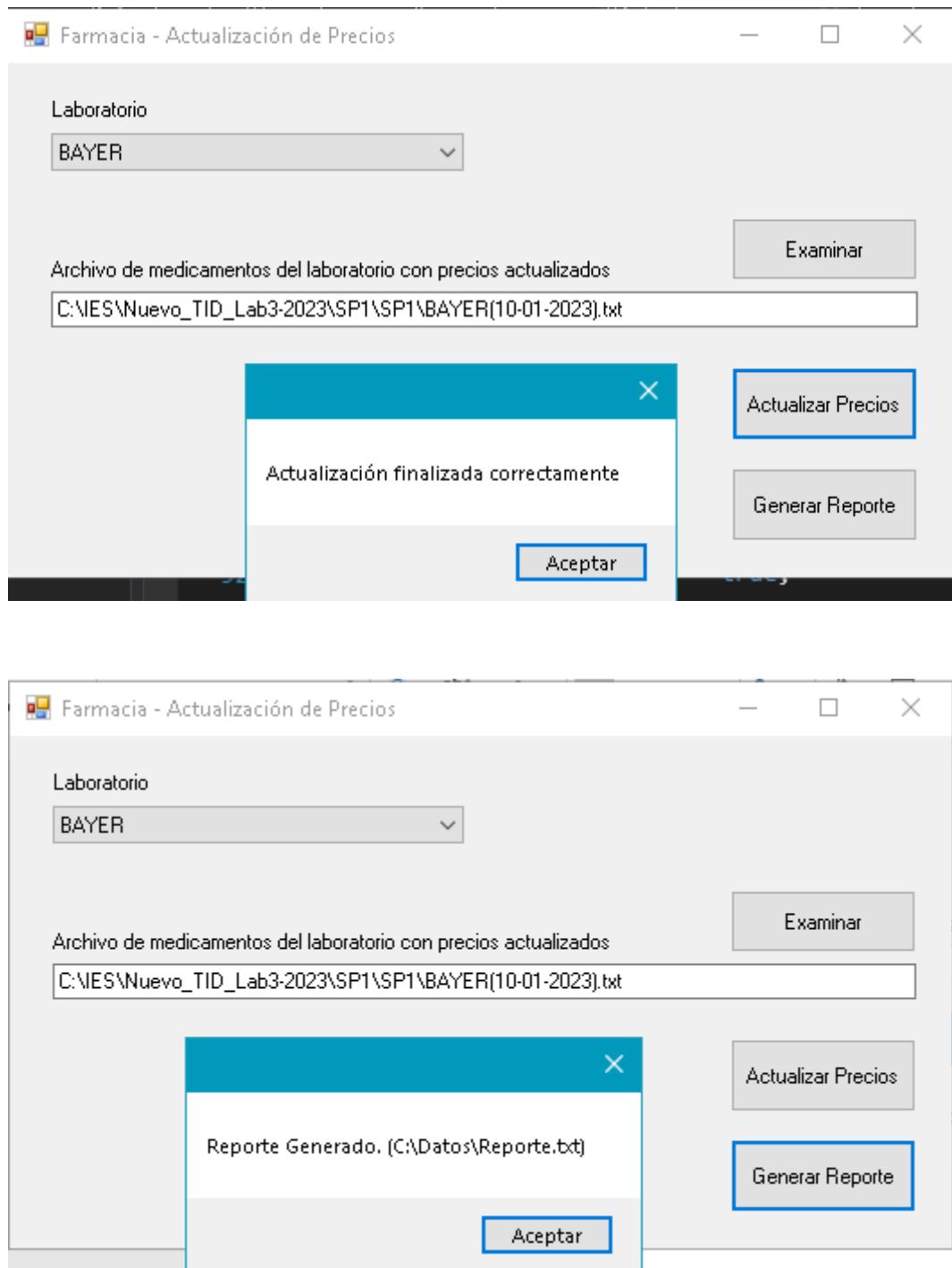
Elegimos este archivo de medicamentos para actualizar la base de datos:

```

BAYER(10-01-2023).txt
1010002, "GENIOL X 3 TABLETAS", 29.50
1010001, "ASPIRINA X 6 TABLETAS", 25.00
1010004, "ASPIRINETA X 3 TABLETAS", 27.25

```

```
1010010, "VITAMINA C 1G", 250.00
1010003, "CAFIASPIRINA X 6 TABLETAS", 21.50
1010009, "VITAMINA C 500mG", 175.00
```



El archivo del reporte generado (Reporte.txt) contiene estos datos:

Reporte de Medicamentos Actualizados				
Código	Nombre	Lab	Precio	Fecha
1010002	GENIOL X 3 TABLETAS	101	29,50	10/01/2023
1010001	ASPIRINA X 6 TABLETAS	101	25,00	10/01/2023
1010004	ASPIRINETA X 3 TABLETAS	101	27,25	10/01/2023
1010010	VITAMINA C 1G	101	250,00	10/01/2023
1010003	CAFIASPIRINA X 6 TABLETAS	101	21,50	10/01/2023
1010009	VITAMINA C 500mG	101	175,00	10/01/2023

Fecha del reporte: 06/03/2023

La consola de salida muestra esta información generada por Trace:

```
Salida ::::::::::::
Mostrar salida de: Depurar
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_32\System.Data
    Iniciando aplicación de Farmacia
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_32\System.Tr
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_32\System.Er
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_32\System.Er
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.
    Se seleccionó el laboratorio BAYER
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Accessi
        Datos de Laboratorios obtenidos 2
        Se seleccionó el archivo C:\IES\Nuevo_TID_Lab3-2023\SP1\SP1\BAYER(10-01-2023).txt
        Medicamento actualizado GENIOL X 3 TABLETAS
        Medicamento actualizado ASPIRINA X 6 TABLETAS
        Medicamento actualizado ASPIRINETA X 3 TABLETAS
        Medicamento actualizado VITAMINA C 1G
        Medicamento actualizado CAFIASPIRINA X 6 TABLETAS
        Medicamento actualizado VITAMINA C 500mG
        Reporte generado
        Reporte generado: C:\Datos\Reporte.txt
'SP1.exe' (CLR v4.0.30319: SP1.exe): 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\mscorli
```

Y el Visor de Eventos de Windows muestra estos eventos:

The screenshot shows the Windows Event Viewer interface. On the left, there's a navigation pane with sections like 'Visor de eventos (local)', 'Registros de Windows' (with 'Aplicación' selected), and 'Eventos reenviados'. The main area is titled 'Aplicación' with 'Número de eventos: 9.986'. It displays a table of events with columns: Nivel (Level), Fecha y hora (Date and Time), Origen (Source), Id. del evento (Event ID), and Categoría (Category). Most events are 'Información' level, originating from 'AppFarmaciaEventLog' at various times between 06/03/2023 18:25:42 and 18:25:26. A specific event is highlighted with a red box, showing 'Reporte generado: C:\Datos\Reporte.txt' in its details. The entire event table and its details window are enclosed in a large red box.

Nivel	Fecha y hora	Origen	Id. del evento	Categoría
Información	06/03/2023 18:26:42	MSSQL\$SQLSERVER	17890	Servidor
Información	06/03/2023 18:25:48	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:46	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:35	MSSQL\$SQLSERVER	17890	Servidor
Información	06/03/2023 18:25:26	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:26	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:26	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:26	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:26	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:21	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:14	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:14	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:25:14	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:24:10	AppFarmaciaEventLog	0	Ninguno
Información	06/03/2023 18:23:25	AppFarmaciaEventLog	0	Ninguno

Evento 0, AppFarmaciaEventLog

General Detalles

Reporte generado: C:\Datos\Reporte.txt

Nombre de registro: Aplicación
Origen: AppFarmaciaEventLog Registrado: 06/03/2023 18:25:48

Es la misma información que se observa en la consola de salida de Visual Studio, pero acá queda grabada de forma permanente cada vez que se ejecute la aplicación para que pueda ser consultada en el momento que sea necesario.

SP1/Ejercicio por resolver

Se solicita continuar el desarrollo de la aplicación realizada para la farmacia, el objetivo será mejorar sus prestaciones y agregar nuevas funcionalidades.

- Mejorar el proceso de actualización de cada medicamento controlando que el archivo de entrada tenga los valores correctos en todos sus campos: número de medicamento, nombre de medicamento y precio, cada línea deberá tener 3 valores y ser del tipo de dato correcto, en caso contrario se informará el error por medio de Trace, se debe descartar del proceso, esa línea del archivo, pero se continuará procesando al resto de los medicamentos hasta finalizar el archivo.
- Usar los métodos TraceInformation, TraceWarning y TraceError para diferenciar el tipo de mensaje que se genera con Trace.
- Agregar un segundo formulario para la generación de los reportes, el formulario debe permitir filtrar el contenido del reporte por laboratorio, es decir que el usuario debe poder elegir el laboratorio por medio de una lista desplegable y generar el reporte de los medicamentos actualizados de ese laboratorio solamente, se deberá incluir también la opción de generar el reporte con los medicamentos de todos los laboratorios. Agregar al archivo generado una línea que contenga el nombre del laboratorio seleccionado o la leyenda “Todos los laboratorios” según la selección realizada por el usuario.

SP1/Evaluación de paso

1. Indique la opción correcta:

La clave principal de una tabla debe estar formada por un único campo de la tabla.

- Verdadero
- Falso

2. Indique la opción correcta:

ADO .NET trabajando con entornos desconectados consume menos recursos que en entornos conectados.

- Verdadero
- Falso

3. Indique la opción correcta:

La clave principal de una tabla debe estar formada por un único campo de la tabla.

- Verdadero
- Falso

4. Indique la opción correcta:

Para crear un nuevo tipo de excepción se debe heredar de la clase base “Exception”.

- Verdadero
- Falso

5. Indique la opción correcta:

En la clase “OpenFileDialog” la propiedad que define la ubicación del archivo seleccionado en “FilePath”.

- Verdadero
- Falso

6. Indique la opción correcta:

Los registros creados con la clase Trace se pueden diferenciar de tipo usando los métodos “WriteInformation”, “WriteWarning” y “WriteError”.

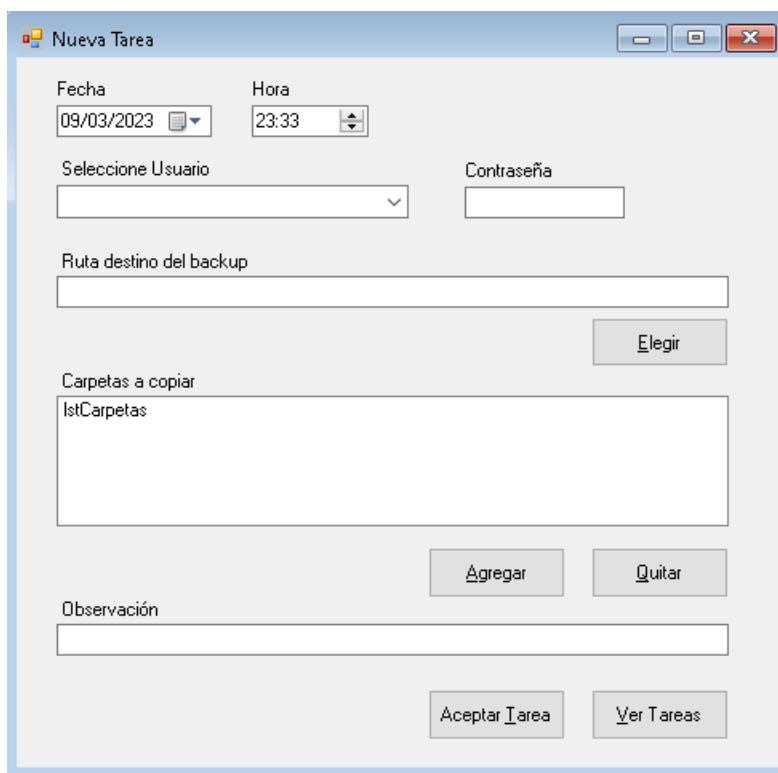
- Verdadero
- Falso

Respuestas correctas⁶

⁶1) Falso. 2) Verdadero. 3) Falso. 4) Verdadero. 5) Falso. 6) Falso.

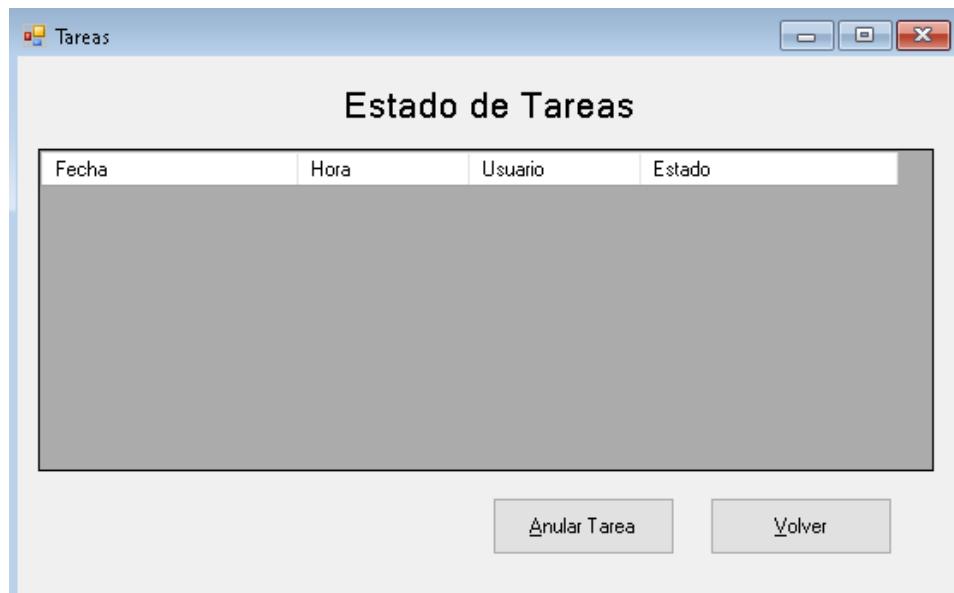
Situación profesional 2:Backup Programado

El administrador de una empresa del rubro automotriz le ha solicitado a usted, el desarrollo de una aplicación para que los empleados puedan almacenar tareas programadas con fecha y hora para el respaldo automático de la información generada en los distintos sectores de la empresa. El procedimiento para almacenar una tarea es la siguiente: seleccionar la fecha y hora de ejecución del respaldo, seleccionar el usuario responsable, ingresar la contraseña del mismo, la ruta para realizar la copia, las rutas de las carpetas a copiar y una observación sobre la copia.



Antes de almacenar una tarea se debe controlar que no exista una tarea con la fecha y hora seleccionada, la fecha y la hora no puede ser menor a la fecha y la hora actual, la contraseña del usuario debe ser correcta, la ruta destino tiene que ser correcta y se tiene que haber elegido al menos una carpeta para copia, no permita copiar rutas de carpetas repetidas y la carpeta destino nunca puede ser igual a una carpeta origen, en todos los casos informe el error correspondiente.

También el usuario tiene que poder ver todas las tareas almacenadas y poder anular cualquier tarea que se encuentre pendiente de ejecutar. Los estados de las tareas son: pendiente, terminada o anulada. De cada tarea se muestra la fecha, la hora, el nombre del usuario responsable y el estado de la tarea.



El proceso temporizado se tiene que ejecutar cada 30 segundos. Este proceso es el encargado de ejecutar todas las tareas pendientes al cumplirse la fecha y la hora programada. Para llevar a cabo las tareas mencionadas anteriormente, se cuenta con una base de datos que tiene tres tablas: ·

La tabla **usuarios** almacena un número para identificar al usuario, el nombre del usuario y su contraseña, la clave principal de la tabla es el número de usuario. ·

La tabla **tareas** almacena la fecha y hora de ejecución de la tarea, el número de usuario que generó la tarea, la ruta destino dónde se realizará la copia, una observación hecha por el usuario y el estado de la tarea (0=pendiente, 1=terminada o 2=anulada), la clave principal de la tabla está formada por las columnas fecha y hora. ·

La tabla **carpetas** almacena la fecha, la hora, el orden de la copia y la ruta de la carpeta a copiar, la clave principal de la tabla está formada por las columnas fecha, hora y orden.

Usuarios:

Nombre del campo	Tipo de datos
usuario	Número
nombre	Texto corto
palabra	Texto corto

Tareas:

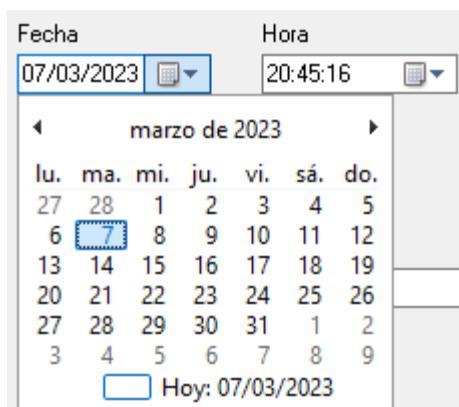
Nombre del campo	Tipo de datos
fecha	Fecha/Hora
hora	Texto corto
usuario	Número
rutadestino	Texto corto
observacion	Texto corto
estado	Número

Carpetas:

Nombre del campo	Tipo de datos
fecha	Fecha/Hora
hora	Texto corto
orden	Número
rutaorigen	Texto corto

SP2/H1: Control DateTimePicker

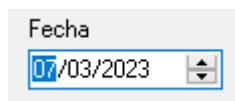
El control DateTimePicker se usa para permitir al usuario seleccionar una fecha y hora, y mostrar esa fecha y hora en el formato específico. El control DateTimePicker facilita el trabajo con fechas y horas, ya que controla las validaciones de datos automáticamente.



A la izquierda de la imagen: DateTimePicker configurado con formato de fecha corta, al activarlo despliega un calendario para seleccionar una fecha de manera más cómoda, la ventana del calendario permite cambiar también el mes y el año.

A la derecha se presenta otro control DateTimePicker, pero en este caso está configurado para mostrar la hora en formato "hh:mm:ss".

Otra posibilidad de uso frecuente es reemplazar la ventana que muestra el calendario por dos botones "up-down", para subir o bajar los valores del día, mes o año según qué parte se seleccione. La propiedad "ShowUpDown" en verdadero activa este modo y desactiva el calendario:



Este formato puede resultar más cómodo cuando tenemos disponible un rango de fechas acotado. Las propiedades “MinDate” y “MaxDate” permiten establecer los valores mínimos y máximos entre los que se podrán elegir las fechas.

La propiedad “Value” devuelve el valor seleccionado en un tipo DateTime, es decir que contiene siempre la fecha y la hora. El valor de la hora será la hora del sistema salvo que el control DateTimePicker esté configurado para mostrar la hora en cuyo caso el valor de la hora será el configurado por el usuario en el control.

Los valores del control pictureBox se pueden mostrar en cuatro formatos, que se establecen mediante la propiedad “Format”: “Long” (fecha larga), “Short” (fecha corta), “Time” (hora) o “Custom” (personalizado). El valor predeterminado de la propiedad “Format” es “Long”.

Si desea que aparezca el control DateTimePicker para seleccionar o editar horas en lugar de fechas, establezca la propiedad “ShowUpDown” en true y la propiedad “Format” en Time.

Si la propiedad “Format” está establecida “DateTimePickerFormat.Custom”, puede crear su propio estilo de formato estableciendo la propiedad “CustomFormat” y creando una cadena de formato personalizada.

La cadena de formato personalizado puede ser una combinación de caracteres de campo personalizados y otros caracteres literales. Por ejemplo, puede mostrar la fecha como "Junio 1, 2012 Viernes" estableciendo la propiedad “CustomFormat” en "MMMM dd, yyyy - dddd".

```
DateTimePicker.Format = DateTimePickerFormat.Custom;  
DateTimePicker.CustomFormat = "MMMM dd, yyyy dddd";
```

En el caso de necesitar establecer el valor de la fecha y el valor de la hora en el mismo control DateTimePicker podemos asignar este formato:

```
DateTimePicker.Format = DateTimePickerFormat.Custom;  
DateTimePicker.CustomFormat = "dd/MM/yyyy hh:mm:ss";
```

Puede revisar la lista completa de propiedades y métodos del control “DateTimePicker” en este enlace: [DateTimePicker Clase \(System.Windows.Forms\) | Microsoft Learn](#)

SP2/Autoevaluación 1

1. Indique la opción correcta:

El control DateTimePicker permite mostrar los valores de fecha y hora solamente con formatos fijos y preestablecidos.

- Verdadero
- Falso

2. Indique la opción correcta:

El nombre de la propiedad que permite obtener el valor seleccionado en el control DateTimePicker es:

- Text
- Value
- Date
- Ninguno de los anteriores

3. Indique la opción correcta:

En un control DateTimePicker es posible fijar las fechas mínimas y máximas que admite como válidas.

- Verdadero
- Falso

4. Indique la opción correcta:

Para que el control DateTimePicker muestre la ventana con el calendario desde el botón incluido a su derecha se debe ajustar el valor de la propiedad:

- ShowUpDown
- ShowCheckBox
- ShowCalendar

5. Indique la opción correcta:

El control DateTimePicker valida automáticamente si una fecha ingresada manualmente es válida o no.

- Verdadero
- Falso

Respuestas correctas⁷

⁷1) Falso. 2) Value. 3) Verdadero. 4) ShowUpDown 5) Verdadero.

SP2/H2: Control Timer

En esta Situación profesional se plantea la necesidad de que la aplicación lleve un control del tiempo en el que van a transcurrir ciertas tareas. Para ello utilizaremos un control muy útil que es el Temporizador (Timer).

El control Timer responde al paso del tiempo. Es independiente del usuario de la aplicación, y se puede programar para que ejecute acciones a intervalos periódicos de tiempo. Un uso típico es comprobar la hora del sistema para ver si es el momento de ejecutar alguna tarea, como es lo que deseamos realizar para la resolución de nuestra situación profesional. Los cronómetros también son útiles para otro tipo de procesamiento en segundo plano.

Los controles Timer deben estar asociados a un formulario, por lo tanto, para crear una aplicación de cronómetro, debe crear al menos un formulario, aunque no es obligatorio que sea visible, si no se necesita para otro fin.

Propiedades Principales

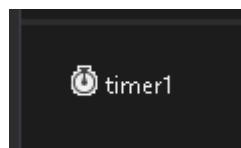
Interval: Número de milisegundos que deben transcurrir para ejecutar el evento Tick. El evento Tick es periódico. La propiedad Interval no determina la duración sino la frecuencia. La duración del intervalo depende de la precisión que se desea. Como existe cierta posibilidad de error, se recomienda que la precisión del intervalo sea el doble de la deseada. Se recomienda no fijar intervalos demasiado cortos, a menos que sea realmente necesario.

Enabled (verdadero o falso). Determina si el control Timer realiza el conteo de los milisegundos establecidos en la propiedad Interval. Cuando su valor es True, el cronómetro empieza a funcionar, cuando su valor es False el cronómetro se detiene. Esta propiedad se puede establecer en tiempo de diseño o en tiempo de ejecución.

Eventos del control Timer:

El principal evento se denomina **Tick**: este evento se genera en forma automática cada vez que el cronómetro alcanza el valor establecido en la propiedad **Interval**, y se repetirá en forma periódica mientras la propiedad **Enabled** del timer sea **True**. El control Timer es un control invisible en la interfaz de la aplicación y, por lo tanto, no se mostrará al usuario, solamente se visualiza en tiempo de diseño.

Ejemplo demostración: En el formulario agregamos un control Timer:



En el evento Load del formulario establecemos el valor de la propiedad “Interval” en 1000, serían 1000 milisegundos = 1 segundo, y habilitamos la ejecución del timer colocando “true” en la propiedad “Enabled”.

```
private void Form1_Load(object sender, EventArgs e)
{
    timer1.Interval = 1000;
    timer1.Enabled = true;
}

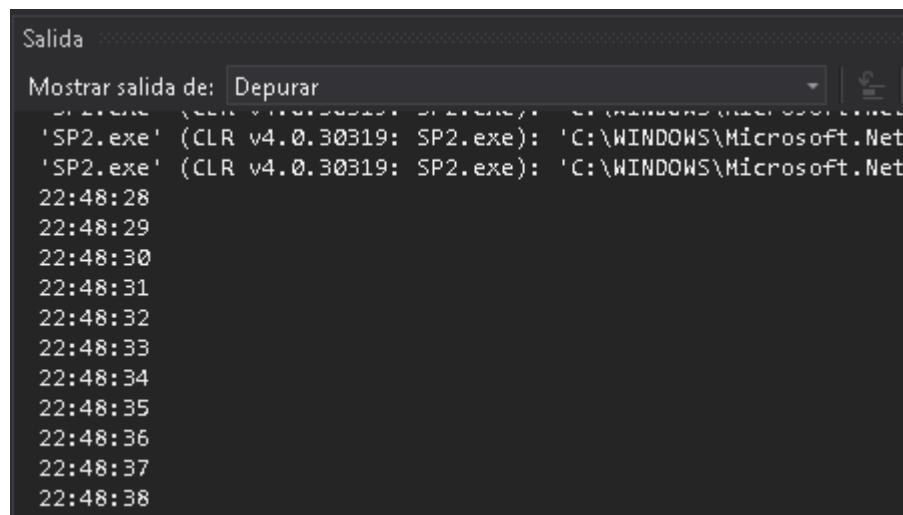
private void timer1_Tick(object sender, EventArgs e)
{
    // el evento se ejecuta una vez por segundo
    Trace.WriteLine(DateTime.Now.ToString());
}
```

Haciendo doble clic sobre el ícono del timer en el formulario generamos el evento “Tick”, y en él escribimos una llamada al método **WriteLine** del objeto **Trace**, el valor que se escribirá en la consola de salida de Visual Studio será el valor de la hora del sistema con formato “hh:mm:ss”, eso ocurrirá una vez cada un segundo porque así está determinado por la propiedad “Interval”.

Tenga en cuenta que para poder usar la clase **Trace** se debe agregar al código del formulario:

```
using System.Diagnostics;
```

Al ejecutar el formulario podemos ver la consola de salida con los valores de la hora cada un segundo, esto es porque el evento “Tick” del timer se ejecuta automáticamente cada 1 segundo.



The screenshot shows the 'Salida' (Output) window in Visual Studio. The 'Mostrar salida de:' dropdown is set to 'Depurar' (Debug). The output pane displays the following text:

```
'SP2.exe' (CLR v4.0.30319: SP2.exe): 'C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\Temporary ASP.NET Files\root\1e3a2f3d\133e\assembly\dl3\1e3a2f3d\133e1_0.cs' (C:\Windows\Microsoft.NET\Framework\v4.0.30319\Temporary ASP.NET Files\root\1e3a2f3d\133e\133e1_0.cs): Line 12:     'SP2.exe' (CLR v4.0.30319: SP2.exe): 'C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\Temporary ASP.NET Files\root\1e3a2f3d\133e\assembly\dl3\1e3a2f3d\133e1_0.cs' (C:\Windows\Microsoft.NET\Framework\v4.0.30319\Temporary ASP.NET Files\root\1e3a2f3d\133e\133e1_0.cs): Line 12:     22:48:28  
22:48:29  
22:48:30  
22:48:31  
22:48:32  
22:48:33  
22:48:34  
22:48:35  
22:48:36  
22:48:37  
22:48:38
```

En un caso real el evento “Tick” es el que lleva el control del proceso que se desea ejecutar en forma periódica de acuerdo al valor de la propiedad “Interval”. En ese evento se deberá colocar toda la lógica del proceso que necesitamos ejecutar periódicamente.

En nuestra situación profesional deberemos agregar el control TIMER para poder realizar las tareas necesarias y controlar la ejecución de las copias de las carpetas de backup en la fecha y hora registradas en la base de datos.

Podrá consultar toda la documentación del control Timer en este enlace:
<https://learn.microsoft.com/en-us/dotnet/api/system.timers.timer?view=netframework-4.8.1>

SP2/Autoevaluación 2

1. Relaciones conceptos y características. Relacione los conceptos de la primera columna con sus características correspondientes en la columna de la derecha:

Conceptos	Características
a. Interval.	1. Tiene lugar cuando ha transcurrido el intervalo de tiempo especificado.
b. Enable	2. Habilita la generación de eventos con el intervalo establecido.
c. Tick	3. Frecuencia de los eventos en milisegundos.

2. Indique la opción correcta

¿Cuál es el evento que nos permite agregar código para manejar la acción de un Timer?:

- Evento Tick
- Enabled
- Changed
- Load

3. Indique la opción correcta

¿Cuál es la propiedad que nos permite habilitar la cuenta del tiempo transcurrido en un control Timer?:

- Interval
- Time

- Enabled
- Tick

4. Indique la opción correcta

Cuando se cumple el intervalo de un control Timer, se genera un evento Tick.

- Verdadero
- Falso

5. Indique la opción correcta

El control Timer puede mostrarse en la interfaz o no según el valor de la propiedad Visible.

- Verdadero
- Falso

Respuestas correctas⁸

⁸1) a→3, b→2, c→1. 2) Evento Tick. 3) Enabled. 4) Verdadero. 5) Falso.

SP2/H3: Transacciones en ADO .NET

Las transacciones están relacionadas a las operaciones sobre la base de datos y nos permiten tratar varias operaciones como una sola unidad, como si fueran una sola operación atómica, este tratamiento tiene por objetivo mantener la integridad de los datos entre las distintas tablas que participan de esas operaciones individuales. Si al ejecutar algunos de los pasos que componen la transacción falla entonces la transacción se cancela y todos los cambios realizados hasta ese momento se revierten dejando todo en el estado inicial, si todos los pasos se completan exitosamente entonces la transacción finaliza confirmando todos los cambios. Este mecanismo tiene entonces tres estados: “Begin” para dar inicio a la transacción, “Commit” para confirmar todos los cambios realizados y “Rollback” para cancelar y revertir todos los cambios realizados.

Una transacción se caracteriza por cumplir las llamadas propiedades **ACID**, que identifican los requisitos para que una transacción se realice. Estas propiedades son: atomicidad (**Atomicity**), coherencia (**Consistency**), aislamiento (**Isolation**) y permanencia (**Durability**).

Desde ADO .NET podemos trabajar con transacciones a partir del objeto “Connection”, este objeto posee los métodos para controlar el inicio, confirmación o cancelación de las transacciones:

- BeginTransaction
- Commit
- Rollback

Con el objeto “Transaction” creado al ejecutar “BeginTransaction” se controlará todo el proceso incluido en la transacción, todos los comandos ejecutados deben llevar asociado el objeto transacción creado al comienzo para poder determinar si algo falla o se completa exitosamente.

Ejemplo: supongamos que debemos hacer un proceso que trabaja con 2 tablas: “Movimientos” y “Cuentas”. La tabla “Movimientos” registra las operaciones de depósito que se realizan en cada cuenta, la tabla “Cuentas” registra el saldo actual de cada cuenta. Cuando se quiere procesar una operación de depósito se debe agregar un registro nuevo a la tabla “Movimientos” y luego editar la tabla de “Cuentas” para ajustar el saldo de acuerdo al importe del depósito, son dos operaciones independientes, cada una trabaja con una tabla diferente. Si no usamos transacciones para agrupar ambas operaciones podría ocurrir que luego de procesar correctamente el movimiento ocurra un error en el proceso de la cuenta, lo que nos dejaría cargado un movimiento cuyo importe no está reflejado en el saldo de la cuenta, con una transacción eso no puede ocurrir porque ante cualquier error todos los cambios se revierten y se vuelve al estado inicial.

Código principal que hace uso de una transacción, para simplificar el ejemplo asumimos que la conexión con la base de datos ya está creada previamente, es el objeto “CNN”, la conexión debe estar abierta.

```
public void AddMovimiento()
{
    OleDbTransaction Transaccion = null;
    try
    {
        // iniciar la transaccion (CNN es el objeto OleDbConnection)
        Transaccion = CNN.BeginTransaction();
        // agregar un registro nuevo en Movimientos
        InsertMovimiento(Transaccion);
        // actualizar el registro de la cuenta con el nuevo saldo
        UpdateCuenta(Transaccion);
        // confirmar todos los cambios realizados
        Transaccion.Commit();
    }
    catch (Exception ex)
    {
        // revertir los cambios de la transacción
        Transaccion.Rollback();
    }
}
```

Como se observa en el código anterior, el método “BeginTransaction” crea el objeto y da inicio al proceso de transacción, luego se realizan las operaciones que se necesiten completar como si fuera una única operación, en este ejemplo hay dos operaciones, la primera inserta un registro nuevo en la tabla de Movimientos y la segunda actualiza el

saldo en la tabla de Cuentas, si ambas operaciones se completan sin error se ejecuta el método “Commit” que se encarga de confirmar sobre la base de datos todos los cambios realizados en los pasos anteriores. Si se produce algún error el bloque catch() captura la excepción y se ejecuta el método “Rollback” de la transacción, revirtiendo todos los cambios previos y dejando la base de datos en el estado inicial que tenía antes de iniciar la transacción.

Un detalle importante es que cada una de las operaciones que forman parte de la transacción deben asignar el objeto Transacción creado al objeto “Command” como se puede ver en las dos funciones siguientes:

```
public void InsertMovimiento(OleDbTransaction transaccion)
{
    // asignar la transacción al comando de Movimientos
    CmdMov.Transaction = transaccion;
    // agregar un nuevo registro a la tabla de Movimientos
    DataRow dr = DS.Tables["Movimientos"].NewRow();
    dr["Fecha"] = fecha;
    dr["Importe"] = importe;
    dr["Cuenta"] = cuenta;
    DS.Tables["Movimientos"].Rows.Add(dr);
    OleDbCommand cb = new OleDbCommandBuilder(DAMov);
    DAMov.Update(DS, "Movimientos");
}
```

Este método se encarga de agregar un nuevo registro en la tabla de Movimientos, el proceso ya se conoce y no necesita mayores comentarios, la única salvedad es esta línea:

```
// asignar la transacción al comando de Movimientos
CmdMov.Transaction = transaccion;
```

El objeto “Command” tiene la propiedad “Transaction” y debe ser asignada con el objeto “Transaction” creado inicialmente con el método “BeginTransaction” para que la transacción pueda conocer todas las operaciones que deben ser controladas y finalmente confirmadas o canceladas.

Algo similar resulta en el método para actualizar el saldo de la cuenta:

```
public void UpdateCuenta(OleDbTransaction transaccion)
{
    // asignar la transacción al comando de Cuentas
    CmdCta.Transaction = transaccion;
    // actualizar el registro de la cuenta con el nuevo saldo
    DataRow dr = DS.Tables["Cuentas"].Rows.Find(cuenta);
    dr.BeginEdit();
    dr["Saldo"] = dr["Saldo"] + importe;
    dr.EndEdit();
    OleDbCommand cb = new OleDbCommandBuilder(DACta);
    DACta.Update(DS, "Cuentas");
}
```

En este ejemplo se asume que todos los objetos DataSet, Command y DataAdapter, como así también los valores de los campos que se usan para asignar en cada tabla son todos accesibles desde cada uno de los métodos programados, por ejemplo, declarados como miembros privados de la clase. En casos reales, se recomienda que esos objetos y variables sean parámetros de cada uno de los métodos.

Puede consultar este tema en el enlace oficial de Microsoft: [Transacciones locales - ADO.NET | Microsoft Learn](#)

SP2/Autoevaluación 3

1. Indique la opción correcta

Una transacción permite procesar varias operaciones como si fueran una sola unidad, si alguna operación falla se descartan todos los cambios realizados.

- Verdadero
- Falso

2. Indique la opción correcta

- El objetivo de las transacciones es:
- Mantener la integridad de los datos
- Ejecutar más rápido las operaciones
- Delegar los controles a la base de datos

3. Indique la opción correcta

Indique cuál es la secuencia correcta en la ejecución de las etapas de una transacción

- Begin, Rollback, Commit
- Commit, Rollback, Begin
- Begin, Commit, Rollback
- Rollback, Begin, Commit

4. Indique la opción correcta

En ADO .NET el manejo de las transacciones se genera a través de la conexión con la base de datos.

- Verdadero
- Falso

5. Indique la opción correcta

Una vez confirmada la transacción ya no es posible revertir los cambios en forma automática.

- Verdadero
- Falso

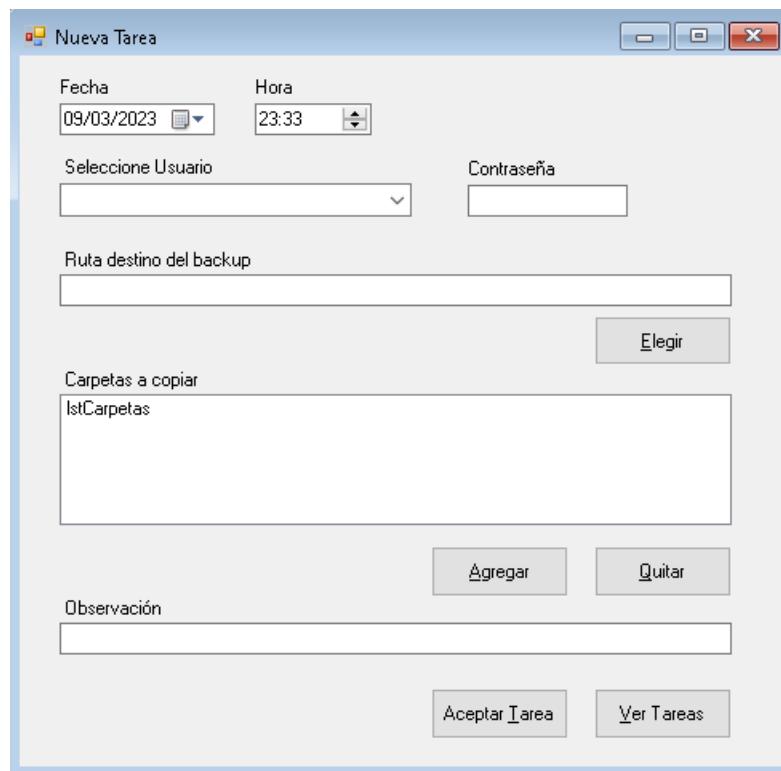
Respuestas correctas⁹

⁹1) Verdadero. 2) Mantener la integridad de los datos. 3) Begin, Commit, Rollback. 4) Verdadero. 5) Verdadero.

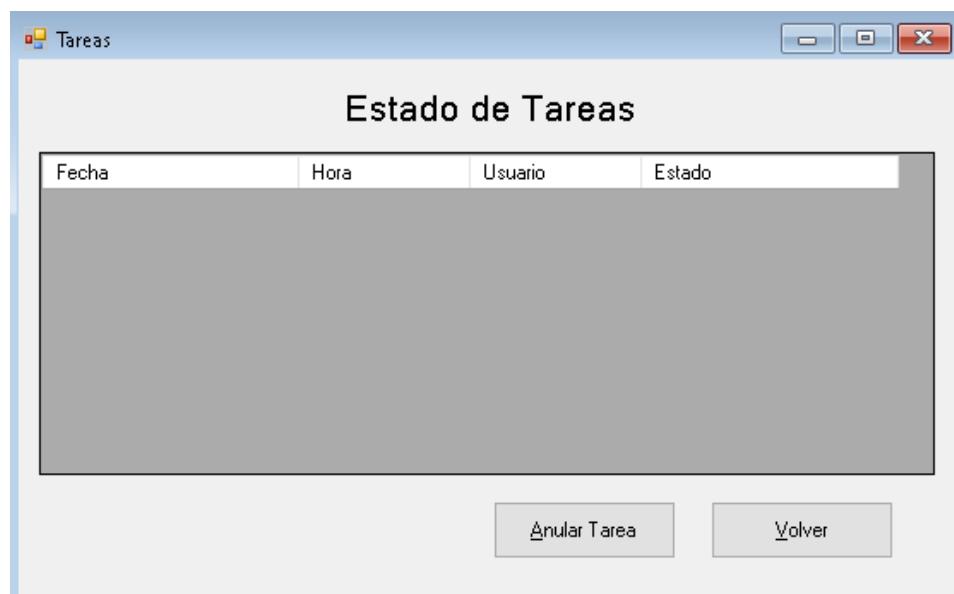
SP2/Ejercicio resuelto

Comenzaremos la resolución de esta situación profesional creando el proyecto y diseñando la interfaz gráfica de cada formulario:

Form1:

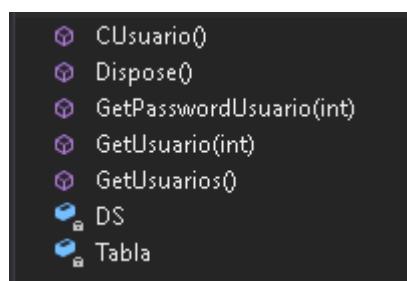


Form2:



Seguidamente agregaremos las clases necesarias para manejar las tablas de la base de datos y los procedimientos que debemos realizar en ellas. Tendremos solamente 2 clases: “CUsuario” y “CTarea”, describiremos el contenido y la implementación de cada clase.

Clase “**CUsuario**”, se encargará de acceder a la tabla de Usuarios por medio del constructor de la clase, y tendrá los métodos “Dispose” para liberar los recursos empleados, “GetPasswordUsuario” para recuperar el valor del password de un usuario determinado, “GetUsuario” para obtener el nombre de un usuario y “GetUsuarios” para obtener la tabla completa con todos los usuarios.



Clase “**CTarea**”, la idea es similar a la clase anterior, pero en este caso se controlarán 2 tablas: la tabla de Tareas y la tabla de Carpetas, esto es porque tenemos una relación y dependencia entre las tablas, una tarea deberá tener una o varias carpetas asociadas para hacer las copias de archivos. Los métodos serán, además del constructor y del método Dispose, los métodos “AddTarea” encargada de agregar una nueva tarea a la base de datos, internamente ejecutará otros dos métodos, “InsertTarea” e “InsertRutas” para distribuir los datos en las 2 tablas de la base. Tenemos también el método “GetTareas” que devuelve la tabla de tareas completa, el método “EjecutarTarea” que será el método invocado desde el evento “Tick” del control timer y finalmente el método “AnularTarea” para cambiar su estado y evitar que sea ejecutada.

```

    ↗ AddTarea(System.DateTime, string, int, string, string, System.Collections.Generic.IList<SP2.Tarea>)
    ↗ AnularTarea(System.DateTime, string)
    ↗ CopyAll(System.IO.DirectoryInfo, System.IO.DirectoryInfo)
    ↗ CTarea()
    ↗ EjecutarTarea(System.Windows.Forms.Timer)
    ↗ GetTareas()
    ↗ InsertRutas(System.Data.OleDb.OleDbTransaction, System.DateTime, string, string)
    ↗ InsertTarea(System.Data.OleDb.OleDbTransaction, System.DateTime, string, string)
        ↘ CmdCarpetas
        ↘ CmdTareas
        ↘ CNN
        ↘ DACarpetas
        ↘ DATareas
        ↘ DS
        ↘ TablaCarpetas
        ↘ TablaTareas

```

Observe que han declarado como objetos privados a la clase además de la conexión (CNN) y del DataSet (DS), los objetos Command y los DataAdapter de cada tabla, esto es necesario porque se usarán transacciones y estos objetos deben estar disponibles en varios métodos de la clase.

Veamos ahora la implementación completa de la clase “**CUsuario**”:

```

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

namespace SP2
{
    public class CUsuario
    {
        DataSet DS;
        String Tabla = "Usuarios";
        // constructor
        public CUsuario()
        {
            try
            {
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=BackupProgramado.mdb";
                cnn.Open();
                DS = new DataSet();
                OleDbCommand cmd = new OleDbCommand();
                cmd.Connection = cnn;

```

```

        cmd.CommandType = CommandType.TableDirect;
        cmd.CommandText = Tabla;
        OleDbDataAdapter DA = new OleDbDataAdapter(cmd);
        DA.Fill(DS, Tabla);
        DataColumn[] pk = new DataColumn[1];
        pk[0] = DS.Tables[Tabla].Columns["usuario"];
        DS.Tables[Tabla].PrimaryKey = pk;
        OleDbCommandBuilder cb = new OleDbCommandBuilder(DA);
        cnn.Close();
    }
    catch (Exception ex)
    {
        String MsgErr = "CUsuario: " + ex.Message;
        throw new Exception(MsgErr);
    }
}

public DataTable GetUsuarios()
{
    if (DS.Tables.Count == 1)
    {
        return DS.Tables[Tabla];
    }
    else
    {
        throw new Exception("La tabla no existe");
    }
}

public String GetUsuario(int usuario)
{
    String nombre = "";
    DataRow drU = DS.Tables[Tabla].Rows.Find(usuario);
    if(drU != null)
    {
        nombre = drU["Nombre"].ToString();
    }
    return nombre;
}

public String GetPasswordUsuario(int usuario)
{
    DataRow drU = DS.Tables[Tabla].Rows.Find(usuario);
    if(drU != null)
    {
        return drU["palabra"].ToString();
    }
    else
    {
        throw new Exception("El usuario no existe");
    }
}

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

```

El método **constructor** realiza la conexión con la base de datos y carga el **DataSet** con la tabla de Usuarios, define también la clave primaria de la tabla.

El método “**GetUsuario**” recibe el número de usuario por parámetro y devuelve el nombre del usuario, para buscarlo se usa el método “Find” ya que la tabla tiene definida la clave primaria por el mismo campo del número de usuario.

El método “**GetPasswordUsuario**” funciona igual al anterior, pero devuelve el valor del campo “palabra” (password) del usuario.

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

Implementación de la clase “**CTarea**”

```
public class CTarea
{
    OleDbConnection CNN;
    DataSet DS;
    OleDbDataAdapter DATareas;
    OleDbDataAdapter DACarpetas;
    OleDbCommand CmdTareas;
    OleDbCommand CmdCarpetas;
    String TablaTareas = "Tareas";
    String TablaCarpetas = "Carpetas";

    public CTarea()
    {
        try
        {
            CNN = new OleDbConnection();
            CNN.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=BackupProgramado.mdb";
            CNN.Open();
            DS = new DataSet();
            CmdTareas = new OleDbCommand();
            CmdTareas.Connection = CNN;
            CmdTareas.CommandType = CommandType.TableDirect;
            CmdTareas.CommandText = TablaTareas;
            DATareas = new OleDbDataAdapter(CmdTareas);
            DATareas.Fill(DS, TablaTareas);
            DataColumn[] pkT = new DataColumn[2];
            pkT[0] = DS.Tables[TablaTareas].Columns["Fecha"];
            pkT[1] = DS.Tables[TablaTareas].Columns["Hora"];
            DS.Tables[TablaTareas].PrimaryKey = pkT;
            OleDbCommandBuilder cbT = new OleDbCommandBuilder(DATareas);
            //
            CmdCarpetas = new OleDbCommand();
            CmdCarpetas.Connection = CNN;
            CmdCarpetas.CommandType = CommandType.TableDirect;
```

```

        CmdCarpetas.CommandText = TablaCarpetas;
        DACarpetas = new OleDbDataAdapter(CmdCarpetas);
        DACarpetas.Fill(DS, TablaCarpetas);
        DataColumn[] pkC = new DataColumn[3];
        pkC[0] = DS.Tables[TablaCarpetas].Columns["Fecha"];
        pkC[1] = DS.Tables[TablaCarpetas].Columns["Hora"];
        pkC[2] = DS.Tables[TablaCarpetas].Columns["Orden"];
        DS.Tables[TablaCarpetas].PrimaryKey = pkC;
        OleDbCommandBuilder cbC = new OleDbCommandBuilder(DACarpetas);
        CNN.Close();
    }
    catch (Exception ex)
    {
        String MsgErr = "CTarea: " + ex.Message;
        throw new Exception(MsgErr);
    }
}
public DataTable GetTareas()
{
    if (DS.Tables.Contains(TablaTareas))
    {
        // devuelve la tabla de Tareas completa
        return DS.Tables[TablaTareas];
    }
    else
    {
        throw new Exception("La tabla no existe");
    }
}

public void AddTarea(DateTime Fecha,
                      String Hora,
                      int usuario,
                      String RutaDestino,
                      String Observacion,
                      List<String> Rutas)
{
    OleDbTransaction Transaccion = null;
    try
    {
        // abrir la conexión
        CNN.Open();
        // iniciar la transaccion
        Transaccion = CNN.BeginTransaction();
        // agregar la tarea
        InsertTarea(Transaccion, Fecha, Hora, usuario, RutaDestino,
Observacion);
        // agregar las rutas
        InsertRutas(Transaccion, Fecha, Hora, Rutas);
        // confirmar todos los cambios realizados
        Transaccion.Commit();
        CNN.Close();
    }
    catch(Exception ex)
    {
        // deshacer los cambios
        Transaccion.Rollback();
        throw ex;
    }
}

```

```

}

public void InsertTarea(OleDbTransaction transaccion, DateTime Fecha,
                        String Hora,
                        int usuario,
                        String RutaDestino,
                        String Observacion)
{
    try
    {
        CmdTareas.Transaction = transaccion;
        DataRow dr = DS.Tables[TablaTareas].NewRow();
        dr["Fecha"] = Fecha;
        dr["Hora"] = Hora;
        dr["Usuario"] = usuario;
        dr["RutaDestino"] = RutaDestino;
        dr["Observacion"] = Observacion;
        dr["Estado"] = 0; // estado pendiente
        DS.Tables[TablaTareas].Rows.Add(dr);
        DATareas.Update(DS, TablaTareas);
    }
    catch(Exception ex)
    {
        throw new Exception(ex.Message);
    }
}

public void InsertRutas(OleDbTransaction transaccion, DateTime Fecha,
                        String Hora, List<String> Rutas)
{
    int orden = 1;
    try
    {
        CmdCarpetas.Transaction = transaccion;
        foreach (String ruta in Rutas)
        {
            DataRow dr = DS.Tables[TablaCarpetas].NewRow();
            dr["Fecha"] = Fecha;
            dr["Hora"] = Hora;
            dr["Orden"] = orden;
            dr["RutaOrigen"] = ruta;
            DS.Tables[TablaCarpetas].Rows.Add(dr);
            orden++; // incrementa el número de orden
        }
        DACarpetas.Update(DS, TablaCarpetas);
    }
    catch(Exception ex)
    {
        throw new Exception(ex.Message);
    }
}

public void AnularTarea(DateTime Fecha,
                        String Hora)
{
    try
    {
        // formar el arreglo con los valores de la clave a buscar en

```

Tareas

```

        object[] clave = new object[2];
        clave[0] = Fecha; // primer campo de la clave
        clave[1] = Hora; // segundo campo de la clave
        DataRow dr = DS.Tables[TablaTareas].Rows.Find(clave);
        if (dr != null)
        {
            // si encuentra la tarea se controla el estado
            if ((int)dr["Estado"] != 0) // solo se anulan tareas
pendientes
            {
                throw new Exception("La tarea no está pendiente, no se
puede anular.");
            }
            dr.BeginEdit();
            dr["Estado"] = 2; // estado anulada
            dr.EndEdit();
            DATareas.Update(DS, TablaTareas);
        }
        else
        {
            // si no encuentra la tarea se dispara una excepción
            throw new Exception("La tarea no existe");
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

public void EjecutarTarea(Timer timer)
{
    timer.Enabled = false;
    try
    {
        DateTime Fecha = DateTime.Now.Date;
        String Hora = DateTime.Now.ToShortTimeString().Substring(0, 5);
        Object[] clave = new Object[2];
        clave[0] = Fecha;
        clave[1] = Hora;
        DataRow dr = DS.Tables[TablaTareas].Rows.Find(clave);
        // controlar que exista la tarea y que su estado sea Pendiente: 0
        if (dr != null && (int)dr["Estado"] == 0)
        {
            int orden = 1;
            bool existeRuta = true;
            // buscar todas las carpetas a copiar de la tarea
            while (existeRuta)
            {
                Object[] claveRuta = new Object[3];
                claveRuta[0] = Fecha;
                claveRuta[1] = Hora;
                claveRuta[2] = orden;
                DataRow drR =
DS.Tables[TablaCarpetas].Rows.Find(claveRuta);
                if (drR != null)
                {
                    // realizar las copias de los archivos

```

```

        DirectoryInfo diSource = new
DirectoryInfo(drR["RutaOrigen"].ToString());
        DirectoryInfo diTarget = new
DirectoryInfo(dr["RutaDestino"].ToString());

        CopyAll(diSource, diTarget); // copia el contenido de
la carpeta
        orden++;
    }
    else
    {
        existeRuta = false; // no hay más rutas para copiar
    }
}
// actualizar el estado de la tarea
dr.BeginEdit();
dr["Estado"] = 1; // estado finalizada
dr.EndEdit();
DATareas.Update(DS, TablaTareas);
}
}
catch (Exception ex)
{
    throw ex;
}
timer.Enabled = true;
}

public void CopyAll(DirectoryInfo source, DirectoryInfo target)
{
    String Ruta = Path.Combine(target.FullName, source.Name);
    Directory.CreateDirectory(Ruta); // si existe crea el directorio
destino

    // copiar los archivos del directorio origen al directorio destino.
    foreach (FileInfo fi in source.GetFiles())
    {
        fi.CopyTo(Path.Combine(Ruta, fi.Name), true);
    }
}

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

}
}

```

El método **constructor** realiza la conexión con la base de datos y carga el dataSet con las tablas de Tareas y Carpetas, define también las claves primarias de cada tabla.

El método “**GetTareas**” devuelve la tabla completa de Tareas.

El método “**AddTarea**” recibe por parámetro todos los valores necesarios para grabar una nueva tarea: fecha, hora, usuario, carpeta destino para las copias, observación y una lista de Strings con las rutas de las carpetas origen para la copia de los archivos. Este método hace uso de una transacción y las operaciones involucradas son las que se ejecutan con los métodos “**InsertTarea**” e “**InsertRutas**”, si ambas se ejecutan correctamente se realiza el Commit y en caso de algún error se realiza el Rollback.

Vemos en detalle la implementación de estos dos últimos métodos.

```
public void InsertTarea(OleDbTransaction transaccion, DateTime Fecha,
                        String Hora,
                        int usuario,
                        String RutaDestino,
                        String Observacion)
{
    try
    {
        CmdTareas.Transaction = transaccion;
        DataRow dr = DS.Tables[TablaTareas].NewRow();
        dr["Fecha"] = Fecha;
        dr["Hora"] = Hora;
        dr["Usuario"] = usuario;
        dr["RutaDestino"] = RutaDestino;
        dr["Observacion"] = Observacion;
        dr["Estado"] = 0; // estado pendiente
        DS.Tables[TablaTareas].Rows.Add(dr);
        DATareas.Update(DS, TablaTareas);
    }
    catch(Exception ex)
    {
        throw new Exception(ex.Message);
    }
}

public void InsertRutas(OleDbTransaction transaccion, DateTime Fecha,
                        String Hora, List<String> Rutas)
{
    int orden = 1;
    try
    {
        CmdCarpetas.Transaction = transaccion;
        foreach (String ruta in Rutas)
        {
            DataRow dr = DS.Tables[TablaCarpetas].NewRow();
            dr["Fecha"] = Fecha;
            dr["Hora"] = Hora;
            dr["Orden"] = orden;
            dr["RutaOrigen"] = ruta;
            DS.Tables[TablaCarpetas].Rows.Add(dr);
            orden++; // incrementa el número de orden
        }
        DACarpetas.Update(DS, TablaCarpetas);
    }
    catch(Exception ex)
    {
```

```
        throw new Exception(ex.Message);  
    }  
}
```

Ambos métodos reciben varios parámetros y el primero de ellos es un objeto de tipo **OleDbTransaction**, ese objeto debe ser asignado a la propiedad “**Transaction**” de cada uno de los comandos (**OleDbCommand**) creados en el constructor de la clase para el manejo de cada tabla. Sin ese valor asignado los cambios realizados en las tablas no podrán ser controlados por la transacción.

El resto de los parámetros que recibe cada método corresponden a los valores que se usarán para crear el nuevo registro en el caso de la tabla Tareas y para crear uno o varios registros en el caso de la tabla de Carpetas, dependiendo de la cantidad de elementos que contenga la lista “Rutas”.

El método “**AnularTarea**” recibe por parámetro la fecha y la hora de la tarea que se desea anular, se realiza una búsqueda con el método “**Find**” y si el registro es localizado y además su estado es pendiente se edita y el campo “Estado” recibe el valor 2 indicando que la tarea queda anulada y no se ejecutará.

El método “**EjecutarTarea**” recibe por parámetro el control Timer, necesario para poder detenerlo y reanudarlo al final del proceso. Se debe detener para que no se vuelva a ejecutar hasta que no termine completamente de realizar las copias de los archivos.

Luego toma la fecha y hora actuales del sistema y con esos valores busca en la tabla de Tareas alguna tarea que esté grabada con esos mismos valores de fecha y hora y que además esté en estado pendiente de ejecutar, si encuentra una tarea para ejecutar comienza un ciclo donde busca en la tabla de Carpetas la ruta origen de las copias a realizar, con todos esos valores conocidos se ejecuta la copia de archivos por medio del método “**CopyAll**”, el proceso se repite con el resto de carpetas que tenga la tarea. Finalmente, el registro de la tarea se edita y se modifica el estado pasando de pendiente (0) a terminada (1).

En el método “**CopyAll**” se hace uso de las clases “`DirectoryInfo`”, `Path`”, “`Directory`” y “`FileInfo`”, todas definidas en el espacio “`System.IO`” y ya conocidas en la materia Laboratorio de Programación 1 y 2.

Referencias a estas 4 clases:

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

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

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

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

Finalmente, el método “**Dispose**” libera los recursos usados por el objeto `DataSet`.

Continuamos ahora con la implementación de los eventos del primer formulario (Form1), cuyo código completo resulta así:

```
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.Diagnostics;

namespace SP2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // cargar los usuarios
            CUsuario usr = new CUsuario();
            cmbUsuarios.DisplayMember = "Nombre";
            cmbUsuarios.ValueMember = "Usuario";
```

```

        cmbUsuarios.DataSource = usr.GetUsuarios();
        // iniciar el timer
        tmrTareas.Interval = 30000;
        tmrTareas.Enabled = true;
    }

    private void tmrTareas_Tick(object sender, EventArgs e)
    {
        try
        {
            CTarea tarea = new CTarea();
            tarea.EjecutarTarea(tmrTareas);
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private bool ValidarPassword()
    {
        bool resultado = false;
        CUusuario usr = new CUusuario();
        String password =
usr.GetPasswordUsuario((int)cmbUsuarios.SelectedValue);
        if(password.CompareTo(txtPassword.Text) == 0)
        {
            resultado = true;
        }
        return resultado;
    }

    private bool ValidarDatos()
    {
        DateTime fecha = dtpFecha.Value.Date;
        String hora = dtpHora.Value.ToShortTimeString().Substring(0,5);
        if(fecha.CompareTo(DateTime.Now.Date) < 0)
        {
            throw new Exception("La fecha no puede ser menor a la fecha
actual");
        }
        if (fecha.CompareTo(DateTime.Now.Date) == 0 &&
hora.CompareTo(DateTime.Now.ToString("HH:mm")) < 0)
        {
            throw new Exception("La hora no puede ser menor a la hora
actual");
        }
        if(txtRutaDestino.Text == "")
        {
            throw new Exception("Debe seleccionar la ruta destino de las
copias");
        }
        if (lstCarpetas.Items.Count == 0)
        {
            throw new Exception("Debe seleccionar al menos una carpeta para
copiar");
        }
        return true;
    }
}

```

```

private void InicializarInterfaz()
{
    txtPassword.Text = "";
    txtRutaDestino.Text = "";
    lstCarpetas.Items.Clear();
    txtObservacion.Text = "";
}

private void btnAceptarTarea_Click(object sender, EventArgs e)
{
    try
    {
        if (ValidarPassword())
        {
            if (ValidarDatos())
            {
                CTarea tarea = new CTarea();
                // crear una lista con los nombres de las carpetas a copiar
                List<String> lista = new List<string>();
                for (int i = 0; i < lstCarpetas.Items.Count; i++)
                {
                    lista.Add(lstCarpetas.Items[i].ToString());
                }
                DateTime fecha = dtpFecha.Value.Date;
                // la hora se guarda en formato hora:minutos,
                // con la hora en formato 24 horas
                String hora = dtpHora.Value.ToString("HH:mm");
                // obtener el número del usuario seleccionado en el
                comboBox
                int usuario = (int)cmbUsuarios.SelectedValue;
                String rutaDestino = txtRutaDestino.Text;
                String observacion = txtObservacion.Text;
                tarea.AddTarea(fecha, hora, usuario, rutaDestino,
                observacion, lista);
                // restablecer la interfaz
                InicializarInterfaz();
            }
        }
        else
        {
            MessageBox.Show("Password incorrecto!.");
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void btnElegir_Click(object sender, EventArgs e)
{
    txtRutaDestino.Text = "";
    FolderBrowserDialog dlg = new FolderBrowserDialog();
    dlg.Description = "Seleccione la carpeta destino";
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        txtRutaDestino.Text = dlg.SelectedPath;
    }
}

```

```

        dlg.Dispose();
    }

    private void cmbUsuarios_SelectedIndexChanged(object sender, EventArgs e)
    {
        txtPassword.Text = "";
    }

    private void btnAgregar_Click(object sender, EventArgs e)
    {
        FolderBrowserDialog dlg = new FolderBrowserDialog();
        dlg.Description = "Seleccione la carpeta a copiar";
        if (dlg.ShowDialog() == DialogResult.OK)
        {
            lstCarpetas.Items.Add(dlg.SelectedPath);
        }
        dlg.Dispose();
    }

    private void btnQuitar_Click(object sender, EventArgs e)
    {
        if(lstCarpetas.SelectedIndex != -1)
        {
            lstCarpetas.Items.RemoveAt(lstCarpetas.SelectedIndex);
        }
    }

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

```

En el evento “**Load**” se inicia la ejecución del Timer, asignado un intervalo de 30 segundos.

En el evento “**Tick**” del timer se crea un objeto de tipo CTarea y se invoca al método “EjecutarTarea” para controlar y ejecutar la tarea de copiar los archivos de backup si hubiera alguna programada para la fecha y hora actual.

Los métodos “**ValidarPassword**” y “**ValidarDatos**” controlan que toda la información ingresada en el formulario sea correcta y válida para poder grabarla en la base de datos, las principales validaciones son el password del usuario seleccionado y que la fecha y hora para la nueva tarea correspondan a un tiempo futuro. Los errores detectados generan excepciones y no permitirán grabar ningún registro.

El método “**IniciarInterfaz**” se ocupa de colocar los controles del formulario en su estado inicial.

El evento “Click” del botón “**AceptarTarea**” ejecuta las validaciones de datos que comentamos anteriormente y si todo es correcto hace uso de un objeto de la clase “CTarea” invocando el método “AddTarea” pasando por parámetro todos los valores ingresados por el usuario en los controles del formulario. Finalmente ejecuta el método “IniciarInterfaz” para restaurar el estado del formulario.

En el evento “Click” del botón “**Elegir**” se utiliza un “FolderBrowserDialog” para que el usuario pueda seleccionar la carpeta destino de las copias backup, el valor obtenido se asigna al control TextBox llamado “txtRutaDestino”.

En el evento “Click” del botón “**Agregar**” también se trabaja con un “FolderBrowserDialog” pero la ruta obtenida de cada carpeta a incluir en el backup se va agregando al control ListBox de nombre “lstCarpetas”.

En el caso del evento “Click” del botón “**Quitar**”, se controla que exista algún ítem del ListBox seleccionado para luego eliminarlo.

Finalmente, en el evento “Click” del botón “**VerTareas**” se crea un objeto de la clase “Form2” y se ejecuta el método “ShowDialog” para que el usuario pueda trabajar con él.

Y para finalizar con el código veamos la implementación del segundo formulario: 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 SP2
{
    public partial class Form2 : Form
```

```

{
    public Form2()
    {
        InitializeComponent();
    }

    private void Form2_Load(object sender, EventArgs e)
    {
        CargarTareas();
    }

    private void CargarTareas()
    {
        grTareas.Rows.Clear();
        // cargar la grilla con los datos de las tareas
        String[] Estados = new String[3] { "Pendiente", "Terminada", "Anulada" };
    }

    CUuario usr = new CUuario();
    CTarea tarea = new CTarea();
    DataTable TablaTarea = tarea.GetTareas();
    foreach (DataRow dr in TablaTarea.Rows)
    {
        String Nombre = usr.GetUsuario((int)dr["usuario"]);
        grTareas.Rows.Add(dr["Fecha"].ToString().Substring(0, 10),
dr["Hora"].ToString(),
Nombre, Estados[(int)dr["Estado"]]);
    }
}

private void btnAnular_Click(object sender, EventArgs e)
{
    if(grTareas.SelectedRows.Count == 1)
    {
        try
        {
            DateTime fecha =
DateTime.Parse(grTareas.SelectedRows[0].Cells[0].Value.ToString());
            String hora =
grTareas.SelectedRows[0].Cells[1].Value.ToString();
            CTarea tarea = new CTarea();
            tarea.AnularTarea(fecha, hora);
            CargarTareas();
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
    else
    {
        MessageBox.Show("Debe seleccionar una tarea para anular");
    }
}

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

```

```
}
```

Desde el evento “Load” se ejecuta el método “**CargarTareas**” que se encarga de obtener los datos de la tabla Tareas y darles el formato correcto para mostrarlos en la grilla (control DataGridView), Utiliza las clases “CTareas” y “CUsuarios” y un arreglo de tipo string para mostrar los nombres de los estados en lugar de sus valores numéricos.

El evento “Click” del botón “**AnularTarea**” invoca el método “**AnularTarea**” del objeto tarea pasando por parámetro los valores de fecha y hora de la tarea que el usuario seleccionó en la grilla. Si no hay ninguna tarea seleccionada el botón no realiza ninguna acción.

Finalmente, el botón “**Volver**” cierra el formulario y retorna la ejecución al formulario principal (From1).

De esta forma finalizamos la resolución de la situación profesional solicitada, sugerimos ejecutar y verificar que las tareas de copias son ejecutadas correctamente en los horarios programados y que la carpeta destino contenga todos los archivos.

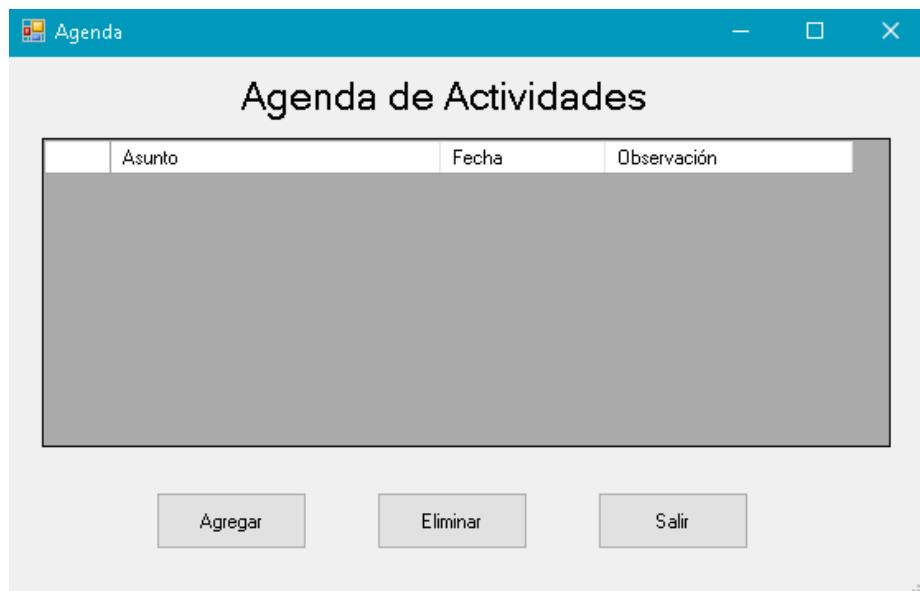
SP2/Ejercicio por resolver

La consultora para la que trabaja le solicita a usted que desarrolle una aplicación que permita a sus usuarios crear una agenda de actividades y establecer avisos de vencimientos. Los datos se registran en una base de datos tipo Access que cuenta con una tabla denominada Actividades, cuya estructura se detalla a continuación:

- IdActividad Integer
- Asunto Text (50)
- Fecha Fecha/Hora
- Observación Text (30)

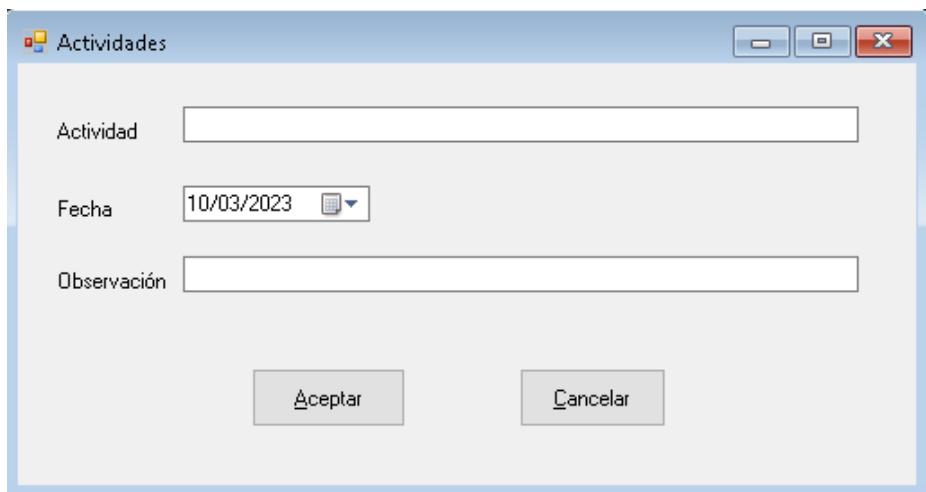
El campo IdActividad es la clave primaria de la tabla.

Al iniciar la aplicación debe presentar la siguiente interfaz gráfica:



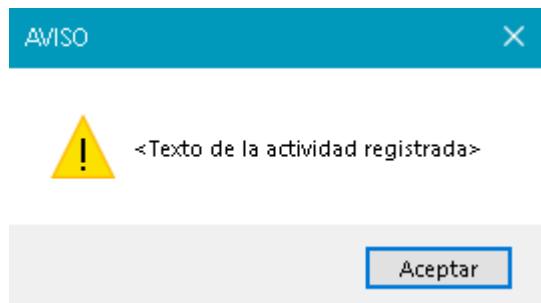
El botón “Agregar” abrirá un segundo formulario para el ingreso de nuevas actividades, el botón “Eliminar” debe permitir eliminar la actividad seleccionada en la grilla y el botón “Salir” cierra la aplicación.

Para agregar las tareas:



Se validará que el campo Actividad no esté vacío y que la fecha sea un valor futuro, el valor del campo Observación es opcional. La fecha a registrar será la seleccionada en el control DateTimePicker con la hora siempre en el valor “12:00:00”. Ese será el vencimiento de la tarea.

Y para mostrar un aviso programado:



El aviso debe mostrarse automáticamente 2 veces para cada actividad, la primera vez se mostrará 24 horas antes y la segunda vez se mostrará 12 horas antes del vencimiento registrado de la tarea.

Este ejercicio deberá ser resuelto en forma práctica creando un proyecto en Visual Studio.

SP2/Evaluación de paso

1. Indique la opción correcta.

El control DateTimePicker se usa tanto para seleccionar fechas como para seleccionar horas.

- Verdadero
- Falso

2. Indique la opción correcta.

El control DateTimePicker no permite seleccionar una fecha y una hora al mismo tiempo

- Verdadero
- Falso

3. Indique la opción correcta.

En una transacción se puede trabajar como máximo con dos tablas de la base de datos

- Verdadero
- Falso

4. Indique la opción correcta.

Una transacción en ADO .NET sólo puede controlar operaciones sobre la base de datos.

- Verdadero
- Falso

5. Indique la opción correcta.

En un control Timer el evento a programar para ejecutar las acciones periódicas del timer es:

- Timer
- Tick
- Load
- Execute

6. Indique la opción correcta.

En un control Timer el valor del intervalo de ejecución está establecido en segundos

- Verdadero
- Falso

Respuestas correctas¹⁰

¹⁰1) Verdadero. 2) Falso. 3) Falso. 4) Verdadero. 5) Tick. 6) Falso.

Situación profesional 3: Incendios

Un funcionario público le ha solicitado a usted que, como pasante del área de sistemas de la dependencia estatal que él gestiona, desarrolle una aplicación para obtener resultados de los incendios registrados en el último año en las provincias detalladas por departamento.

Las tareas a realizar son las siguientes:

1. Agregar a un control de tipo árbol un nodo raíz con la leyenda “Incendios”. Del nodo raíz dependen nodos con los nombres de las provincias. Y de cada nodo con el nombre de una provincia dependen nodos con los nombres de los departamentos que pertenecen a cada provincia.
2. Al seleccionar un nodo con el nombre de una provincia o el nombre de un departamento, se muestra en un control de tipo vista de detalle con dos columnas, el tipo de incendio y la cantidad de incendios registrada en la provincia o en el departamento (la cantidad de incendios obtenidos por cada lista depende del nodo seleccionado). Al seleccionar el nodo raíz se deberá mostrar los valores acumulados de todas las provincias.
3. También se muestra en un control barra de estado la cantidad total de incendios, es la suma de todos los incendios que aparecen en el control vista de detalle.
4. Cuando se selecciona el nodo raíz del control tipo árbol se inicializa el control vista de detalle y el control barra de estado.

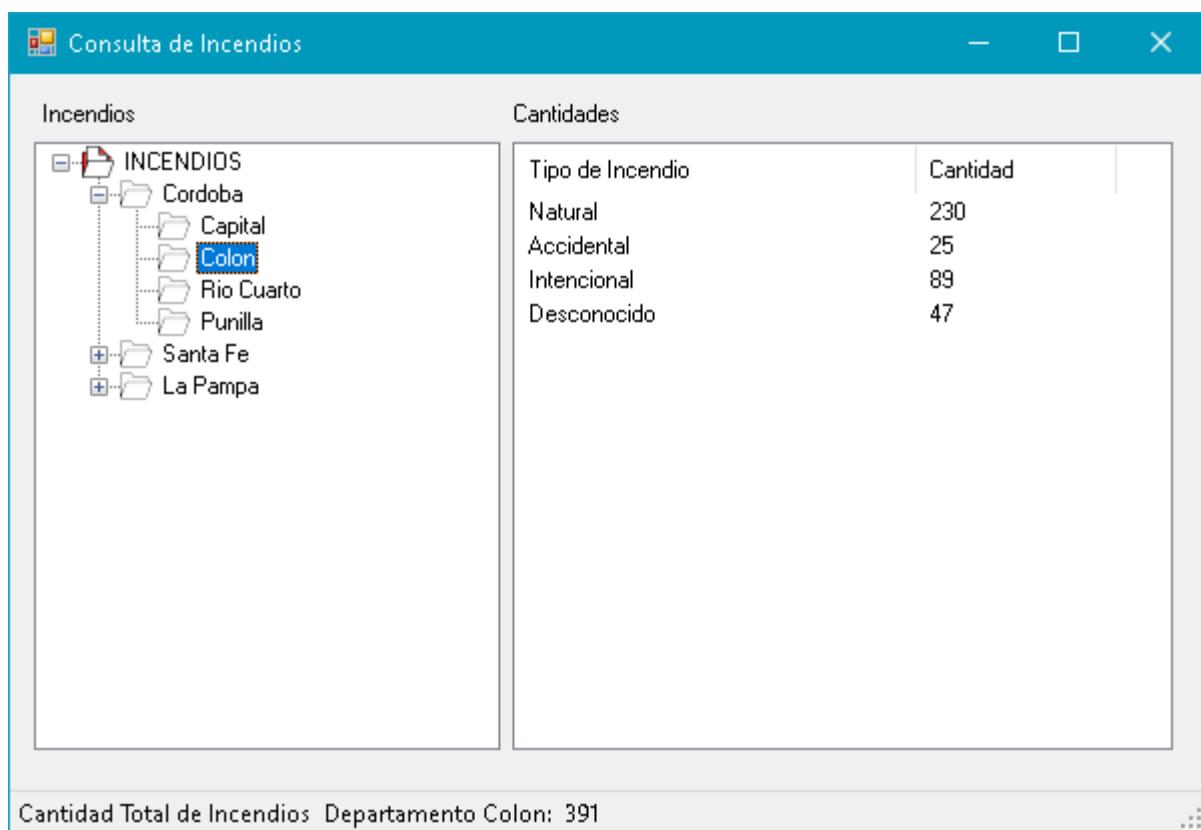
La base de datos se llama ‘Incendios.mdb’ y contiene cuatro tablas.

La tabla ‘Provincias’ contiene las provincias participantes en la consulta, sus datos son: un número para identificar la provincia y el nombre de la provincia. La clave principal de la tabla es el número de provincias.

La tabla ‘Departamentos’ contiene los departamentos de cada provincia, sus datos son: un número para identificar al departamento, el nombre del departamento y el número de provincia al que pertenece el departamento. La clave principal de la tabla es el número de departamento.

La tabla ‘TipolIncendio” contiene los diferentes tipos de incendio según el origen del mismo, sus datos son: un número para identificar el tipo y la descripción de cada tipo, la clave principal de la tabla es el número de lista.

La tabla ‘Incendios’ contiene la cantidad de incendios registrados por cada tipo en los respectivos departamentos, sus datos son: el número de departamento, el tipo de incendio y la cantidad de incendios. La clave principal de la tabla está compuesta por las dos primeras columnas.

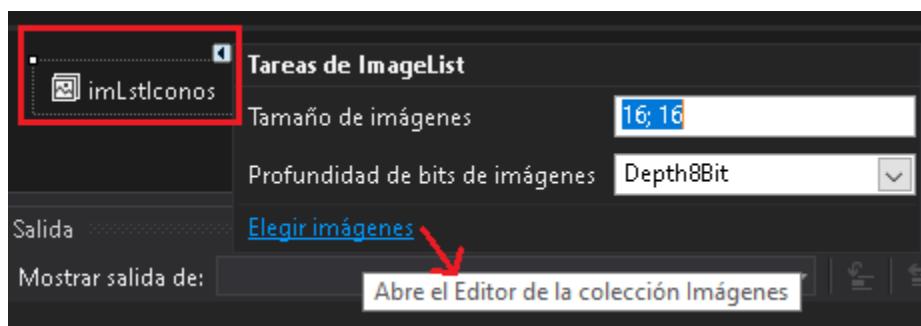


SP3/H1: Control ImageList

El control ImageList es un contenedor de imágenes. Nos permitirá mostrar los distintos tipos de carpetas cuando desarrollemos la aplicación que se nos solicitó.

Un control ImageList contiene una colección de objetos Image, a los que se puede hacer referencia mediante su índice o clave.

El control ImageList no está concebido para usarlo en solitario, sino como punto de almacenamiento central para proporcionar cómodamente imágenes a otros controles. Es un control que no se visualiza sobre el formulario, es un complemento para ser usado en combinación con otros controles que puedan visualizar iconos o imágenes, como, por ejemplo, un control PictureBox, TreeView, ListView, DataGridView, CheckBox, RadioButton, Label y algunos más. En general cualquier control que posea la propiedad “ImageList” podrá asociarse con el nombre de un control ImageList.

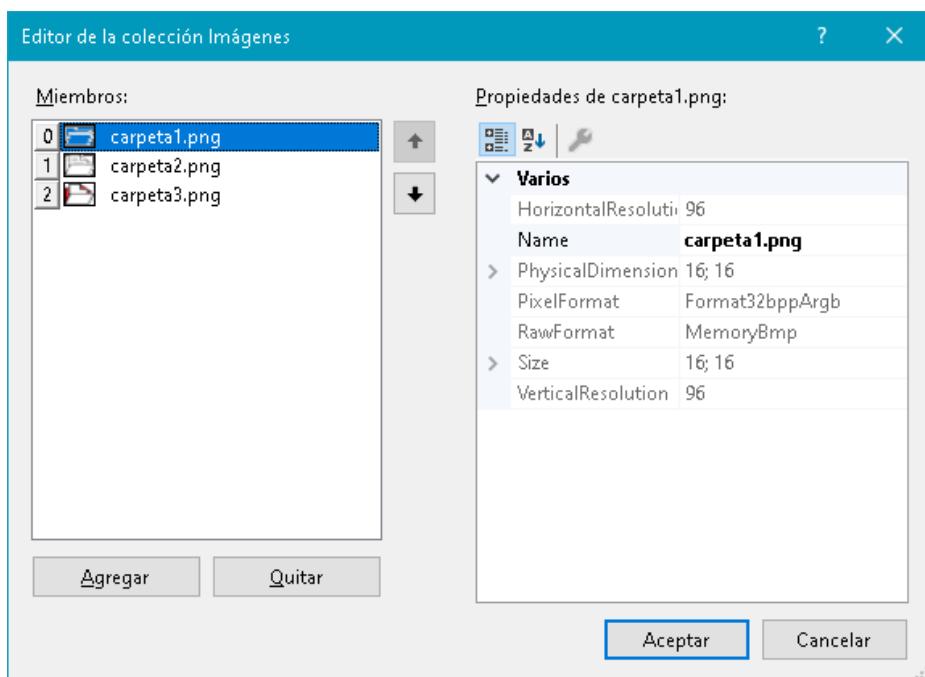


Para Agregar/Quitar Imágenes

Puede agregar/quitar imágenes a un componente ImageList de varias maneras. Puede utilizar la etiqueta inteligente asociada al componente ImageList, o quizás le resulte más cómodo agregar/quitar imágenes con la ventana de Propiedades. También puede agregar imágenes utilizando un código.

- Utilizando la ventana de propiedades:** Seleccione el componente ImageList o agregue uno al formulario. En la ventana Propiedades, haga clic en el botón de puntos suspensivos (...) situado junto a la propiedad Images.

En el Editor de la colección de imágenes, haga clic en Agregar o Quitar para agregar o quitar las imágenes de la lista.



b. **Mediante la etiqueta inteligente:** Seleccione el componente ImageList o agregue uno al formulario.

Haga clic en el glifo de la etiqueta inteligente.

En el cuadro de diálogo seleccione "Elegir imágenes".

En el Editor de la colección de imágenes, haga clic en Agregar o Quitar para agregar o quitar las imágenes de la lista.

c. **Mediante Programación:** Utilice el método Add de la propiedad Images de la lista de imágenes.

El siguiente ejemplo de código requiere que tenga un formulario con un control ImageList ya agregado y también se requiere tener agregado:

```
using System.Drawing;
```

```
Image imagen = Image.FromFile("C:\\\\Carpetas1.png");
imLstIconos.Images.Add(imagen);
```

“imLstIconos” es el nombre del control ImageList.

“Image” es la clase que maneja los objetos de tipo Image.

Para Quitar Imágenes Mediante Programación

Utilice el método RemoveAt para quitar una sola imagen, indicando el índice de la misma, o bien, el método Clear, para borrar todas las imágenes de la lista de imágenes:

```
imLstIconos.Images.RemoveAt(0);  
imLstIconos.Images.Clear();
```

Asociar una Lista de Imágenes con un Control de Windows

Para los controles comunes de Windows, se puede especificar un ImageList durante el diseño del programa, mediante el cuadro de diálogo Propiedades.

En tiempo de ejecución se puede especificar también un control ImageList utilizando la propiedad ImageList, como en el siguiente ejemplo:

```
chkImagen.ImageList = ImageList1;  
chkImagen.ImageIndex = 1;
```

Al usar el control ImageList con controles comunes de Windows, será necesario insertar todas las imágenes que se necesitarán (en orden adecuado), en el control ImageList antes de enlazarlo al control secundario. Una vez que ImageList está enlazado a un control secundario, no se podrán eliminar ni insertar imágenes en la mitad de la colección ListImages. A pesar de todo, se podrá agregar imágenes al final de la colección.

Una vez asociado un ImageList con un control común de Windows, se usará el valor de la propiedad ImageIndex o ImageKey para hacer referencia a un objeto ListImage en un procedimiento.

Puede encontrar más detalles sobre el uso de la clase ImageList en este enlace:

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

SP3/Autoevaluación 1

1. Indique la opción correcta

¿Cómo se hace referencia a un objeto ListItem en un procedimiento?:

- Con propiedad ImageIndex o KeyImage
- Con propiedad Index o Image
- Con propiedad ImageIndex o ImageKey

2. Indique la opción correcta

¿A qué controles le puede proporcionar imágenes el control ImageList?:

- ListView, TreeView
- ComboBox, ListBox
- Chart, DataGridView

3. Indique la opción correcta

¿Para qué sirve el control ImageList?:

- Mostrar Imágenes
- Contener Imágenes
- Guardar Imágenes

4. Indique la opción correcta

El control TreeView se puede asociar al ImageList, pero debemos agregar los Nodos.

- Verdadero
- Falso

5. Indique la opción correcta

Se pueden agregar o eliminar imágenes a la lista una vez que ImageList está enlazado a un control secundario.

- Verdadero
- Falso

6. Indique la opción correcta

El control ImageList se utiliza independientemente de otros controles para ver imágenes.

- Verdadero
- Falso

Respuestas correctas¹¹

¹¹1) Con propiedad ImageIndex o ImageKey. 2) ListView, TreeView. 3) Contener Imágenes. 4) Verdadero. 5) Verdadero. 6) Falso.

SP3/H2: Controles de presentación jerárquica de datos

Para resolver lo solicitado en la situación profesional, debemos tener en cuenta la utilización de controles avanzados de presentación de información. Este tipo de controles, además de facilitar la interpretación de la información en la interfaz, hará más ágil la carga o búsqueda de información para su posterior tratamiento.

Control para visualización de la información en forma de árbol (TreeView) En la situación profesional, podemos observar que debemos mostrar las provincias y departamentos en forma de árbol. Para hacer lo que nos solicitan, utilizaremos el control TreeView. El mismo está diseñado para mostrar datos de naturaleza jerárquica, como árboles organizativos, las entradas de un índice, o los archivos y directorios de un disco.

Ejemplos:



Un control TreeView puede aplicarse para:

- Crear un árbol organizativo que pueda ser manipulado por el usuario.
- Crear un árbol que muestre, al menos, dos o más niveles de una base de datos.

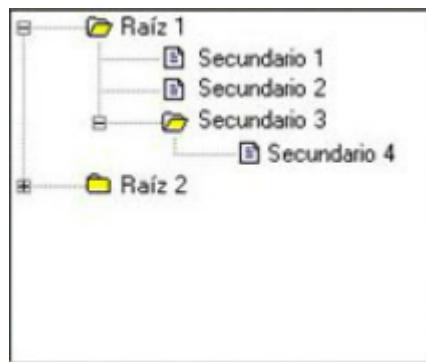
El control TreeView se usa a menudo junto con el control ListView. Esta combinación permite al usuario atravesar diferentes niveles jerárquicos: el control TreeView muestra la estructura de nivel superior, mientras que el control ListView muestra los

conjuntos individuales de registros al seleccionar cada nodo. Siempre que se seleccione un nodo del control TreeView, el código llenará el control ListView con los conjuntos individuales de registros.

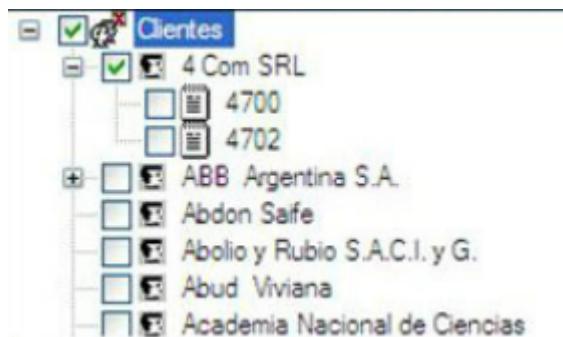
Propiedades del control TreeView

Un "árbol" se compone de ramas sucesivas de "nodos". Cada nodo de la vista de árbol podría contener otros nodos, llamados secundarios. Puede mostrar nodos primarios o nodos que contienen nodos secundarios, como expandidos o contraídos. Un nodo puede expandirse o contraerse dependiendo de si tiene o no nodos secundarios (nodos que parten de él). En el nivel superior están los nodos "raíz", y cada nodo raíz puede tener cualquier número de nodos secundarios. El número total de nodos no está limitado (salvo por las restricciones del sistema). En la siguiente figura se muestra un árbol con dos nodos raíz.

"Raíz 1" tiene tres nodos secundarios y "Secundario 3" tiene, a su vez, un nodo secundario. "Raíz 2" tiene nodos secundarios, como indica el signo "+", pero está sin expandir.



También puede mostrar una vista de árbol con casillas de verificación junto a los nodos, estableciendo la propiedad Checkboxes de la vista de árbol en true. De este modo, es posible activar o desactivar nodos mediante programación, determinando la propiedad Checked del nodo en true o false.



La apariencia del control TreeView se puede cambiar estableciendo algunas de sus propiedades de presentación y estilo. Si se establece ShowPlusMinus en true, se muestra un botón con el signo más (+) o con el signo menos (-), junto a cada nodo que se puede expandir o contraer, según corresponda.

Si se establece la propiedad ShowRootLines en true, el TreeView muestra líneas que unen entre sí todos los nodos de árbol raíz. Se pueden mostrar líneas que unan los nodos secundarios con su correspondiente nodo raíz, estableciendo la propiedad ShowLines en true.

Si se establece la propiedad HotTracking en true, cambia la apariencia de las etiquetas de los nodos cuando el puntero del mouse pasa sobre ellas, las etiquetas adquieren el aspecto de un hipervínculo. También se puede personalizar totalmente la apariencia del control TreeView. Para ello, establezca la propiedad DrawMode en un valor distinto del Normal y controle el evento DrawNode.

Se pueden mostrar imágenes junto a los nodos del árbol asignando un ImageList a la propiedad ImageList del control y haciendo referencia al valor de índice de una Imagen de ImageList. Utilice las propiedades siguientes para asignar las imágenes:

Establezca la propiedad ImageIndex en el valor de índice de la Imagen que desea que se muestre cuando no está seleccionado un nodo del árbol.

Determine la propiedad SelectedImageIndex en el valor de índice de la Imagen que desea que se muestre cuando esté seleccionado un nodo del árbol

Nodos del control TreeView

La propiedad Nodes del control TreeView contiene la lista de nodos del nivel superior de la vista de árbol. Cada nodo de un árbol es en realidad un objeto TreeNode programable que pertenece a la colección TreeNodeCollection. Se puede acceder a esta colección a través de la propiedad Nodes del control TreeView. Como en otras colecciones, cada miembro de la colección tiene un valor único en las propiedades Index y Key, lo que permite el acceso a las propiedades del nodo.

Las imágenes a las que hacen referencia los valores de las propiedades ImageIndex y SelectedImageIndex del control Treeview son las imágenes predeterminadas que muestran todos los nodos de árbol asignados a la colección Nodes. Los nodos individuales del árbol pueden reemplazar las imágenes predeterminadas estableciendo las propiedades ImageIndex y SelectedImageIndex del nodo.

Los nodos de árbol se pueden expandir para mostrar el siguiente nivel de nodos secundarios. El usuario puede expandir también el nodo haciendo clic en el botón con el signo más (+), si se muestra al lado del nodo, o bien, llamando al método Expand.

Para expandir todos los niveles de nodos secundarios de un nodo primario, llame al método ExpandAll. El nivel secundario del nodo se puede contraer llamando al método Collapse, o presionando el botón con el signo menos (-) si se muestra al lado del nodo. También se puede llamar al método Toggle para alternar los estados expandido y contraído del nodo, que cambiará del estado opuesto al actual, ya sea expandido o contraído.

Relaciones entre nodos

Cada nodo del árbol puede ser secundario o primario, según su relación con otros nodos y tener propiedades que se pueden utilizar para explorar la vista del árbol:

FirstNode: obtiene el primer nodo secundario en la colección de nodos que se almacena en la propiedad Nodes del actual nodo. Si el nodo actual no tiene ningún nodo secundario, la propiedad FirstNode devolverá referencia de objeto null.

LastNode: obtiene el último nodo secundario en la colección de nodos que se almacena en la propiedad Nodes del actual nodo. Si éste no tiene ningún nodo secundario, la propiedad LastNode devolverá referencia de objeto null.

NextNode: obtiene el siguiente nodo relacionado en la colección de nodos que se almacena en la propiedad Nodes del nodo primario, del nodo actual. Si no hay ningún nodo a continuación, la propiedad NextNode devolverá referencia de objeto null.

PrevNode: obtiene el anterior nodo relacionado en la colección de nodos que se almacena en la propiedad Nodes del nodo primario, del nodo actual. Si no hay ningún nodo anterior, la propiedad PrevNode devolverá referencia de objeto null.

Parent: obtiene el nodo primario del actual nodo de árbol. Si el nodo está en el nivel de raíz, la propiedad Parent devolverá referencia de objeto null. El valor de la propiedad Parent es el nodo principal, del nodo actual. De existir los nodos secundarios del nodo actual, se enumeran en su propiedad Nodes. El propio control TreeView dispone de la propiedad TopNode, que es el nodo raíz de toda la vista del árbol.

Para Agregar/Quitar nodos del árbol

Puede agregar/quitar nodos a un control TreeView de tres maneras que describiremos a continuación:

- a. **Utilizando la etiqueta inteligente asociada al control TreeView.** Seleccione el control TreeView o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente () En el cuadro de diálogo Tareas de TreeView, seleccione Editar Nodos. En el Editor de la colección de nodos, haga clic en Agregar Raíz o Agregar Secundario para añadir un nodo primario o secundario, o presione el botón Quitar () para eliminar nodos.
- b. **Puede hacerlo desde la ventana de Propiedades:** Seleccione el control TreeView o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón de puntos suspensivos () situado junto a la propiedad Nodes. En el Editor de la colección de nodos, haga clic en Agregar Raíz o Agregar Secundario para añadir un nodo primario o secundario, o presione el botón Quitar () para eliminar nodos.

- c. Puede hacerlo utilizando código (o programación): Para agregar un nodo utilizando código debe utilizar el método Add de la propiedad Nodes de la vista de árbol. Este método agrega el nodo al final de la colección:
- TreeView.Nodes.Add(String): agrega un nuevo nodo con el texto de etiqueta especificado.
 - TreeView.Nodes.Add(Nodo): agrega un nodo anteriormente creado.
 - TreeView.Nodes.Add(String, String): crea un nuevo nodo con la clave y texto especificados.
 - TreeView.Nodes.Add(String, String, Int32): crea un nuevo nodo con la clave, texto e imagen(índice) especificados.
 - TreeView.Nodes.Add(String, String, String): crea un nuevo nodo con la clave, texto e imagen(clave) especificados.
 - TreeView.Nodes.Add(String, String, Int32, Int32): crea un nuevo nodo con la clave, texto, imagen(índice) cuando no está seleccionado, imagen(índice) cuando está seleccionado.
 - TreeView.Nodes.Add(String, String, String, String): crea un nuevo nodo con la clave, texto, imagen(clave) cuando no está seleccionado, imagen(clave) cuando está seleccionado.

El ejemplo de código siguiente requiere que tenga un formulario con un control TreeView ya agregado:

```
TreeNode nodo = new TreeNode("Texto del nodo");
treeView1.Nodes.Add(nodo);
```

Para quitar un nodo mediante programación debe utilizar el método Remove de la propiedad Nodes de la vista del árbol para quitar un solo nodo, o el método Clear para borrar todos los nodos.

El ejemplo de código siguiente requiere que tenga un formulario con un control TreeView ya agregado:

```
treeView1.Nodes[0].Remove();

treeView1.Nodes.Clear();
```

En el primer ejemplo se elimina el primer nodo del treeview, si ese nodo tuviera nodos hijos éstos también se borrarán completamente.

En el segundo caso se eliminan todos los nodos del control treeview (método Clear).

Recorrer todos los nodos del árbol

Para realizar esta actividad debemos:

1. Crear un procedimiento recursivo que compruebe cada nodo.
2. Llamar al procedimiento.

A continuación, se muestra cómo visualizar la propiedad Text de cada nodo:

```
private void ListarNodos()
{
    foreach(TreeNode nodo in treeView1.Nodes)
    {
        // comienza con los nodos raiz del treeview
        RecorrerRecursivo(nodo);
    }
}

private void RecorrerRecursivo(TreeNode nodo)
{
    Trace.WriteLine(nodo.Text); // muestra en la consola el texto del nodo
    foreach(TreeNode n in nodo.Nodes)
    {
        RecorrerRecursivo(n); // repite para cada nodo hijo
    }
}
```

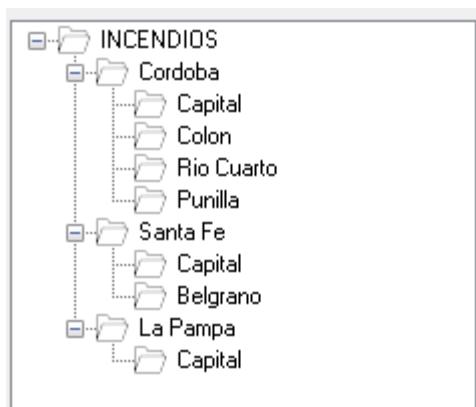
La salida por consola muestra:

```

Salida ::::::::::::
Mostrar salida de: Depurar
'SP3.exe' (CLR v4.0.30319: SP3.exe):
'SP3.exe' (CLR v4.0.30319: SP3.exe):
INCENDIOS
Cordoba
Capital
Colon
Rio Cuarto
Punilla
Santa Fe
Capital
Belgrano
La Pampa
Capital

```

¿Qué son los textos de estos nodos cargados en el treeview?



Los eventos del control TreeView son:

- AfterSelect:** se produce después de seleccionar el nodo, es el evento predefinido.
- AfterCheck:** se produce después de activarse la casilla de verificación del nodo.
- AfterCollapse:** se produce después de contraerse el nodo.
- AfterExpand:** se produce después de expandirse el nodo.

¿Cómo determinar en qué nodo del TreeView se hizo clic?

Cuando se trabaja con el control TreeView, una tarea frecuente es determinar en qué nodo se hizo clic, para lo cual trabajaremos con el evento AfterXXXX mencionados en el punto anterior, por ejemplo, **AfterSelect:** se debe comprobar la clase TreeViewEventArgs, que contiene datos relacionados con el evento provocado por el usuario. La propiedad Node obtiene el nodo que se ha comprobado, expandido, contraído o seleccionado.

En el siguiente ejemplo, el controlador de eventos recibe un argumento de tipo TreeViewEventArgs, que contiene datos relacionados con este evento. Luego, se utiliza la propiedad Node, que contiene una referencia al nodo cliqueado:

```
private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
{
    MessageBox.Show(e.Node.Text); // muestra el texto del nodo que se seleccionó
}
```

Puede consultar más información en este enlace:

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

SP3/Autoevaluación 2

1. Indique la opción correcta

Los nodos se pueden expandir o contraer sólo por la acción del usuario.

- Verdadero
- Falso

2. Indique la opción correcta

La propiedad Nodes permite acceder a los nodos del árbol.

- Verdadero
- Falso

3. Indique la opción correcta

Solo se pueden agregar o quitar nodos utilizando el editor de la colección de nodos.

- Verdadero
- Falso

4. Indique la opción correcta

¿Cuál es la aplicación de un TreeView?

- Mostrar información aleatoria
- Mostrar información en orden jerárquico
- Mostrar información en orden
- Mostrar información de base de datos

5. Indique la opción correcta

La propiedad Parent obtiene el nodo primario del nodo actual.

- Verdadero
- Falso

Respuestas correctas¹²

¹²1) Falso. 2) Verdadero. 3) Falso. 4) Mostrar información en orden jerárquico 5) Falso.

SP3/H3: Controles para visualización de información en forma de lista (ListView)

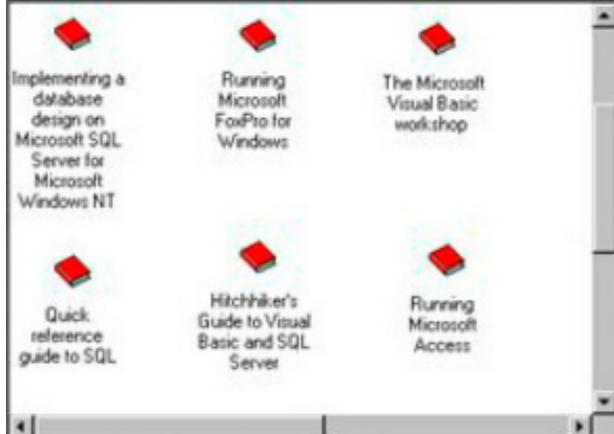
El control **ListView** muestra una lista de elementos con íconos. Este control es idóneo para representar subconjuntos de datos (como los miembros de una base de datos) u objetos discretos (como plantillas de documentos).

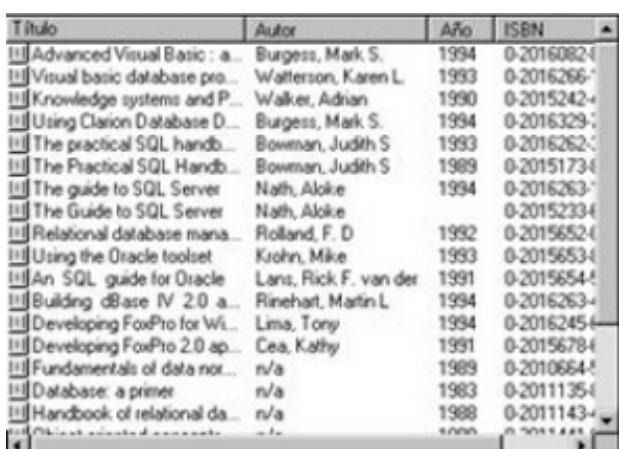
Un control ListView puede aplicarse:

- A mostrar el resultado de una consulta de una base de datos.
- A mostrar todos los registros de una tabla de una base de datos.

Junto con un control TreeView; y dar al usuario una visión ampliada de la información relacionada a un nodo del control TreeView.

El control tiene cuatro modos de vista: LargeIcon, SmallIcon, List y Details. Cada uno de ellos tiene una ventaja en particular con respecto a los demás. Algunos se indican en la tabla siguiente:

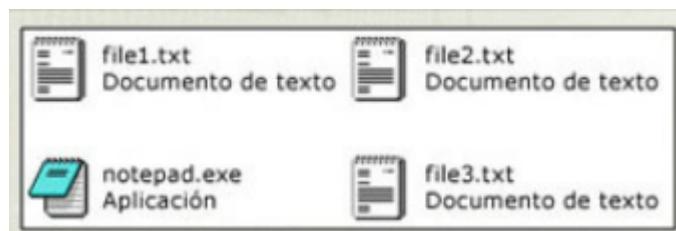
Modo de	Ventaja	Imagen
Vista LargeIcon	Cada elemento aparece como un ícono de tamaño normal debajo del cual figura una etiqueta.	

Vista SmallIcon	Cada elemento aparece como un ícono pequeño a cuya derecha figura una etiqueta.	
Vista List	Cada elemento aparece como un ícono pequeño a cuya derecha figura una etiqueta. Los elementos están organizados en columnas sin encabezado.	
Vista Details	Cada elemento aparece en una línea independiente con información más detallada acerca de cada elemento y está organizada en columnas. La columna situada más a la izquierda contiene un pequeño ícono y una etiqueta, y las columnas siguientes contienen los subelementos especificados por la aplicación. Las columnas incluyen un encabezado que puede mostrar un título correspondiente a la columna. El usuario puede cambiar el tamaño de cada columna, en tiempo de ejecución.	

Un modo de vista adicional, Mosaico. La característica de la vista en mosaico del control ListView permite ofrecer un equilibrio visual entre la información gráfica y la textual.

La información textual mostrada de un elemento en la vista en mosaico es igual que la información de columna definida para la vista de detalles.

La vista en mosaico utiliza un ícono de 32 x 32 píxeles y varias líneas de texto, como se muestra en las siguientes imágenes:



Mostrar Íconos

El control ListView puede mostrar íconos procedentes de tres listas de imágenes:

- Las vistas List, Details y SmallIcon muestran imágenes procedentes de la lista de imágenes especificada en la propiedad SmallImageList.
- La vista LargeIcon muestra imágenes procedentes de la lista de imágenes especificada en la propiedad LargeImageList.

Asimismo, una vista de lista puede mostrar un conjunto adicional de íconos, que se establece en la propiedad StatImageList, junto a los íconos grandes o pequeños.

Para Agregar/Quitar elementos

Puede agregar/quitar elementos a un control ListView de tres maneras:

- a. **Puede utilizar la etiqueta inteligente asociada al control ListView:** Seleccione el control ListView o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente (). En el cuadro de diálogo Tareas de ListView, seleccione

Editar Elementos. En el Editor de la colección de Items, haga clic en Agregar o Quitar, para agregar y quitar ítems.

- b. **Puede utilizar la ventana de Propiedades:** Seleccione el control ListView o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón de puntos suspensivos () situado junto a la propiedad Items. En el Editor de la colección de Items, haga clic en Agregar o Quitar, para agregar y quitar items.
- c. **Puede utilizar código:** El proceso para agregar un elemento a un control ListView consiste básicamente en especificar el elemento y asignarle propiedades.

Utilice el método Add de la propiedad Items del control ListView:

- ListView.Items.Add(String): agrega un nuevo elemento con el texto especificado. ListView.Items.Add(Item): agrega un elemento anteriormente creado.
- ListView.Items.Add(String, String): crea un nuevo elemento con el texto y la imagen(clave) especificados.
- ListView.Items.Add(String, Int32): crea un nuevo elemento con el texto e imagen(índice) especificados.
- ListView.Items.Add(String, String, String): crea un nuevo elemento con la clave, texto e imagen(clave) especificados.
- ListView.Items.Add(String, String, Int32): crea un nuevo elemento con la clave, texto e imagen(índice) especificados.

Quitar elementos

Para quitar elementos mediante programación utilice el método RemoveAt o Clear de la propiedad Items. El método RemoveAt quita un solo elemento; el método Clear quita todos los elementos de la lista.

Para mostrar imágenes en una vista de lista

Será necesario contar con un control de tipo ImageList cargado con algunas imágenes o iconos para usar como repositorio desde el cuál se tomarán las imágenes para mostrar sobre el control ListView.

1. Establezca la propiedad adecuada (SmallImageList, LargeImageList o Statelist) en el componente ImageList existente que desee utilizar.
2. Establezca la propiedad ImageIndex o StatelistIndex para cada elemento de la lista que tenga un ícono asociado.

```
ListView1.SmallImageList = ImageList1;
ListView1.Items(0).ImageIndex = 3;
```

Para agregar/quitar columnas en una vista de detalle

En la vista Detalles, el control ListView puede mostrar varias columnas para cada elemento de la lista. Puede utilizar las columnas para mostrar al usuario información de diversos tipos respecto de cada elemento de la lista.

Por ejemplo, una lista de libros puede mostrar el título, el autor, el año, etc.

Título	Autor	Año	ISBN
dbASE IV : complete reference for...	Hergert, Douglas.	1989	1-5561!
dbASE IV : programmer's quick ref...	Viescas, John	1989	1-5561!
Hitchhiker's Guide to Visual Basic ...	Vaughn, William R.	1996	1-5561!

Puede agregar/quitar columnas a un control ListView de varias maneras. Utilizando la etiqueta inteligente asociada al control ListView, o quizás desde la ventana de Propiedades. También puede agregar/quitar columnas utilizando un código.

- a. **Utilizando la ventana de propiedades:** Seleccione el control ListView o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón de puntos suspensivos (...) situado junto a la propiedad Columns. En el Editor de la colección de Columnas haga clic en Agregar o Quitar, para agregar y quitar columnas.
- b. **Mediante la etiqueta inteligente:** Seleccione el control ListView o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente (...). En el cuadro de diálogo Tareas de ListView, seleccione Editar Columnas. En el Editor de la colección de Columnas haga clic en Agregar o Quitar, para agregar y quitar columnas.

c. **Mediante programación:** El proceso de agregar una columna a un control ListView consiste básicamente en especificar la columna y asignarle propiedades. Utilice el método Add de la propiedad Columns del control ListView:

- ListView.Columns.Add(String): agrega una nueva columna con el texto especificado.
- ListView.Columns.Add(Columna): agrega una columna anteriormente creada.
- ListView.Columns.Add(String, String): crea una nueva columna con la clave y texto especificados.
- ListView.Columns.Add(String, Int32): crea una nueva columna con el texto y ancho especificados.
- ListView.Columns.Add(String, Int32, HorizontalAlignment): crea una nueva columna con el texto, ancho y valor de HorizontalAlignment.
- ListView.Columns.Add(String, String, Int32): crea una nueva columna con la clave, texto y ancho especificados.
- ListView.Columns.Add(String, String, Int32, HorizontalAlignment, Int32): crea una nueva columna con la clave, texto, ancho, valor de HorizontalAlignment y valor de índice de la imagen que se va a mostrar en la columna.
- ListView.Columns.Add(String, String, Int32, HorizontalAlignment, String): crea una nueva columna con la clave, texto, ancho, valor de HorizontalAlignment y valor clave de la imagen que se va a mostrar en la columna.

Los valores posibles de HorizontalAlignment son:

- Center: el texto se alinea en el centro.
- Left: el texto se alinea a la izquierda.
- Right: el texto se alinea a la derecha.

Para mostrar subelementos en las columnas

El control ListView puede mostrar texto adicional, o subelementos, para cada elemento de la vista Details. La primera columna muestra el texto del elemento; por ejemplo, el número del empleado. La segunda, tercera y siguientes columnas muestran el primero, segundo y siguientes subelementos asociados.

Para agregar subelementos a un elemento de la lista:

1. Agregue las columnas necesarias. Como la primera columna mostrará la propiedad Text del elemento, necesitará una columna más por cada subelemento.
2. Llame al método Add de la colección que devuelve la propiedad SubItems de un elemento:
 - ListView.Items(indice).SubItems.Add(SubItem): agrega un subitem creado anteriormente.
 - ListView.Items(indice).SubItems.Add(String): agrega un subítem con el texto especificado.
 - ListView.Items(indice).SubItems.Add(string, Color, Color, Font): agrega un subítem con el texto, color del primer plano, del fondo y el tipo de letra del subelemento.

El siguiente ejemplo establece el nombre de una calle y su numeración, para un elemento de la lista:

```
ListView1.Items(0).SubItems.Add("Calle Rondeau");
ListView1.Items(0).SubItems.Add("165");
```

Recuerde que la primera columna del control ListView muestra la propiedad text del Item. El resto de las columnas corresponden a los sub ítems.

Más detalles sobre esta clase:

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

1. Indique la opción correcta

El siguiente código limpia un control ListView: ListView1.Items.Clear();

- Verdadero
- Falso

2. Indique la opción correcta

¿Qué tipo de vista se asigna para que el control ListView muestre las columnas?

- SmallIcon
- LargeIcon
- Details
- List

3. Indique la opción correcta

¿Cuál es la aplicación de un ListView?

- Junto con un control TreeView; y dar al usuario una visión ampliada de un nodo del control TreeView.
- Junto con un control ComboBox; y dar al usuario una visión ampliada de un nodo del control.
- Junto con un control ListBox; y dar al usuario una visión ampliada de un nodo del control ListView.

4. Indique la opción correcta

¿Cuál es el tipo de vista que tiene el control ListView para íconos pequeños?

- SmallIcon
- LargeIcon
- Details
- List

5. Indique la opción correcta

Para borrar un elemento de un listView debemos ingresar:

```
ListView1.Items.Add("Elemento-3", 3)
```

Verdadero

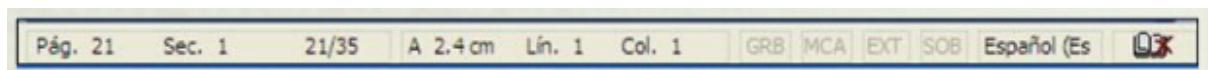
Falso

Respuestas correctas¹³

¹³1) Verdadero. 2) Details. 3) Junto con un control TreeView; y dar al usuario una visión ampliada de un nodo del control TreeView. 4) SmallIcon. 5) Falso.

SP3/H4: Control StatusStrip, barras de estado

Los controles StatusStrip muestran información sobre objetos que se van a visualizar en una ventana o cuadro de diálogo: los componentes del objeto o información contextual respecto a la operación del objeto dentro de la aplicación.



Un control StatusStrip puede aplicarse a:

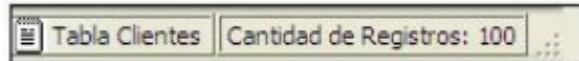
- Informar al usuario sobre las medidas de una tabla de una base de datos, por ejemplo, el número de registros y la posición actual en la base de datos.
- Indicar el estado de ciertas teclas (por ejemplo, bloq mayús o bloq num).

Las propiedades más importantes que podemos mencionar del control StatusStrip son:

BackColor	obtiene o establece el color de fondo del control.
BackGroundImage	obtiene o establece la imagen de fondo que se muestra en el control.
Dock	obtiene o establece que los bordes del control se acoplarán a su control principal (generalmente un formulario), y determina cómo se cambia el tamaño de un StatusStrip, con su elemento primario.
Font	obtiene o establece la fuente utilizada para mostrar texto en el control.
Items	obtiene todos los elementos que puede contener el objeto StatusStrip.
ShowItemToolTips	obtiene o establece un valor que indica si se muestra información sobre herramientas para StatusStrip.

Normalmente, un control StatusStrip está compuesto de objetos ToolStripStatusLabel, que muestran textos, íconos o ambos.

El control ToolStripStatusLabel representa el panel de un control StatusStrip. Puede contener texto o un ícono que refleja el estado de una aplicación.



Las propiedades más importantes que podemos mencionar de este control son:

BorderSides	obtiene o establece un valor que indica qué lados del control ToolStripStatusLabel tienen bordes.
BorderStyle	obtiene o establece el estilo de los bordes del control.
DisplayStyle	obtiene o establece si se muestran el texto y las imágenes en un ToolStripItem,
Font	obtiene o establece la fuente del texto que muestra el elemento.
Image	obtiene o establece la imagen que se muestra en un control ToolStripItem.
Text	obtiene o establece el texto que se mostrará en el elemento.
 TextAlign	obtiene o establece la alineación del texto en un ToolStripLabel.
ToolTipText	obtiene o establece el texto que aparece como ToolTip (pequeña ventana emergente rectangular que muestra una breve descripción de la finalidad de un control cuando el usuario sitúa el puntero del mouse sobre el control).

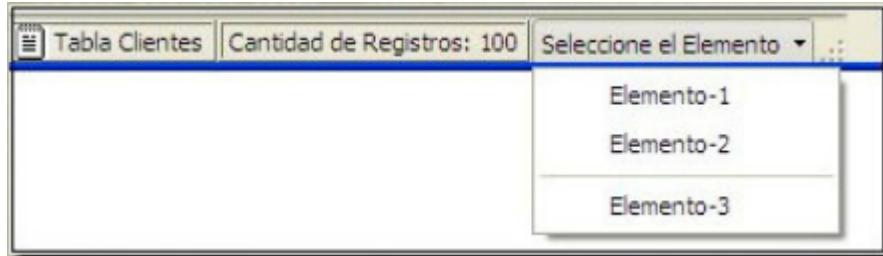
El control StatusStrip también puede contener los controles:

- a. ToolStripDropDownButton,
- b. ToolStripSplitButton y
- c. ToolStripProgressBar.

Control ToolStripDropDownButton

Representa un control en el que, al hacer clic el usuario puede seleccionar un elemento único.

Utilice ToolStripDropDownButton para activar controles desplegables familiares, como los selectores de color. Es similar a un menú contextual de Windows.

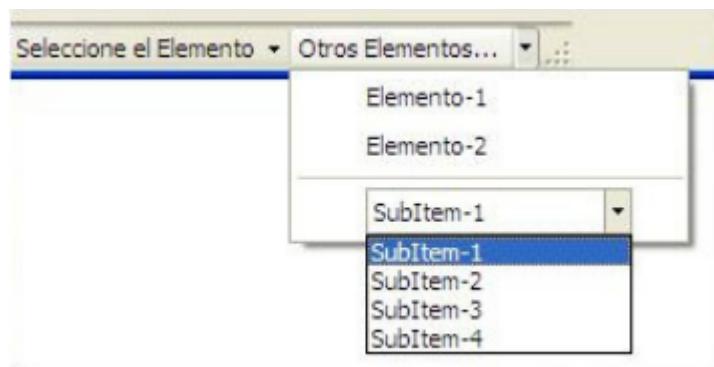


Las propiedades más importantes que podemos mencionar de este control son:

DisplayStyle	muestran el texto y las imágenes en un ToolStripItem.
DropDownItems	obtiene la colección de elementos que puede contener el objeto ToolStripDropDownButton. Utilice el editor de esta colección para agregar/quitar elementos y configurar propiedades. Los elementos que se pueden agregar a un control ToolStripDropDonwButton son: • MenuItem; • ComboBox; • Separador; • TextBox.
Font	determina la fuente del texto que muestra el elemento.
Image	obtiene la imagen que se muestra en un control ToolStripItem.
Text	establece el texto que se mostrará en el elemento.
TextAlign	facilita la alineación del texto.
ToolTipText	permite la aparición del texto, que aparece como ToolTip (pequeña ventana emergente rectangular que muestra una breve descripción de la finalidad de un control cuando el usuario sitúa el puntero del mouse sobre el control).

Control ToolStripSplitButton

Representa una combinación de un botón estándar situado a la izquierda y un botón de lista desplegable situado a la derecha, o la combinación contraria si el valor de RightToLeft es Yes.



Las propiedades más importantes que podemos mencionar de este control son las mismas que hemos mencionado para el control ToolStripDropDownButton.

Control ToolStripProgressBar

Representa un control de barra de progreso de Windows. Indica visualmente el progreso de una operación larga. El control ToolStripProgressBar muestra una barra que se llena de izquierda a derecha con el color de resaltado del sistema a medida que progresá la operación.



Las propiedades **Maximum** y **Minimum** definen el intervalo de valores que representarán el progreso de una tarea. Normalmente, la propiedad **Minimum** se establece en cero, y la propiedad **Maximum** se establece en un valor que indica que la tarea ha terminado. Por ejemplo, para mostrar correctamente el progreso al copiar un grupo de archivos, la propiedad **Maximum** se podría establecer en el número total de archivos que se van a copiar. La propiedad **Value** representa el progreso que la aplicación realiza para terminar la operación.

Para agregar/quitar controles a la barra de estado

Puede agregar/quitar controles a la barra de estado de las mismas tres maneras que los controles anteriores:

- a. Utilizando la ventana de Propiedades:** Seleccione el control StatusStrip o agregue uno al formulario. En la ventana de Propiedades, haga clic en el botón

de puntos suspensivos () situado junto a la propiedad Items. En el Editor de la colección de Items, haga clic en Agregar o Quitar para agregar y quitar controles.

- b. **Utilizando la etiqueta inteligente:** Seleccione el control StatusStrip o agregue uno al formulario. Haga clic en el glifo de la etiqueta inteligente (). En el cuadro de diálogo Tareas de StatusStrip, seleccione Editar Elementos. En el Editor de la colección de Items, haga clic en Agregar o Quitar para agregar y quitar controles.
- c. **Utilizando programación:** El proceso de agregar un control a una barra de estado consiste básicamente en especificar el elemento y en asignarle propiedades. Utilice el método Add de la propiedad Items del control StatusStrip:
 - StatusStrip.Items.Add(Image): agrega ToolStripItem que muestra la imagen especificada a la colección.
 - StatusStrip.Items.Add(String): agrega un ToolStripItem que muestra el texto especificado.
 - StatusStrip.Items.Add(Item): agrega el elemento especificado al final de la colección.
 - StatusStrip.Items.Add(String, Imagen): agrega un ToolStripItem que muestra el texto y la imagen especificados.
 - StatusStrip.Items.Add(String, Imagen, EventHandler): agrega un ToolStripItem que muestra el texto, y la imagen especificados a la colección y provoca el evento Click.

El ejemplo de código siguiente requiere que tenga un formulario con un control StatusStrip agregado:

```
ToolStripProgressBar barra = new ToolStripProgressBar();
sspEstado.Items.Add(barra);
barra.Maximum = 100;
```

También se puede utilizar el método AddRange para agregar un control ToolStripStatusLabel a la barra de estado.

Quitar controles

Para quitar controles mediante programación utilice el método **RemoveAt** o **Clear** de la propiedad **Items**. El método **RemoveAt** quita un solo control; el método **Clear** quita todos los controles de la barra de estado.

Consulte más detalles sobre la clase en el siguiente enlace:

[Información general del control StatusStrip - Windows Forms .NET Framework | Microsoft Learn](#)

Retomando los requerimientos de la situación profesional podemos concluir que:

- Un control TreeView nos permite mostrar en forma jerárquica los datos de las provincias y sus departamentos.
- Un control ListView resuelve de forma eficiente el manejo de la información sobre los tipos de incendio y cantidades referidos al nodo seleccionado en el control TreeView
- Agregar un control StatusSrtip permitirá contar con un componente apto para mostrar los totales de incendios requeridos en cada caso.

SP3/Autoevaluación 4

1. Indique la opción correcta.

Un control StatusStrip puede contener los siguientes controles: ToolStripStatusLabel, DropDownButton, SplitButton y ToolStripProgressBar.

- Verdadero
- Falso

2. Indique la opción correcta

¿Para qué se usa el control StatusStrip?

- Se utiliza en conjunto a un control ToolStripControl.
- Informar al usuario sobre las medidas de una tabla de una base de datos, por ejemplo, el número de registros y la posición actual en la base de datos.
- Informar al usuario sobre las medidas de una tabla de una base de datos.

3. Indique la opción correcta

¿Cuál es la propiedad que permite obtener los elementos contenidos en un Label en el control StatusStrip?

- ShowItemToolTips
- ToolStripStatusLabel
- BorderSides
- Text

4. Indique la opción correcta

¿Qué control contiene un StatusStrip?

- ComboBox
- ListBox
- ProgressBar
- GridView

5. Indique la opción correcta

Para agregar o quitar controles de un StatusBar debemos utilizar, en el modo de diseño, los botones.

- Verdadero
- Falso

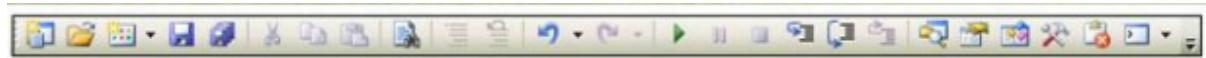
Respuestas correctas¹⁴

¹⁴1) Falso. 2) Informar al usuario sobre las medidas de una tabla de una base de datos, por ejemplo, el número de registros y la posición actual en la base de datos. 3) ToolStripStatusLabel 4) ProgressBar. 5) Falso.

SP3/H5: Control ToolStrip, barras de herramientas

Si bien en la situación profesional planteada no se requiere el uso de una barra de herramientas, conocida como control ToolStrip en Visual Studio, creemos conveniente dedicarle un espacio para tomar contacto con este control que puede ser de mucha utilidad en aplicaciones similares a la planteada acá.

El control ToolStrip permite crear barras de herramientas que se pueden asociar a una aplicación. Por ejemplo, la siguiente imagen:



Normalmente, una barra de herramientas contiene botones que corresponden a elementos de algún menú de la aplicación, lo que proporciona una interfaz gráfica para que el usuario tenga un acceso rápido a las funciones y comandos utilizados con más frecuencia.

Las propiedades más importantes que podemos mencionar del control **ToolStrip** son:

- **BackColor:** obtiene o establece el color de fondo del ToolStrip.
- **Dock:** obtiene o establece que los bordes del ToolStrip se acoplarán a su control principal y determina cómo se cambia el tamaño de un ToolStrip con su elemento primario.
- **Font:** obtiene o establece la fuente utilizada para mostrar texto en el control.
- **Ítems:** obtiene todos los elementos que contiene el ToolStrip. Esta propiedad será tratada en profundidad más adelante.
- **LayoutStyle:** obtiene o establece un valor que indica cómo se disponen los elementos en la barra de herramientas.
- **ShowItemToolTips:** obtiene o establece un valor que indica si se mostrará información sobre herramientas en los elementos de ToolStrip

Elementos de un control ToolStrip

El control ToolStrip puede contener los siguientes controles:

a. Control **ToolStripButton**

Representa un botón de barra de herramientas que admite texto e imagen.



Las propiedades más importantes que podemos mencionar de este control son:

- **DisplayStyle:** obtiene o establece si se muestran el texto y las imágenes en un ToolStripButton.
- **Image:** obtiene o establece la imagen que se muestra en el botón.
- **ImageAlign:** obtiene o establece la posición de la imagen.
- **Text:** obtiene o establece el texto que se mostrará en el botón.
- **TextAlign:** obtiene o establece la posición del texto.
- **ToolTipText:** obtiene o establece el texto que aparece como ToolTip (pequeña ventana emergente rectangular que muestra una breve descripción de la finalidad de un control, cuando el usuario sitúa el puntero del mouse sobre el mismo).

Generalmente se utiliza el evento “Click” de este control para escribir las acciones que se deben ejecutar cuando el usuario presiona un botón de la barra de herramientas. Es similar al comportamiento de los botones de comando ampliamente usados (Control Button).

b. Control **ToolStripComboBox**

Este control muestra un campo de edición combinado con un ListBox y permite al usuario seleccionar una opción de la lista o escribir texto nuevo.

De forma predeterminada, un ToolStripComboBox muestra un campo de edición con una lista desplegable oculta.

La propiedad DropDownStyle determina el estilo que mostrará el cuadro combinado.

c. Control **ToolStripSplitButton**

Representa una combinación de un botón estándar situado a la izquierda y un botón de lista desplegable situado a la derecha, o la combinación contraria si el valor de la propiedad RightToLeft es Yes.

d. Control **ToolStripLabel**

Muestra una etiqueta de texto que suele utilizarse en una barra de estado o en un ToolStrip como un comentario o título.

e. Control **ToolStripSeparator**

Representa una línea utilizada para agrupar elementos de ToolStrip.

f. Control **ToolStripDropDownButton**

Representa un control en el que, al hacer clic, muestra una lista asociada donde el usuario puede seleccionar un elemento único.

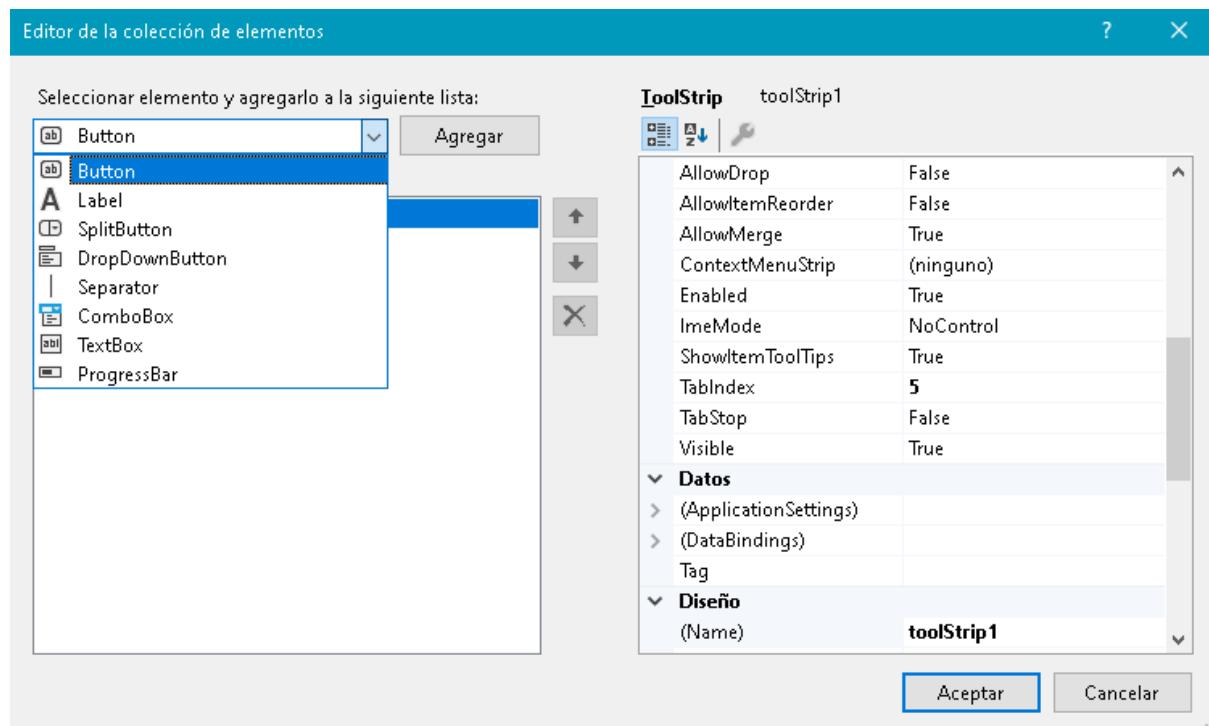
g. Control **ToolStripProgressBar**

Representa un control de barra de progreso de Windows.

h. Control **ToolStripTextBox**

Representa un cuadro de texto en un control ToolStrip que permite al usuario escribir texto.

Podemos editar el contenido del control ToolStrip desde la ventana del editor de elementos:



Podemos ver la lista de opciones disponibles para agregar en la lista desplegable de la izquierda, a la derecha tenemos las propiedades del elemento seleccionado.

Podrá encontrar más información y referencias de este control en este enlace:

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

1. Indique la opción correcta

¿Cómo se quitan controles a la barra de herramientas?

- Remove
- Clear
- RemoveAt
- Delete

2. Indique la opción correcta

El control ToolStripComboBox se utiliza para listas desplegables.

- Verdadero
- Falso

3. Indique la opción correcta

¿Qué controles puede contener un ToolStrip?

- ComboBox
- ToolStripButton
- ProgressBar
- GridView

4. Indique la opción correcta

Para agregar elementos en un control ToolStrip se puede usar
ToolStrip1.Items.Add()

- Verdadero
- Falso

5. Indique la opción correcta

El control ToolStripSeparator se utiliza para agrupar controles.

- Verdadero
- Falso

6. Indique la opción correcta

¿Para qué se usa el control ToolStrip?

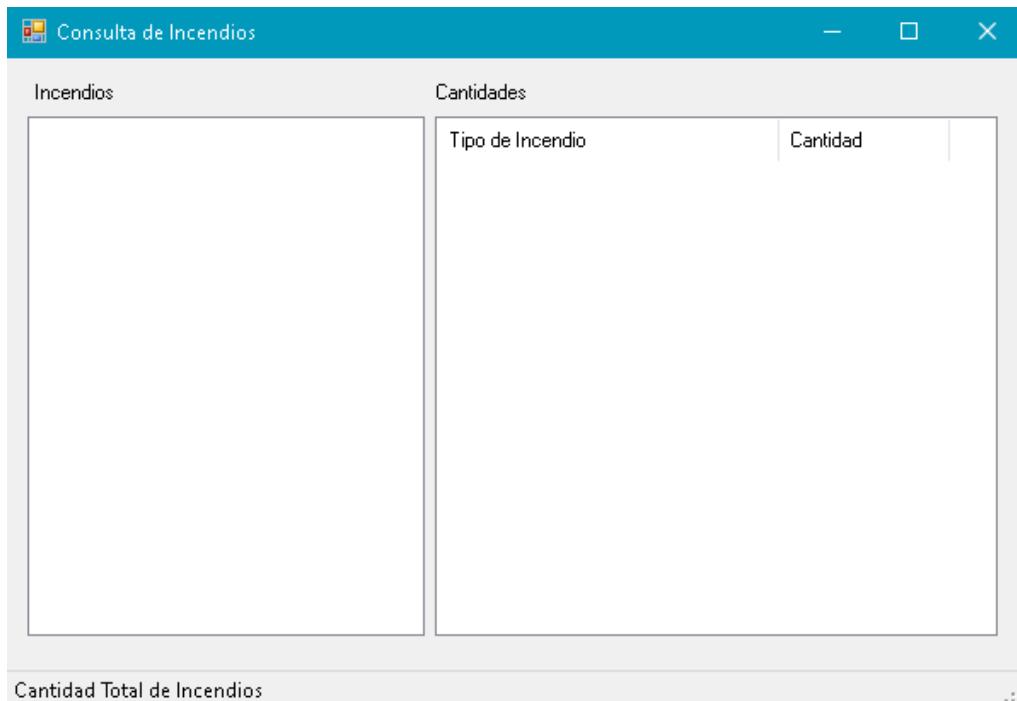
- Se utiliza en conjunto a un control StatusStripControl.
- Informar al usuario sobre las medidas de una tabla de una base de datos, indicando, por ejemplo, el número de registros y la posición actual en la base de datos.
- Acceso rápido a las funciones y comandos utilizados con más frecuencia.

Respuestas correctas¹⁵

¹⁵1) RemoveAt. 2) Verdadero. 3) ToolStripButton. 4) Verdadero. 5) Verdadero. 6) Acceso rápido a las funciones y comandos utilizados con más frecuencia.

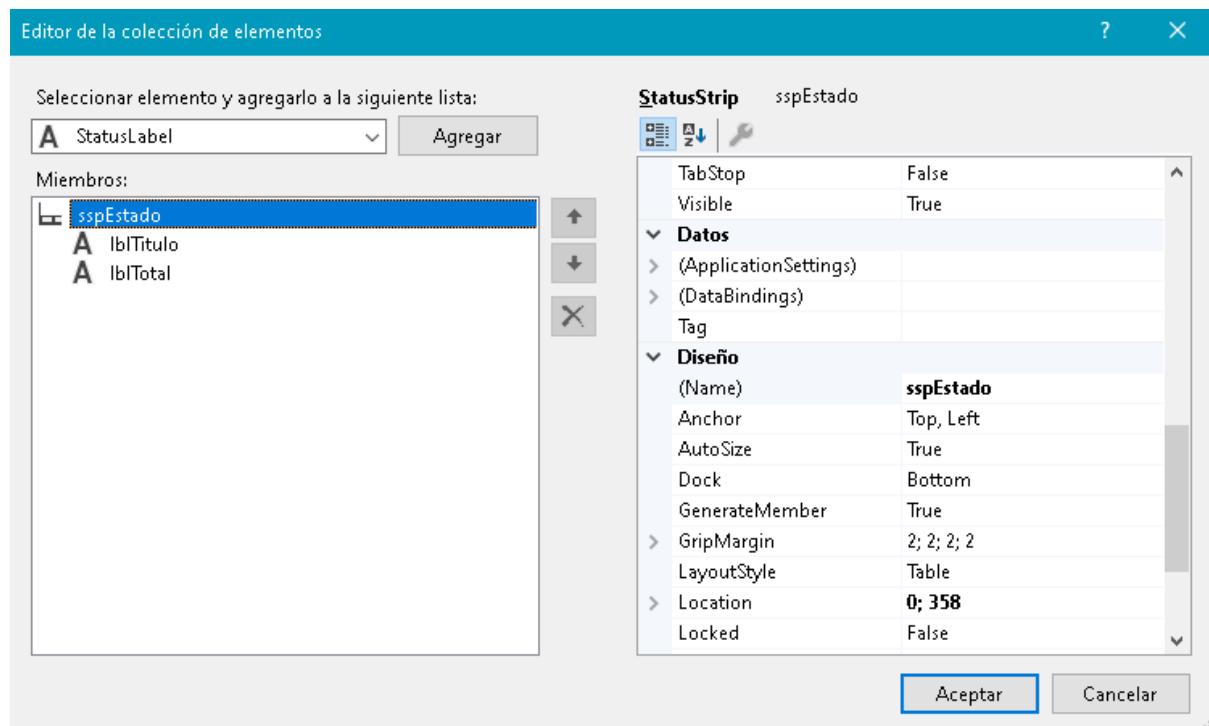
SP3/Ejercicio resuelto

Para esta resolución crearemos un nuevo proyecto de Visual Studio y procederemos a diseñar y configurar todos los componentes del formulario que compone la interfaz gráfica.



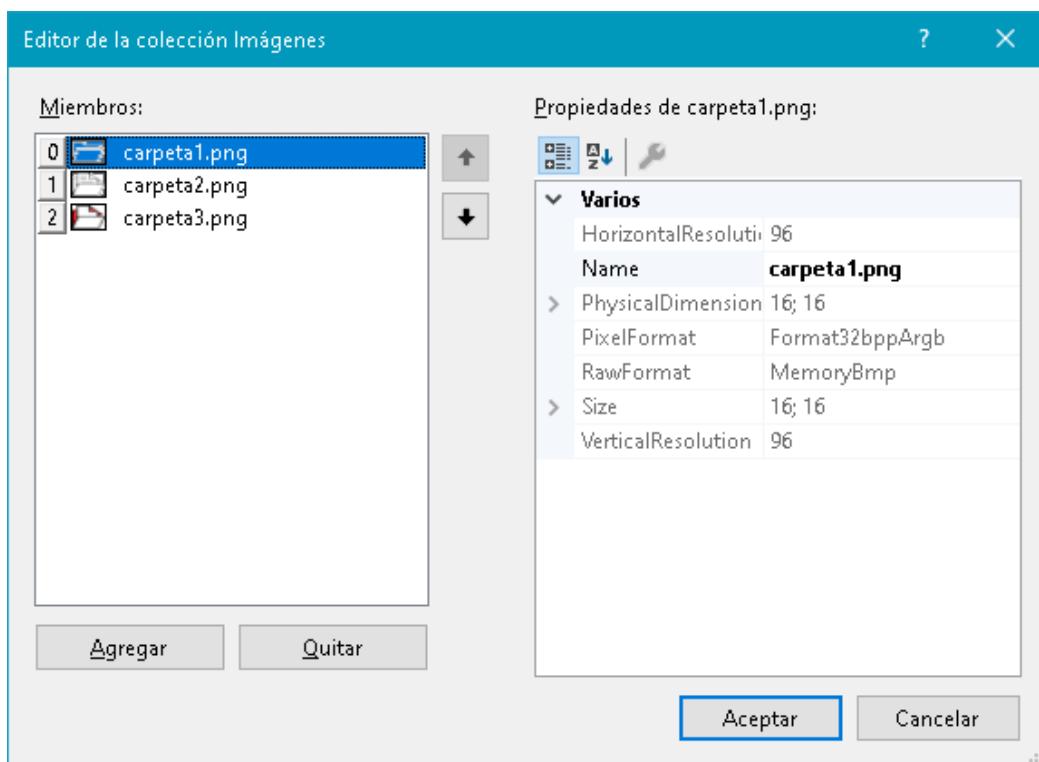
Se necesitan 2 “Labels”, un control “TreeView” (tvwIncendios), un control “ListView” (lvwCantidades) y un control “StatusStrip” (sspEstado).

En el control “StatusStrip” agregamos dos componentes de tipo “StatusLabel”:

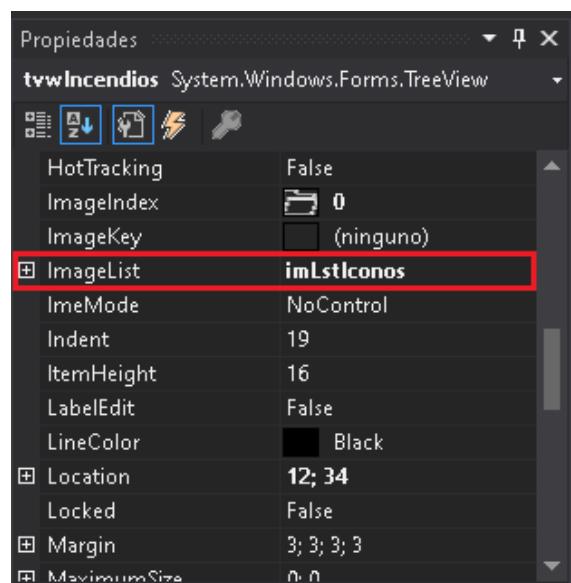


En “lblTitulo” asignaremos a la propiedad “Text” el valor “Cantidad Total de Incendios”.
En “lblTotal”, la propiedad “Text” quedará vacía, su valor será asignado por código dependiendo del nodo que el usuario seleccione en el TreeView.

Para completar la presentación de los datos en el TreeView vamos a agregar al formulario un control “ImageList”, su nombre será “imLstIconos”. Seguidamente configuraremos sus elementos con 3 archivos de tipo “bmp”, “jpg” o “png”, estos archivos de iconos se usarán para asignar a cada tipo de nodo en el treeview.



Una vez cargadas las imágenes en el control ImageList debemos vincular el TreeView para que tome este control como fuente para los iconos de los nodos, esto se logra simplemente asignando el nombre del control ImageList “imLstIconos” a la propiedad “ImageList” del TreeView:



De esta forma queda completo el diseño y configuración del formulario, continuaremos ahora con la implementación de las clases que necesitamos construir para obtener la funcionalidad requerida.

Comenzamos con la clase “CProvincia”, se encargará de manejar los datos de las tablas “Provincias” y “Departamentos”.

Clase “CProvincia”

Esta clase trabajará con las tablas de Provincias y Departamentos y nos permitirá cargar esa información en el control TreeView. Su contenido será este:

```
⊕ CargarTreeView(System.Windows.Forms.TreeView)
⊕ CProvincia()
⊕ Dispose()
⊕ ObtenerProvincia(int)
⊕ DS
⊕ TablaDepartamento
⊕ TablaProvincia
```

El código completo de la clase resulta así:

```
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 SP3
{
    public class CProvincia
    {
        DataSet DS;
        String TablaProvincia = "Provincias";
        String TablaDepartamento = "Departamentos";

        // constructor
        public CProvincia()
        {
            try
            {
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Incendios.mdb";
                cnn.Open();
            }
        }

        public void CargarTreeView(TreeView tv)
        {
            DS = new DataSet();
            OleDbDataAdapter da = new OleDbDataAdapter("SELECT * FROM " + TablaProvincia, cnn);
            da.Fill(DS);
            DataTable dt = DS.Tables[0];
            foreach (DataRow dr in dt.Rows)
            {
                TreeNode tn = new TreeNode(dr["Nombre"].ToString());
                tn.Tag = dr["ID"];
                tv.Nodes.Add(tn);
            }
        }

        public void ObtenerProvincia(int id)
        {
            DataRow dr = DS.Tables[0].Rows[id];
            return dr;
        }

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

```

        DS = new DataSet();
        OleDbCommand cmdP = new OleDbCommand();
        cmdP.Connection = cnn;
        cmdP.CommandType = CommandType.TableDirect;
        cmdP.CommandText = TablaProvincia;
        OleDbDataAdapter DAP = new OleDbDataAdapter(cmdP);
        DAP.Fill(DS, TablaProvincia);
        DataColumn[] pkP = new DataColumn[1];
        pkP[0] = DS.Tables[TablaProvincia].Columns["Provincia"];
        DS.Tables[TablaProvincia].PrimaryKey = pkP;
        //
        OleDbCommand cmdD = new OleDbCommand();
        cmdD.Connection = cnn;
        cmdD.CommandType = CommandType.TableDirect;
        cmdD.CommandText = TablaDepartamento;
        OleDbDataAdapter DAD = new OleDbDataAdapter(cmdD);
        DAD.Fill(DS, TablaDepartamento);
        DataColumn[] pkD = new DataColumn[1];
        pkD[0] = DS.Tables[TablaDepartamento].Columns["Departamento"];
        DS.Tables[TablaDepartamento].PrimaryKey = pkD;
        cnn.Close();
    }
    catch (Exception ex)
    {
        throw new Exception("CProvincia: " + ex.Message);
    }
}

public int ObtenerProvincia(int departamento)
{
    // devuelve el número de provincia de un departamento
    DataRow dr = DS.Tables[TablaDepartamento].Rows.Find(departamento);
    if(dr == null)
    {
        throw new Exception("CProvincia: No existe del departamento");
    }
    return (int)dr["Provincia"];
}

public void CargarTreeView(TreeView tvw)
{
    tvw.Nodes.Clear();
    // agregar la raiz
    TreeNode raiz = tvw.Nodes.Add("raiz", "INCENDIOS", 0, 0);
    try
    {

        // agregar las provincias
        foreach (DataRow drProv in DS.Tables[TablaProvincia].Rows)
        {
            TreeNode prov = raiz.Nodes.Add(drProv["provincia"].ToString(),
drProv["Nombre"].ToString(), 1, 1);
            // agregar los departamentos de la provincia
            foreach (DataRow drDep in DS.Tables[TablaDepartamento].Rows)
            {
                // comparar el numero de provincia
                if ((int)drProv["Provincia"] == (int)drDep["Provincia"])
                {

```

```

        // agregar el departamento como nodo hijo de la
provincia
        prov.Nodes.Add(drDep["Departamento"].ToString(),
drDep["Nombre"].ToString(), 2, 2);
    }
}
}
catch(Exception ex)
{
    throw new Exception("CProvincia: " + ex.Message);
}
}

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

```

En el **constructor** de la clase se realiza la conexión con la base de datos y se cargan en el DataSet los registros de las tablas Provincias y Departamentos, además se asignan las claves primarias de ambas tablas.

El método “**CargarTreeView**” recibe por parámetro el control treeView del formulario, se encarga de agregar el nodo raíz, los nodos con los nombres de las provincias y para cada uno de ellos agrega como hijos los nodos con los nombres de los departamentos.

Observe que en el momento de agregar cada nodo se hace referencia al ícono que se usará para el nodo, los valores usados: 0, 1 y 2 son los índices que ocupan los iconos en el control ImageList que contiene los iconos.

```
// agregar la raiz
TreeNode raiz = tvw.Nodes.Add("raiz", "INCENDIOS", 0, 0);
```

```

// agregar una provincia
TreeNode prov =
raiz.Nodes.Add(drProv["provincia"].ToString(), drProv["Nombre"].ToString(), 1, 1);
    // agregar el departamento como nodo hijo de la provincia
    prov.Nodes.Add(drDep["Departamento"].ToString(), drDep["Nombre"].ToString(),
2, 2);

```

Para la raíz se usa el índice 0, para las provincias el índice 1 y para los departamentos el índice 2.

El método “**ObtenerProvincia**” recibe por parámetro el número de un departamento y valiéndose de la clave primaria de la tabla Departamentos realiza una búsqueda de ese valor.

```
DataRow dr = DS.Tables[TablaDepartamento].Rows.Find(departamento);
```

Del registro localizado se toma el valor del campo “provincia”, ese número de provincia es el valor que devuelve el método.

```
return (int)dr["Provincia"];
```

Esto nos facilita conocer a qué provincia pertenece cada departamento cuando necesitemos procesar las cantidades de incendios.

El método “**Dispose**” libera los recursos del DataSet usado por la clase.

Clase “CIncendio”

La clase “CIncendio” trabajará con las tablas de Incendios y Tipoincendio y sus métodos están relacionados con los procesos que deben cargar el detalle de cantidad de incendios por tipo de incendio en el control ListView del formulario. Su contenido es este:

```
⌚ CIncendio()
⌚ Dispose()
⌚ ObtenerIncendios(System.Windows.Forms.ListView)
⌚ ObtenerIncendiosPorDepartamento(int, System.Windows.Forms.ListView)
⌚ ObtenerIncendiosPorProvincia(int, System.Windows.Forms.ListView)
⌚ DS
⌚ TablaIncendio
⌚ TablaTipoincendio
```

Además de los métodos para el constructor de la clase y Dispose vamos a implementar 3 métodos para cargar el control ListView, los 3 métodos deben mostrar todos los tipos de incendio que existen, y para cada tipo de incendio la cantidad total de incendios

registrado, la diferencia entre estos 3 métodos será el nivel de detalle de los totales obtenidos, para el nodo raíz se mostrarán los totales generales, para un nodo de provincia los totales serán con las cantidades registradas solamente para esa provincia y para un nodo de departamento las cantidades serán las que registre sólo ese departamento.

En todos los casos necesitamos recorrer primero la tabla de TipoDepartamento y por cada tipo luego recorrer la tabla de Incendios.

El contenido completo de la clase “CIncendio” es este:

```
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 SP3
{
    public class CIncendio
    {
        DataSet DS;
        String TablaIncendio = "Incendios";
        String TablaTipoIncendio = "TipoIncendio";

        // constructor
        public CIncendio()
        {
            try
            {
                OleDbConnection cnn = new OleDbConnection();
                cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=Incendios.mdb";
                cnn.Open();
                DS = new DataSet();
                OleDbCommand cmdI = new OleDbCommand();
                cmdI.Connection = cnn;
                cmdI.CommandType = CommandType.TableDirect;
                cmdI.CommandText = TablaIncendio;
                OleDbDataAdapter DAI = new OleDbDataAdapter(cmdI);
                DAI.Fill(DS, TablaIncendio);
                DataColumn[] pkI = new DataColumn[2];
                pkI[0] = DS.Tables[TablaIncendio].Columns["Departamento"];
                pkI[1] = DS.Tables[TablaIncendio].Columns["TipoIncendio"];
                DS.Tables[TablaIncendio].PrimaryKey = pkI;
                //
                OleDbCommand cmdTI = new OleDbCommand();
                cmdTI.Connection = cnn;
                cmdTI.CommandType = CommandType.TableDirect;
```

```

        cmdTI.CommandText = TablaTipoIncendio;
        OleDbDataAdapter DATI = new OleDbDataAdapter(cmdTI);
        DATI.Fill(DS, TablaTipoIncendio);
        DataColumn[] pkTI = new DataColumn[1];
        pkTI[0] = DS.Tables[TablaTipoIncendio].Columns["TipoIncendio"];
        DS.Tables[TablaTipoIncendio].PrimaryKey = pkTI;
        cnn.Close();
    }
    catch (Exception ex)
    {
        String MsgErr = "CIncendio: " + ex.Message;
        throw new Exception(MsgErr);
    }
}

public int ObtenerIncendiosPorDepartamento(int departamento, ListView lvw)
{
    int Total = 0; // cantidad total de incendios del departamento
    int Cantidad = 0; // cantidad de incendios por tipo

    lvw.Items.Clear();
    // recorrer los tipos de Incendio
    foreach (DataRow dr in DS.Tables[TablaTipoIncendio].Rows)
    {
        Cantidad = 0;
        // texto del item (primera columna de ListView)
        ListViewItem item = lvw.Items.Add(dr["Descripcion"].ToString());
        // recorrer la tabla de incendios
        foreach(DataRow drI in DS.Tables[TablaIncendio].Rows)
        {
            if( departamento == (int)drI["Departamento"] &&
                (int)dr["TipoIncendio"] == (int)drI["TipoIncendio"])
            {
                Cantidad += (int)drI["Cantidad"];
            }
        }

        // texto del subitem (segunda columna)
        item.SubItems.Add(Cantidad.ToString());
        // acumular el total
        Total += Cantidad;
    }

    return Total;
}

public int ObtenerIncendiosPorProvincia(int provincia, ListView lvw)
{
    int Total = 0; // cantidad total de incendios de la provincia
    int Cantidad = 0; // cantidad de incendios por tipo
    int prov_dep;
    CProvincia prov = new CProvincia();

    lvw.Items.Clear();
    // recorrer los tipos de Incendio
    foreach (DataRow dr in DS.Tables[TablaTipoIncendio].Rows)
    {
        Cantidad = 0;

```

```

        // texto del item (primera columna de ListView)
        ListViewItem item = lvw.Items.Add(dr["Descripcion"].ToString());
        // recorrer la tabla de incendios
        foreach (DataRow drI in DS.Tables[TablaIncendio].Rows)
        {
            prov_dep = prov.ObtenerProvincia((int)drI["Departamento"]);

            if (provincia == prov_dep &&
                (int)dr["TipoIncendio"] == (int)drI["TipoIncendio"])
            {
                Cantidad += (int)drI["Cantidad"];
            }
        }

        // texto del subitem (segunda columna)
        item.SubItems.Add(Cantidad.ToString());
        // acumular el total
        Total += Cantidad;
    }
    prov.Dispose();
    return Total;
}

public int ObtenerIncendios(ListView lvw)
{
    int Total = 0; // cantidad total de incendios
    int Cantidad = 0; // cantidad de incendios por tipo

    lvw.Items.Clear();
    // recorrer los tipos de Incendio
    foreach (DataRow dr in DS.Tables[TablaTipoIncendio].Rows)
    {
        Cantidad = 0;
        // texto del item (primera columna de ListView)
        ListViewItem item = lvw.Items.Add(dr["Descripcion"].ToString());
        // recorrer la tabla de incendios
        foreach (DataRow drI in DS.Tables[TablaIncendio].Rows)
        {
            if ((int)dr["TipoIncendio"] == (int)drI["TipoIncendio"])
            {
                Cantidad += (int)drI["Cantidad"];
            }
        }

        // texto del subitem (segunda columna)
        item.SubItems.Add(Cantidad.ToString());
        // acumular el total
        Total += Cantidad;
    }

    return Total;
}

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

```

En el **constructor** de la clase se realiza la conexión con la base de datos y se cargan en el DataSet los registros de las tablas Incendios y TipolIncendio, además se asignan las claves primarias de ambas tablas.

El método “**ObtenerIncendiosPorDepartamento**” recibe dos parámetros, el primero es el número de departamento consultado y el segundo parámetro es el control ListView que se debe cargar. Como se mencionó anteriormente, el proceso debe recorrer primero la tabla “TipolIncendio” ya que en el ListView hay que agregar una fila para cada tipo de incendio, luego se deberá recorrer la tabla de Incendios y buscar los registros que sean del tipo adecuado y que además pertenezcan a departamento consultado (hay una doble condición que debe cumplirse):

```
if( departamento == (int)drI["Departamento"] &&
    (int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])
{
    Cantidad += (int)drI["Cantidad"];
}
```

Para cada tipo de incendio se obtiene una cantidad que se muestra en la segunda columna del control ListView, además esas cantidades se acumulan en un total:

```
// acumular el total
Total += Cantidad;
```

El método finaliza devolviendo el total obtenido para que pueda ser mostrado en el control StatusStrip del formulario.

El método “**ObtenerIncendiosPorProvincia**” es similar al anterior en cuanto al proceso de las tablas, pero acá se recibe por parámetro el número de una provincia, entonces los totales de incendios para cada tipo de incendio serán la suma de las cantidades de todos los departamentos que pertenecen a esa provincia.

Como el contenido de la tabla de Incendios está dado por departamento vamos a necesitar consultar a qué provincia pertenece cada departamento y así poder decidir si

se suma o no su cantidad. Usaremos entonces un objeto de la clase “CProvincia” y el método “ObtenerProvincia” para resolver ese punto.

```
prov_dep = prov.ObtenerProvincia((int)drI["Departamento"]);  
  
if (provincia == prov_dep &&  
    (int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])  
{  
    Cantidad += (int)drI["Cantidad"];  
}
```

Como podemos ver acá también tenemos una doble condición que se debe cumplir, que el registro del incendio sea de la provincia consultada y que además sea del mismo tipo de incendio que se está procesando en ese momento.

El método “**ObtenerIncendios**” se usará para obtener las cantidades totales de incendios sin distinguir por provincia o departamento, por lo que resulta un poco más sencillo.

La condición usada para sumar las cantidades en este caso es simple, el registro sólo debe pertenecer al tipo de incendio que se está procesando:

```
if ((int)drI["TipoIncendio"] == (int)drI["TipoIncendio"])  
{  
    Cantidad += (int)drI["Cantidad"];  
}
```

Finalmente tenemos el método “**Dispose**” que como en la clase anterior se encarga de liberar los recursos usados por el objeto DataSet.

De esta forma queda completada la implementación de las dos clases auxiliares para resolver la situación propuesta, nos resta desarrollar el código del formulario que en este caso será bastante simple ya que prácticamente se limitará a crear los objetos y ejecutar sus métodos.

“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;

namespace SP3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            CProvincia provincia = new CProvincia();
            provincia.CargarTreeView(tvwIncendios);
            provincia.Dispose();
        }

        private void tvwIncendios_AfterSelect(object sender, TreeViewEventArgs e)
        {
            CIencendio incendio = new CIencendio();
            int Total;
            // determinar el nivel del nodo seleccionado
            TreeNode nodo = e.Node;
            switch (nodo.Level)
            {
                case 0: // es el nodo raiz
                    Total = incendio.ObtenerIncendios(lvwCantidades);
                    sspEstado.Items["lblTotal"].Text = Total.ToString();
                    break;

                case 1: // es un nodo de Provincia
                    Total =
incendio.ObtenerIncendiosPorProvincia(int.Parse(nodo.Name), lvwCantidades);
                    sspEstado.Items["lblTotal"].Text = "Provincia " + nodo.Text +
": " + Total.ToString();
                    break;

                case 2: // es un nodo de Departamento
                    Total =
incendio.ObtenerIncendiosPorDepartamento(int.Parse(nodo.Name), lvwCantidades);
                    sspEstado.Items["lblTotal"].Text = "Departamento " + nodo.Text +
": " + Total.ToString();
                    break;
            }
            // liberar los recursos
            incendio.Dispose();
        }
    }
}

```

En el evento “**Load**” del formulario vamos a usar un objeto de la clase “CProvincia” para ejecutar el método que cargará el control TreeView con los nodos de las provincias y departamentos.

```
CProvincia provincia = new CProvincia();
provincia.CargarTreeView(tvwIncendios);
provincia.Dispose();
```

En el evento “**AfterSelect**” del TreeView se toma una referencia al nodo que generó el evento, y con una estructura “switch” se evalúa el valor de la propiedad “Level” (nivel), de ahí surgen 3 posibles valores: 0 si es el nodo raíz, 1 si es un nodo de provincia y 2 si es un nodo de departamento.

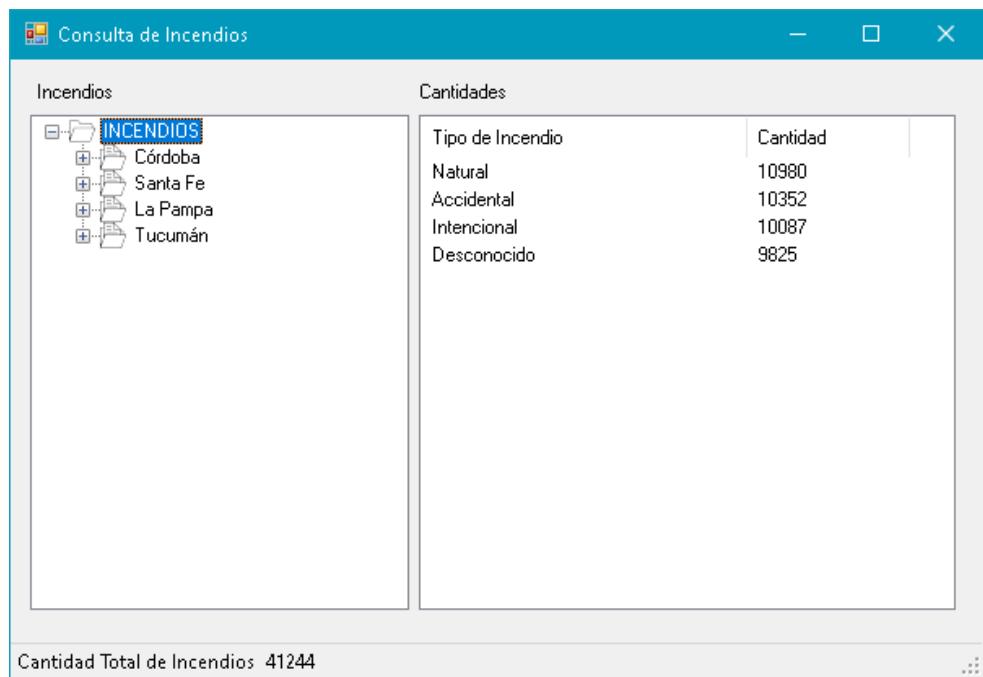
```
switch (nodo.Level)
{
    case 0: // es el nodo raiz
        Total = incendio.ObtenerIncendios(lvwCantidades);
        sspEstado.Items["lblTotal"].Text = Total.ToString();
        break;

    case 1: // es un nodo de Provincia
        Total =
            incendio.ObtenerIncendiosPorProvincia(int.Parse(nodo.Name),
lvwCantidades);
        sspEstado.Items["lblTotal"].Text = "Provincia " + nodo.Text + ": " +
Total.ToString();
        break;

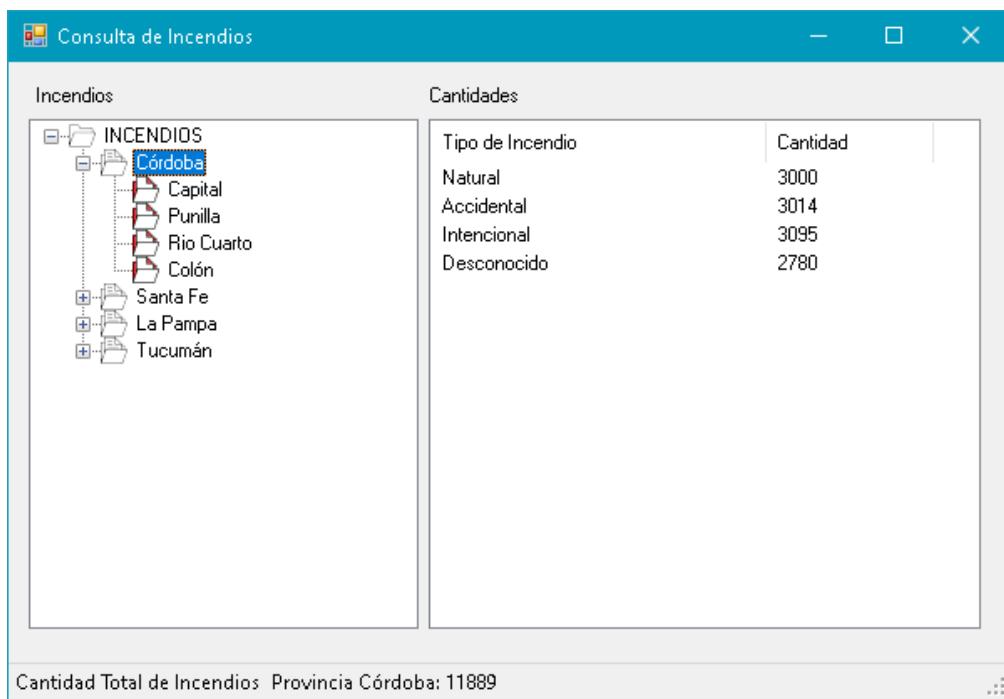
    case 2: // es un nodo de Departamento
        Total =
            incendio.ObtenerIncendiosPorDepartamento(int.Parse(nodo.Name),
lvwCantidades);
        sspEstado.Items["lblTotal"].Text = "Departamento " + nodo.Text + ": " +
Total.ToString();
        break;
}
```

De esa forma se determina qué método de la clase “CIncendio” será invocado para cargar el control ListView con el detalle de las cantidades correspondiente a cada caso. El valor del total de incendios que devuelven esos métodos se asigna al label del control StatusStrip junto con el nombre de la provincia o departamento según el caso.

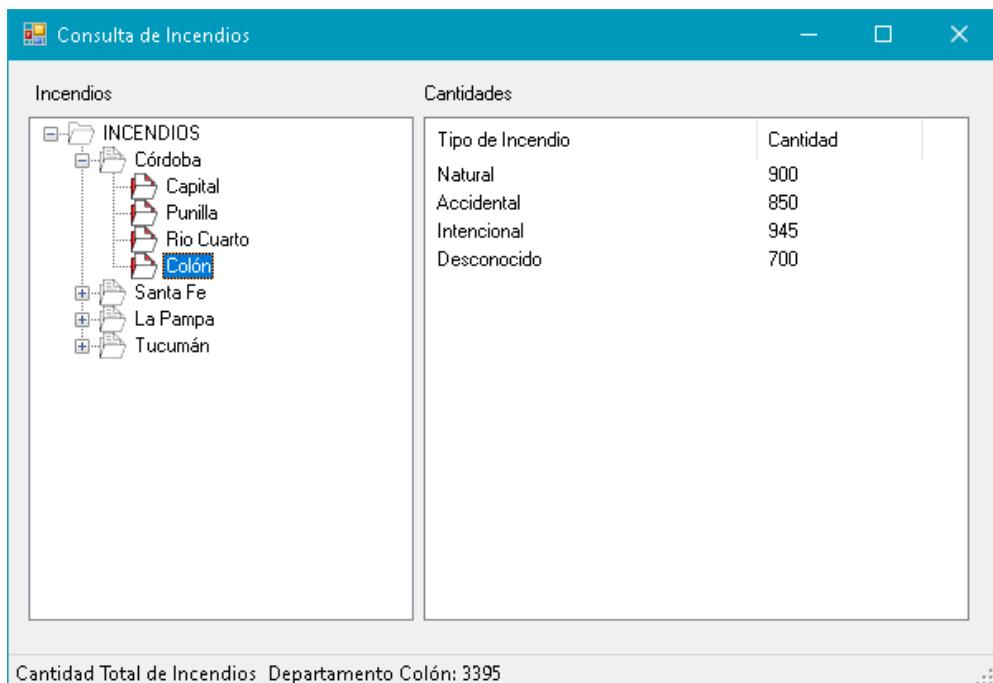
A continuación, vemos el resultado de ejecutar la aplicación y seleccionar el nodo raíz:



Si seleccionamos el nodo de una provincia:



Y si seleccionamos el nodo de un departamento:



Como se puede verificar, en cada caso el ListView y el StatusStrip se actualizan de acuerdo a lo solicitado.

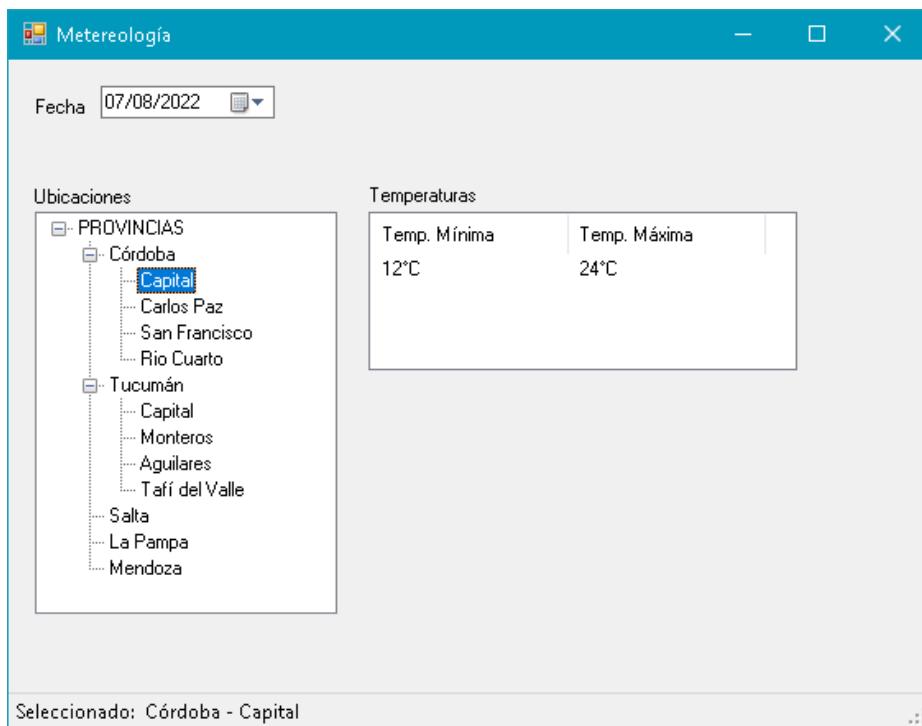
SP3/Ejercicio por resolver

El administrador de una agencia dependiente del Servicio de Meteorología, le ha solicitado a usted que, en calidad de pasante de la agencia, desarrolle una aplicación para consultar las temperaturas mínimas y máximas registradas en las localidades de las provincias de Argentina en todos los días del año.

Esta información será utilizada para conocer la temperatura mínima y máxima en una fecha dada de una localidad determinada o de todas las localidades de una provincia en particular.

Para la consulta de la información, primero seleccionar una fecha determinada de un control “DateTimePicker” luego, en un control de tipo TreeView se podrá seleccionar un nodo con el nombre de una provincia o un nodo con el nombre de una localidad. El control TreeView estará cargado inicialmente con todas las provincias y para cada provincia tendrá todas las localidades que le pertenecen.

Cada vez que el usuario seleccione un nodo en el TreeView la aplicación deberá mostrar en un control ListView las temperaturas mínimas y máximas de la provincia o de la localidad en la fecha seleccionada. Las columnas del ListView son: la temperatura mínima y la temperatura máxima.



En un control StatusStrip se mostrarán los nombres de la provincia y localidad seleccionados.

La base de datos ya contiene almacenados los datos de algunas las provincias y sus principales localidades de Argentina como así también las temperaturas en cada localidad en distintas fechas.

La tabla provincias almacena un número para identificar a la provincia y el nombre de la provincia. La clave principal es el número de provincia.

La tabla localidades almacena un número para identificar a la localidad, el nombre de la localidad y el número de provincia a la que pertenece la localidad. La clave principal es el número de localidad.

La tabla temperaturas almacena un número para identificar a la localidad, la fecha, la temperatura mínima y la temperatura máxima. La clave principal está compuesta por las columnas número de localidad y fecha.

SP3/Evaluación de paso

1. Indique la opción correcta

Para cambiar la vista, en un ListView, se puede usar la propiedad View.

- Verdadero
- Falso

2. Indique la opción correcta

¿Cuál es el objeto que nos permite ver una barra de progreso?

- ToolStripProgressBar
- ToolStripDropDownButton
- ToolStripProgress
- ToolStripButton

3. Indique la opción correcta

La propiedad Nodes del control TreeView contiene la lista de nodos del nivel superior de la vista de árbol.

- Verdadero
- Falso

4. Indique la opción correcta

Para mostrar íconos en tamaño grande dentro de un ListView debemos utilizar la propiedad:

- BigIcon
- LargeIcon
- IconLarge
- Details

5. Indique la opción correcta

Un treeview sirve para crear un árbol que muestre, al menos, dos niveles de una base de datos.

- Verdadero
- Falso

6. Indique la opción correcta

El método LastNode obtiene el último nodo secundario en la colección de nodos que se almacena en la propiedad Nodes del actual nodo.

- Verdadero
- Falso

7. Indique la opción correcta

Un treeview puede contener muchos nodos raíz.

- Verdadero
- Falso

8. Indique la opción correcta

En un ToolStrip la propiedad DisplayStyle muestra el texto y las imágenes en un ToolStripButton.

- Verdadero
- Falso

Respuestas correctas¹⁶

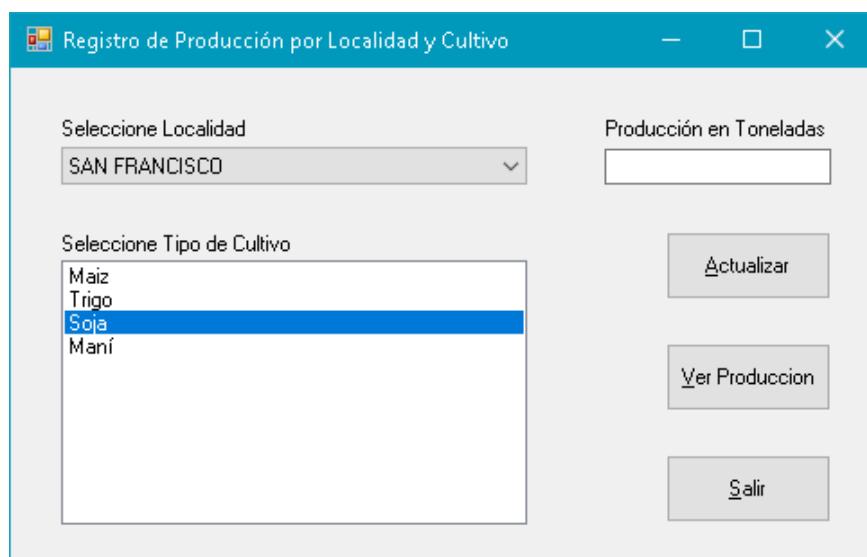
¹⁶1) Verdadero. 2) ToolStripProgressBar 3) Verdadero. 4) LargeIcon. 5) Verdadero. 6) Verdadero. 7) Verdadero.. 8) Falso.

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.

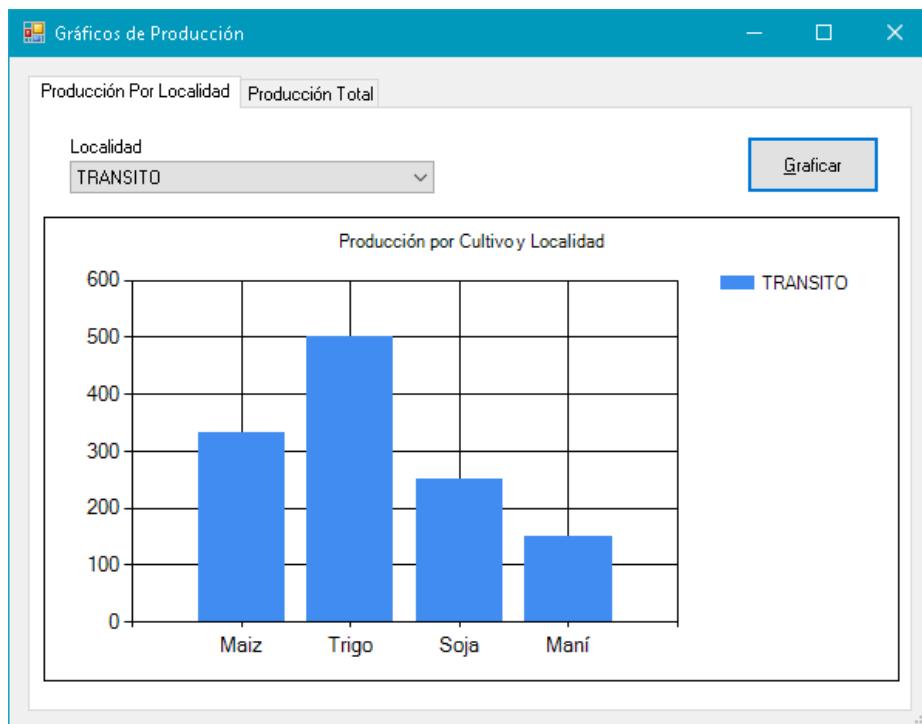
Para registrar la producción en toneladas de una localidad y de un tipo de cultivo, el usuario debe poder seleccionar el nombre de una localidad de un cuadro combinado, seleccionar el nombre del cultivo de un cuadro combinado, ingresar las toneladas en una caja de texto y, al pulsar un botón de comando, 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 ese caso el usuario podrá ingresar un valor y con el botón Actualizar se grabará un nuevo registro.

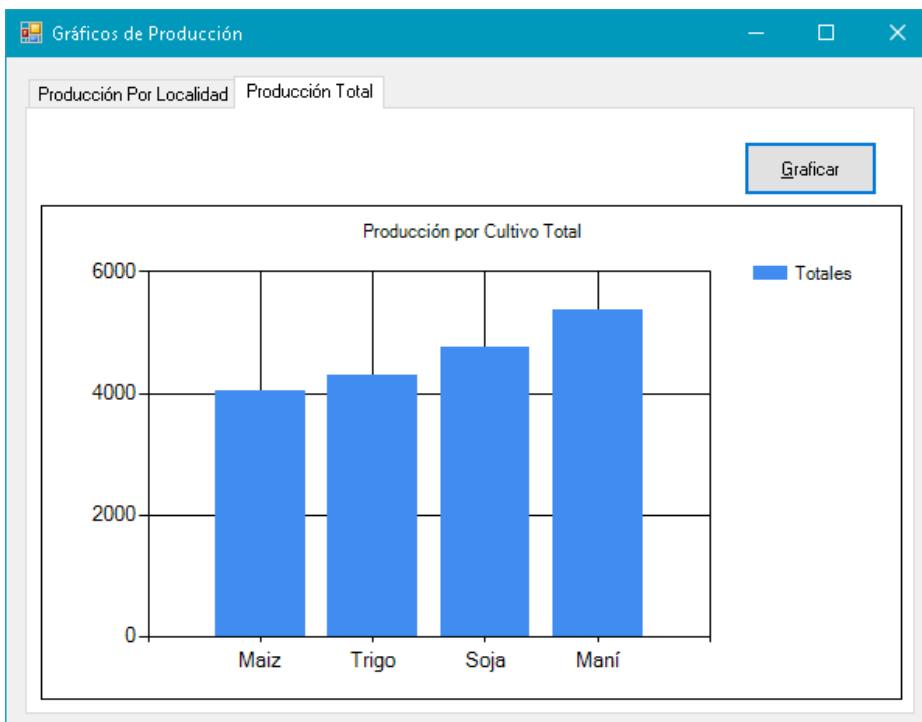


La interfaz cuenta con un botón de comando que, al ser pulsado, abre otro formulario que presentará un control TabControl para mostrar 2 gráficos estadísticos, su primera solapa del control Tab se denomina “Producción Por Localidad”, y mostrará en un control Chart la producción en toneladas de cada localidad 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.

Producción		
	Nombre del campo	Tipo de datos
localidad	Número	
Cultivo	Número	
Producción	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.

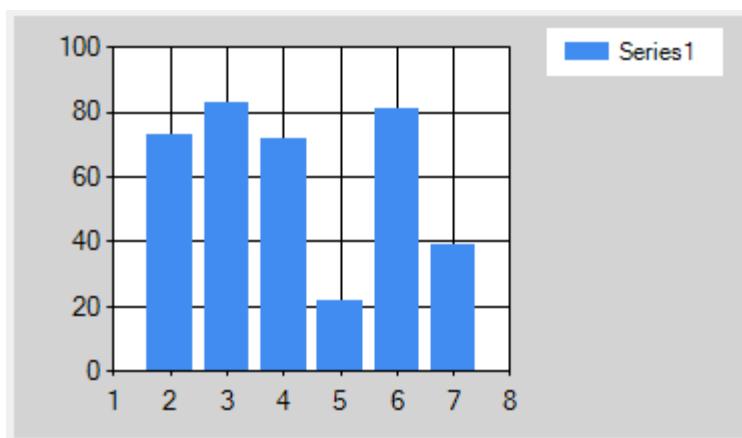
SP4/H1: Control MS Chart

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



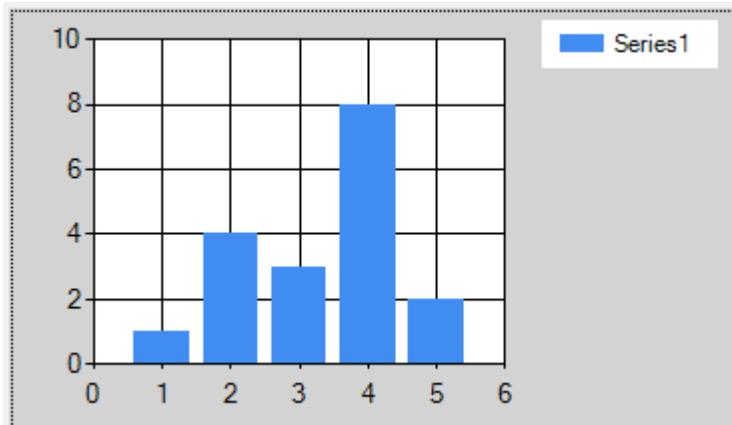
Trazar datos con arreglos y la propiedad Series

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 compuesta 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.

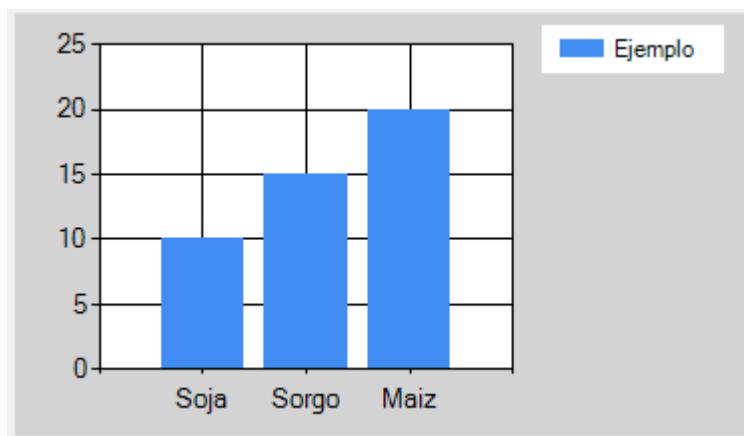
Normalmente, las series y el área del gráfico se especifican de forma declarativa y los puntos de datos se llenan mediante programación, a partir de una consulta a una base de datos o alguna otra base de datos dinámica. De esa forma el formato general del gráfico queda fijo y el contenido del mismo se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

Nombre del campo	Tipo de datos	
Produccion	Número	Cantidad de Toneladas
Cultivo	Texto corto	Nombre del cultivo

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:

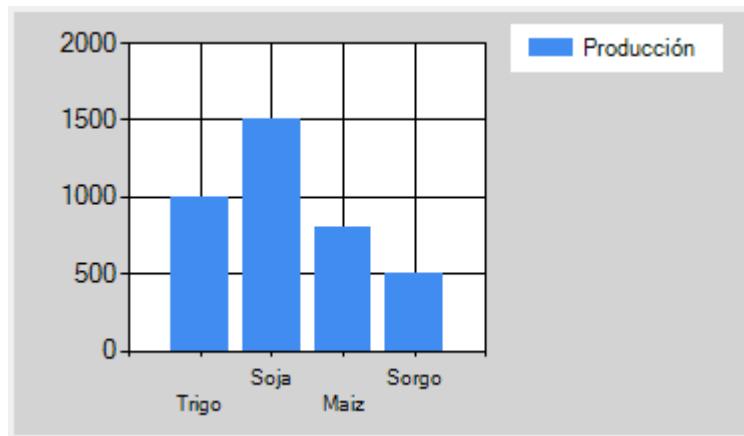
Produccion	Cultivo
1000	Trigo
1500	Soja
800	Maiz
500	Sorgo

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. Esta 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 >= 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:



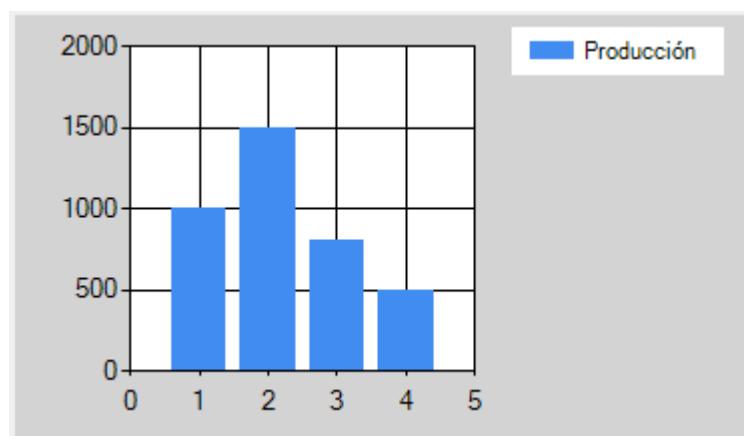
El método “.DataBindTable” recibe como primer parámetro un objeto de tipo “DataView”, en este caso la propiedad “DataView” de la tabla que contiene los datos a graficar.

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.windows.forms.datavisualization.charting.chart?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

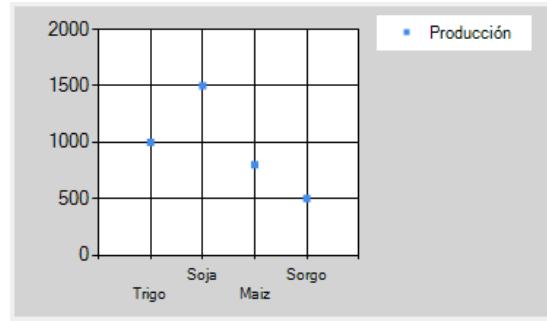
Los tipos de gráficos son:

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.

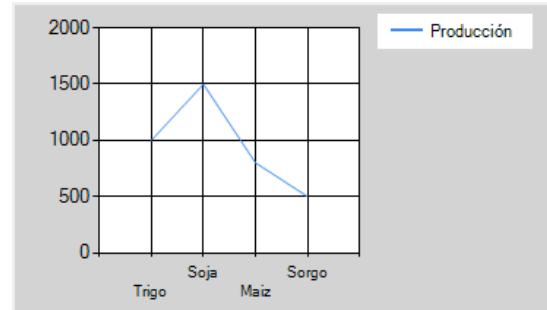
StackedArea100	Tipo de gráfico de áreas 100% apiladas.
Pie	Tipo de gráfico circular
Doughnut	Tipo de gráfico de anillos.
Stock	Tipo de gráfico de cotizaciones.
Candlestick	Tipo de gráfico de vela japonesa.
Range	Tipo de gráfico de intervalos.
SplineRange	Tipo de gráfico de intervalos de spline.
RangeBar	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”:

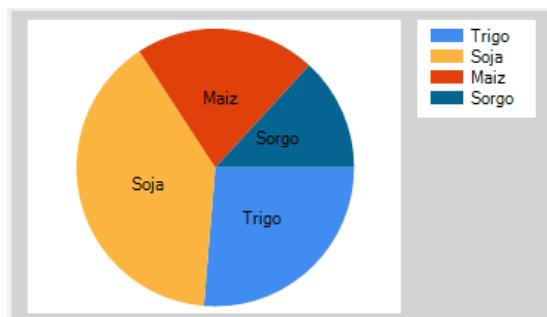
```
chart1.Series[0].ChartType = SeriesChartType.Point;
```



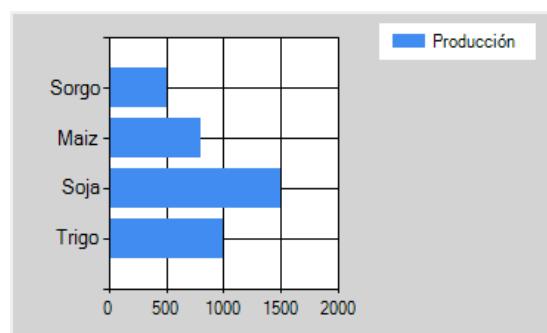
```
chart1.Series[0].ChartType = SeriesChartType.Line;
```



```
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

SP4/Autoevaluación 1

1. Indique la opción correcta

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

- Verdadero
- Falso

2. Indique la opción correcta

Solo se pueden crear gráficos de una sola serie.

- Verdadero
- Falso

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.
- 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.
- 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.

6. Indique la opción correcta

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

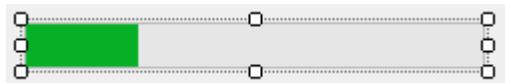
- Agregar un punto de datos al gráfico.
- Agregar un arreglo de puntos en un solo paso.
- Agregar una serie.
- Ninguna de las anteriores.

Respuestas correctas¹⁷

¹⁷1) Verdadero. 2) Falso. 3) Cargar la tabla en el DataSet 4) Repetitiva para cargar datos. 5) Limpia Chart y agrega serie 6) Agregar un punto de datos al gráfico.

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 gráfico 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 PereformStep 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:

[ProgressBar Control - Windows Forms .NET Framework | Microsoft Learn](#)

SP4/Autoevaluación 2

1. Indique la opción correcta

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

- Verdadero
- Falso

2. Indique la opción correcta

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

- Verdadero
- Falso

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

4. Indique la opción correcta

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

- Verdadero
- Falso

5. Indique las opciones

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

- Minimum
- MaxVal
- Maximum
- Step
- StepPerform

- Value
- Valor
- PerformStep

6. Indique la opción correcta

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

- Verdadero
- Falso

7. Indique la opción correcta

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

- Progreso de una operación
- Copiar Archivos
- Mostrar un gráfico al usuario

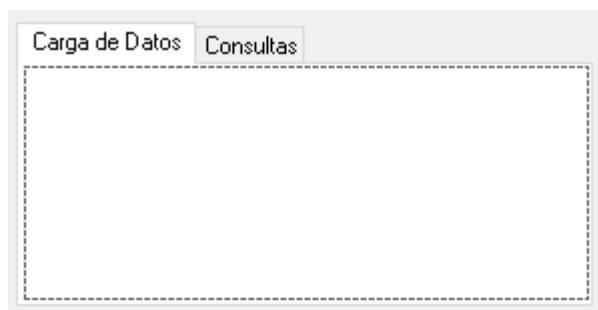
Respuestas correctas¹⁸

¹⁸1) Falso. 2) Falso. 3) Falso. 4) Falso. 5) Minimum, Maximum, Step, Value. 6) Verdadero. 7) Progreso de una operación

SP4/H3: Control TabControl

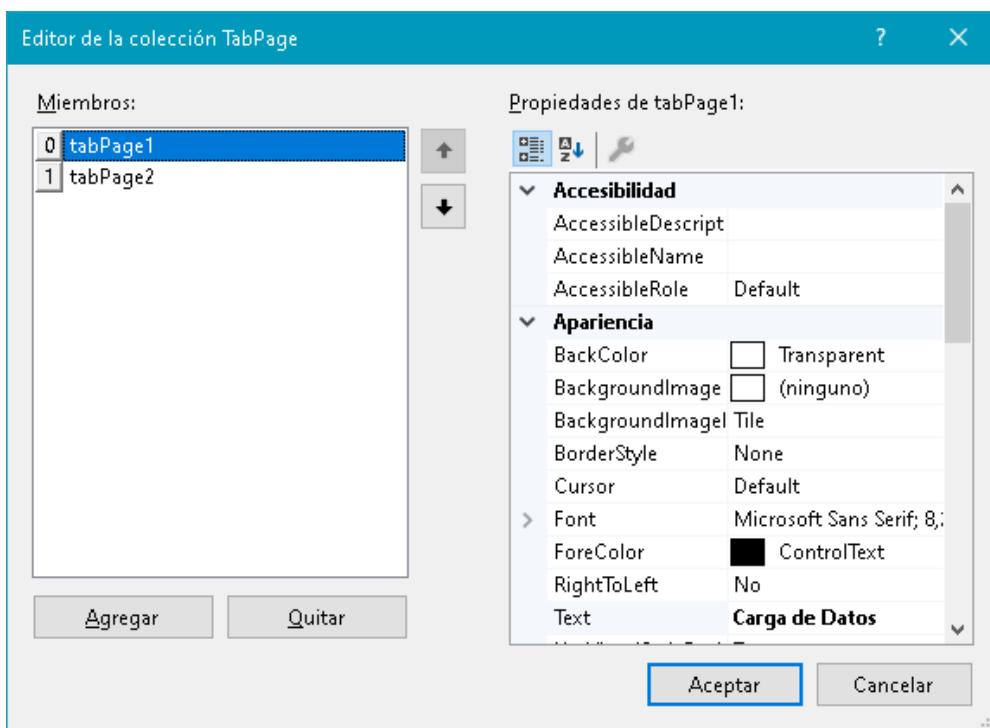
El control TabControl de formularios Windows Forms muestra múltiples fichas, similares a los divisores de un cuaderno o a las etiquetas de las carpetas de un archivador.

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 = newTabPage();
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**.

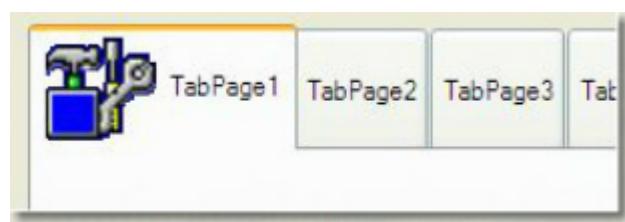
Cambiar la Apariencia de un Control TabControl

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 ícono 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.

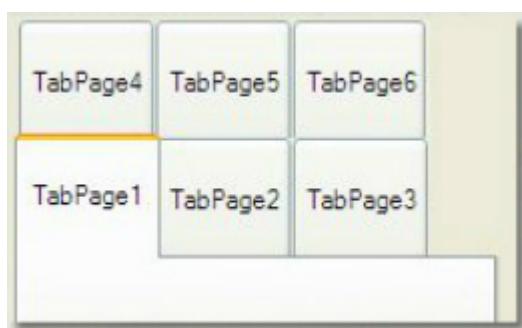
Resultado



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 clic 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>

SP4/Autoevaluación 3

1. Indique la opción correcta

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

- Verdadero
- 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

3. Indique la opción correcta

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

- Verdadero
- Falso

4. Indique la opción correcta

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

- Verdadero
- Falso

5. Indique la opción correcta

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

- Verdadero
- Falso

6. Indique la opción correcta

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

- Verdadero
- Falso

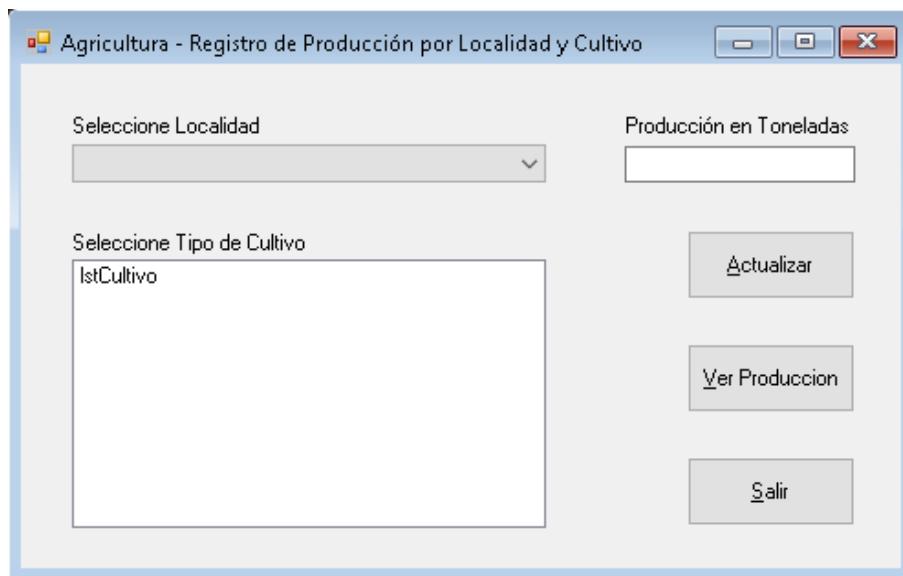
Respuestas correctas¹⁹

¹⁹1) Verdadero. 2) Contener controles e información 3) Verdadero. 4) Falso. 5) Verdadero. 6) Verdadero.

SP4/Ejercicio resuelto

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

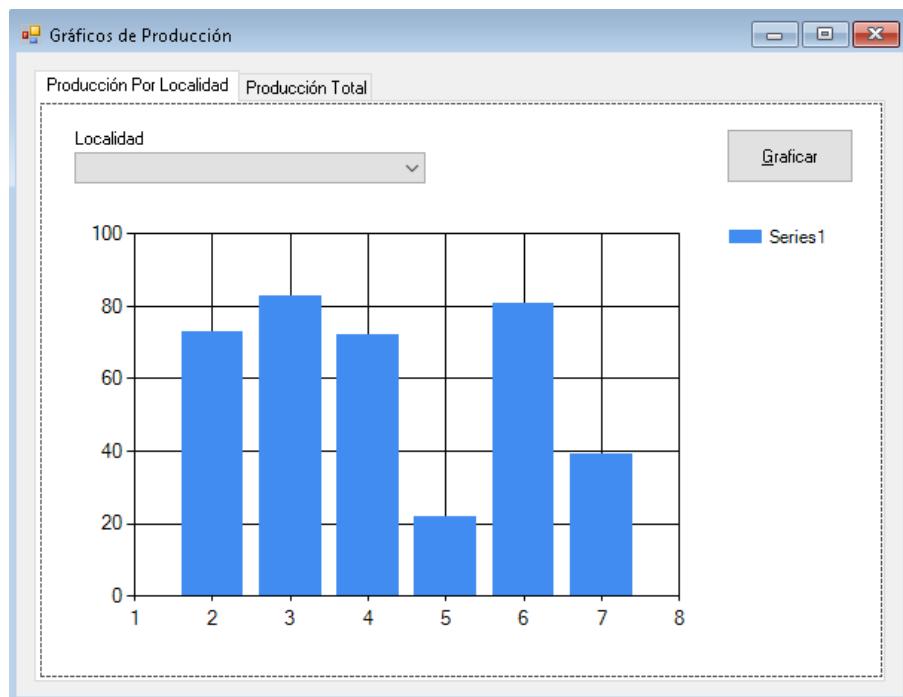
Form1:



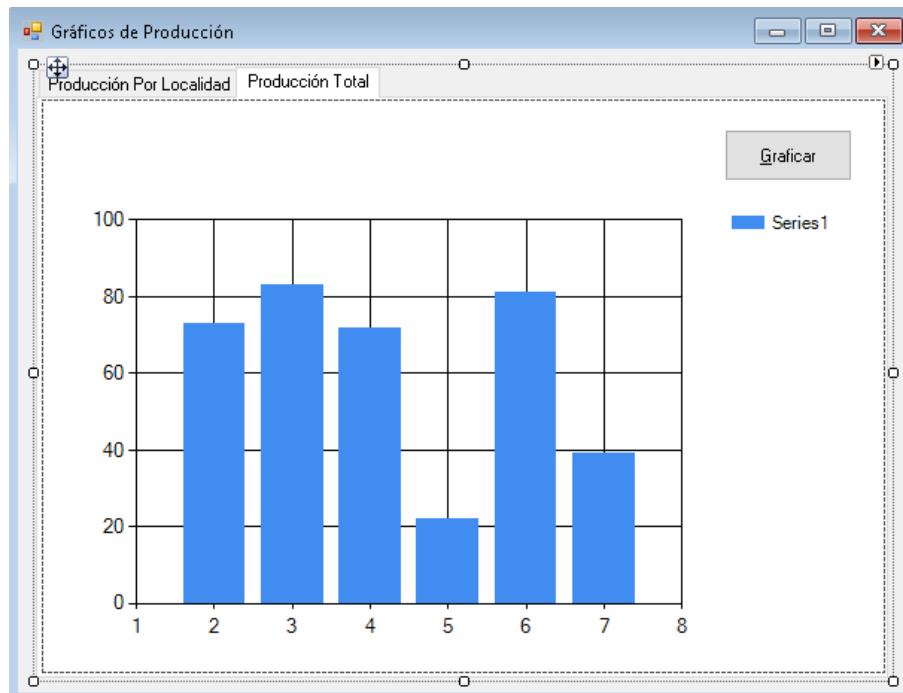
Contiene además de las etiquetas, un control ComboBox (cmbLocalidad), un ListBox (lstCultivo), un TextBox (txtProducción), y 3 controles de tipo Button (btnActualizar, btnVerProducción y btnSalir).

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)



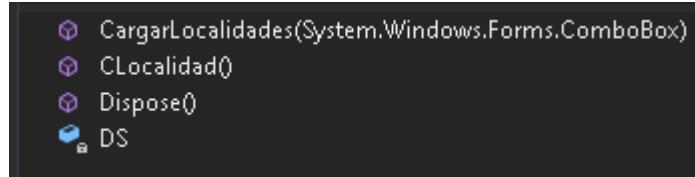
En la segunda pestaña colocamos un control Button (btnGraficarTotal) y un control Chart (chtGraficoTotal)



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.



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 posee 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 se 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()
⌚ GraficarPorLocalidad(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;
        // agregar el objeto commandBuilder 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
producción
    DataRow nuevo = prod.NewRow();
    nuevo["Produccion"] = prod_total; // cantidad total de
    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 agrega 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 realizar 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: “SeriesChartrType.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.BuscarProducción(localidad, cultivo);
            prod.Dispose();
            if (producción == 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	Producción en Toneladas
ARROYITO	2200
Seleccione Tipo de Cultivo	
<input type="checkbox"/> Maiz <input type="checkbox"/> Trigo <input type="checkbox"/> Soja <input type="checkbox"/> Maní	<input type="button" value="Actualizar"/> <input type="button" value="Ver Producción"/> <input type="button" value="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.

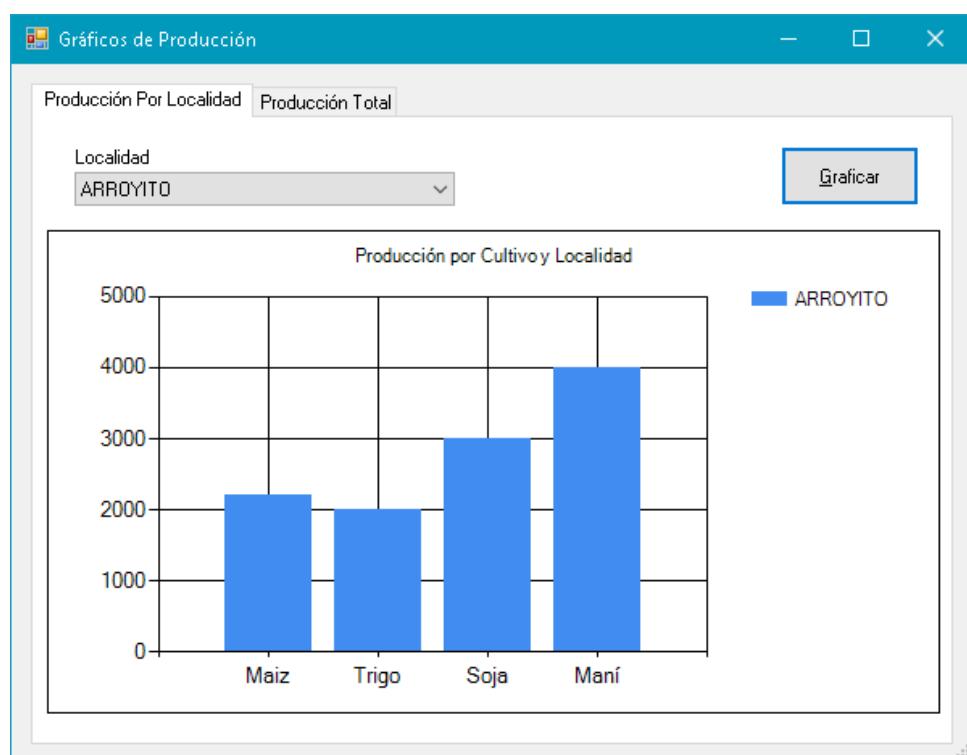


Gráfico de producción total: se observan los valores totales de producción para los cuatro cultivos registrados.



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	Camion	Kilometros	Litros	Precio	kmperLitros	Total	kg	Viaticos
2	Camión 1	3000	300.00	1.25	10	375.00	22000	300
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	16900	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	158.58	1.35	12	211.39	35987	250
10	Camión 9	4500	348.15	1.22	13	422.31	49870	480
11	Camión 10	3900	278.57	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.

SP4/Evaluación de paso

1. Indique la opción correcta

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

- Verdadero
- Falso

2. Indique la opción correcta

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

- Verdadero
- 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;
- 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

5. Indique la opción correcta

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

- Verdadero
- Falso

6. Indique la opción correcta

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

- TabControl1.TabPages.Remove(TabControl1.SelectedTab);
- 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
- Falso

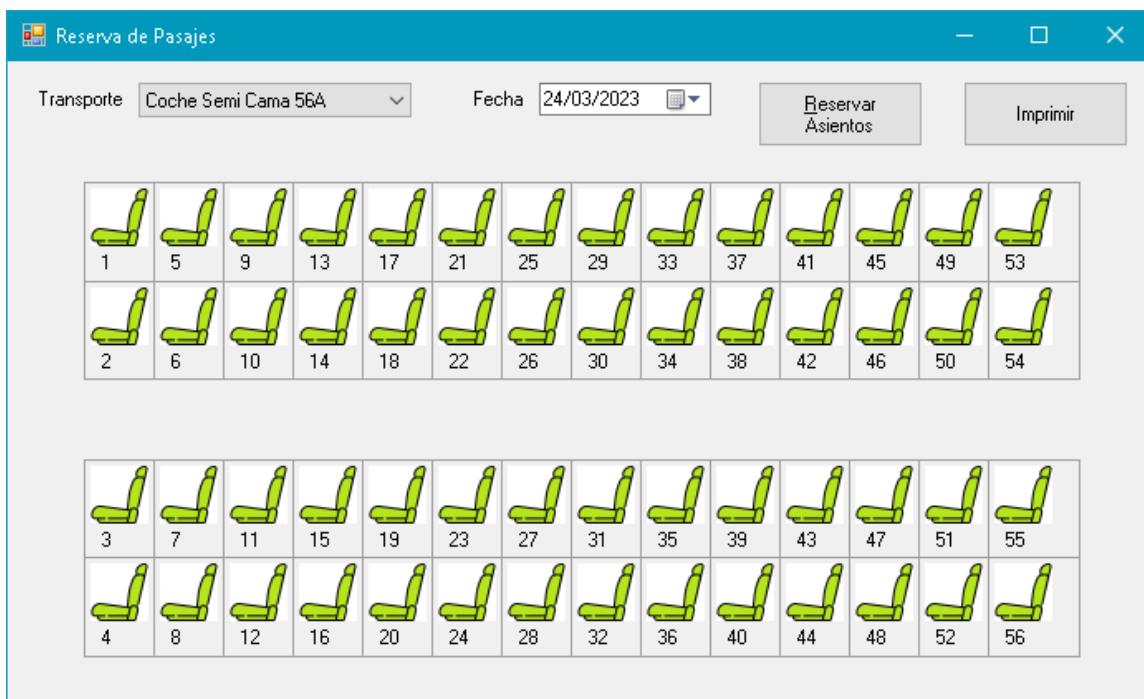
Respuestas correctas²⁰

²⁰1) Verdadero. 2) Verdadero. 3) ProgressBar1.Step = 1;. 4) Falso.. 5) Falso. 6) TabControl1.TabPages.Remove(TabControl1.SelectedTab); 7) Verdadero.

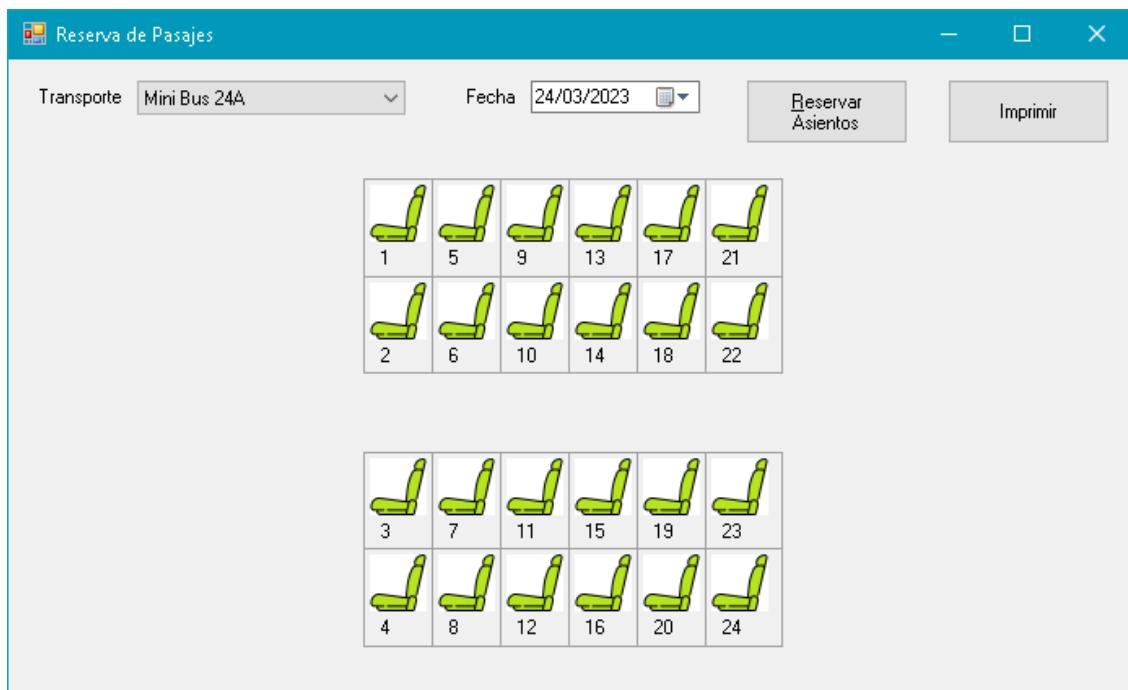
Situación profesional 5: Reserva de pasajes

El administrador de una empresa de transporte de pasajeros le ha solicitado a usted que, en calidad de pasante de la misma, desarrolle una aplicación administrativa para registrar las reservas de pasajes que pueden realizar sus clientes para las diferentes unidades de transporte y fechas de viajes. Además, le solicita que la aplicación genere un reporte con el listado de reservas para una determinada fecha y transporte.

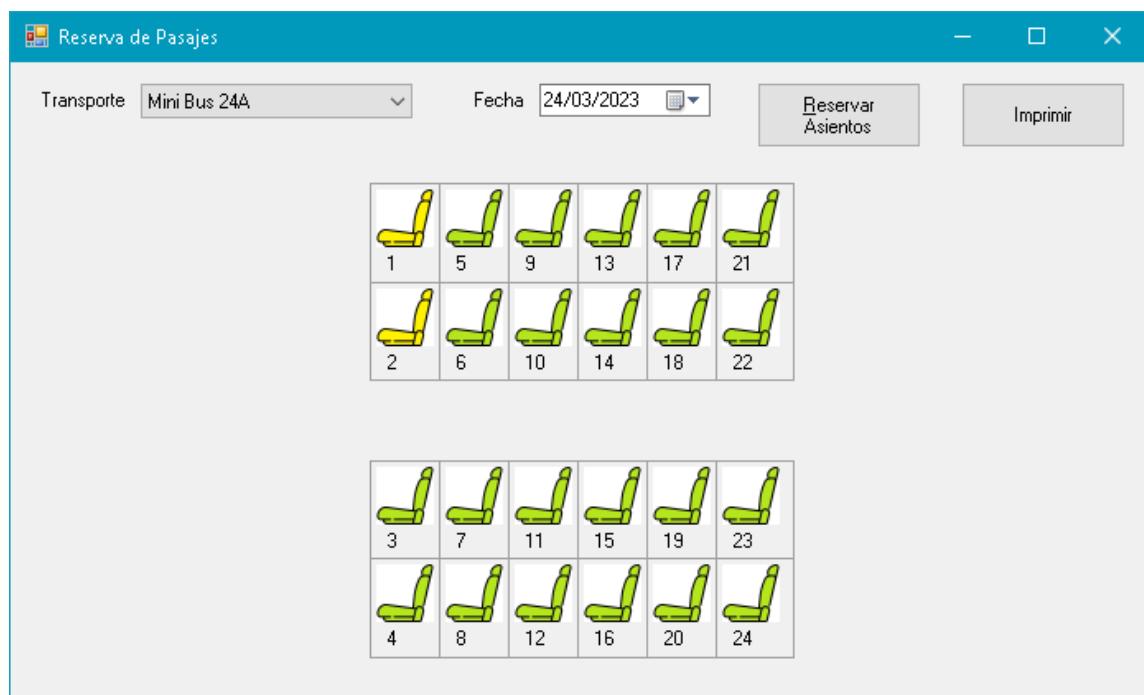
La interfaz gráfica para el registro de reservas tendrá este diseño:



Donde la cantidad y distribución de asientos disponibles será generada en forma dinámica de acuerdo a la capacidad del transporte seleccionado, por ejemplo, para un transporte con menos asientos se vería así:



El usuario de la aplicación debe poder marcar los asientos a reservar directamente haciendo clic sobre la imagen del asiento, al hacerlo, la imagen debe cambiar de color verde a color amarillo:



Y para confirmar la reserva deberá usar el botón “Reservar Asientos”, acción que abrirá un segundo formulario donde se completarán los datos solicitados, documento y nombre de la persona que reserva esos asientos:

The screenshot shows a Windows application window titled "Reserva". It contains the following fields:

- Transporte:** Mini Bus 24A
- Fecha:** 24/03/2023
- DNI:** 25123987
- Nombre:** Pedro Marcheta (highlighted with a blue border)
- Asientos:** A list box containing the numbers 1 and 2.

On the right side of the window are two buttons: "Confirmar Reserva" and "Cancelar".

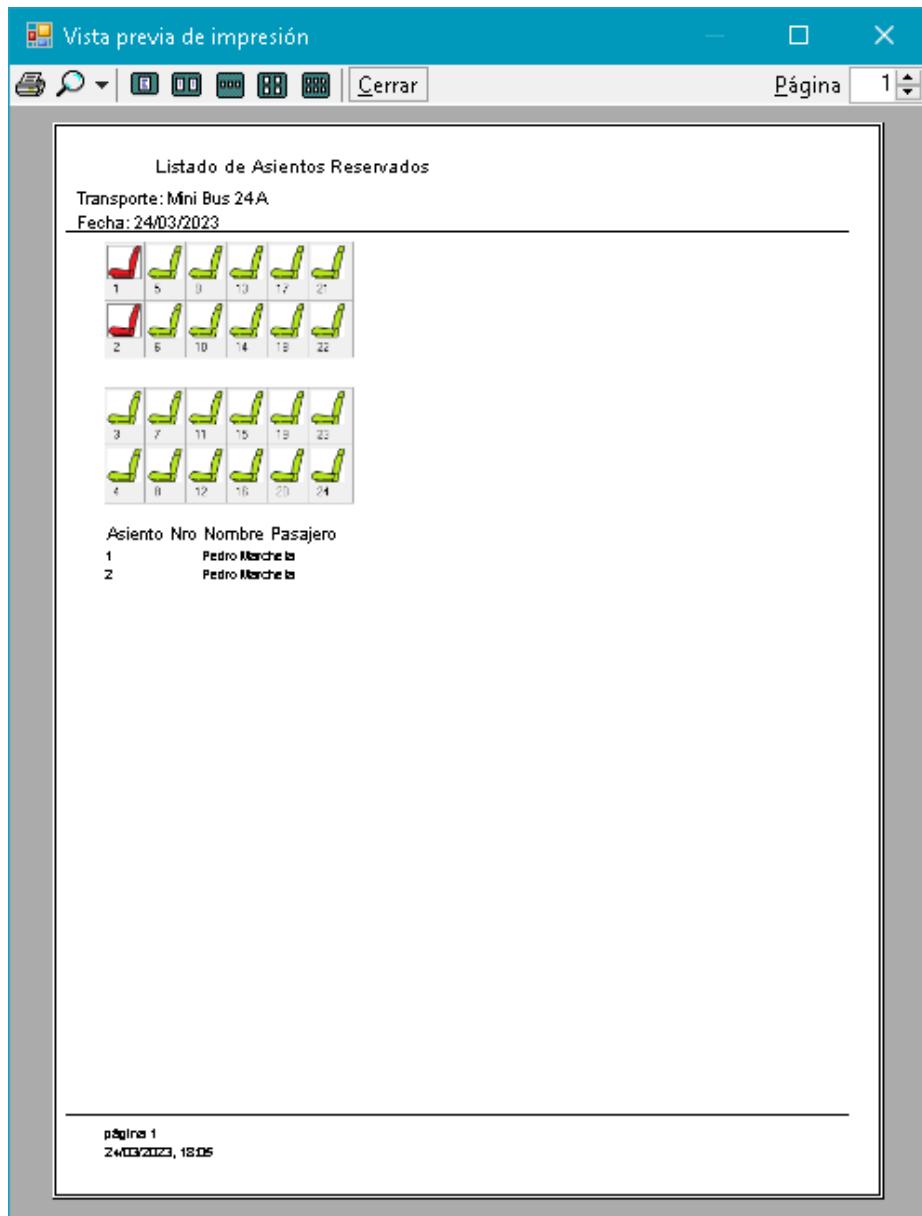
Al confirmar la reserva los asientos seleccionados previamente quedarán de color rojo y ya no se podrán volver a cambiar.

The screenshot shows a Windows application window titled "Reserva de Pasajes". It includes the following controls:

- Transporte:** Mini Bus 24A
- Fecha:** 24/03/2023
- Reservar Asientos** button (highlighted with a blue border)
- Imprimir** button

Below these controls is a seating chart represented as a grid of 24 numbered seats (1-24). Seats 1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, and 22 are highlighted in red, indicating they are reserved. Seats 3, 7, 8, 11, 12, 15, 16, 19, 20, and 23 are green, indicating they are available.

El reporte de reservas se debe generar con el botón “Imprimir”, y mostrará un cuadro de previsualización con la página a imprimir:



El reporte tendrá un encabezado con los datos del transporte y la fecha del viaje, seguidamente se imprimirán los asientos tal como se ven en el formulario, a continuación, los datos de los pasajeros y finalmente en el pie de página se imprimirá el número de página y la fecha y hora en que se generó el reporte.

Se cuenta con una base de datos “Access” denominada “Transporte.mdb” y con estas tablas:

Tabla Transportes

Transportes	
Nombre del campo	Tipo de datos
transporte	Número
Descripcion	Texto corto
Asientos	Número

El campo “transporte” es entero y es la clave principal de la tabla.

El campo “Asientos” es también de tipo entero y almacena la cantidad total de asientos de ese transporte.

Tabla Pasajeros:

Pasajeros	
Nombre del campo	Tipo de datos
Pasajero	Número
Nombre	Texto corto

El campo “Pasajero” es entero, guarda el documento (Dni) del pasajero y forma la clave principal de la tabla.

Tabla Pasajes:

Pasajes	
Nombre del campo	Tipo de datos
Transporte	Número
Fecha	Texto corto
Asiento	Número
Pasajero	Número

Los campos “Transporte”, “Fecha” y “Asiento” forman la clave principal de la tabla.

El campo “Pasajero” guarda el documento del pasajero que reserva el asiento.

La empresa cuenta con varias unidades de transporte, siendo las de mayor capacidad las que tienen 56 asientos, ese valor no puede ser superado en ningún caso.

Ejemplos de transportes:

transporte	Descripcion	Asientos
1	Coché Semi Cama 56A	56
2	Coché Cama 36A	36
3	Mini Bus 24A	24

La cantidad de asientos debe ser siempre múltiplo de 4.

SP5/H1: Control PictureBox

El control **PictureBox** nos permite mostrar imágenes almacenadas en archivos de distinto tipo, tales como *.bmp, *.jpg, *.gif, *.png, etc. Y también sirve para realizar dibujos usando GDI como veremos más adelante.

En nuestra situación profesional será necesario mostrar imágenes predeterminadas, o definidas por el usuario. Por esto, conoceremos cómo manejar y utilizar este tipo de control.

Cabe destacar, que utilizaremos un control PictureBox para contener los asientos de las unidades de transporte que necesitamos graficar.

Las propiedades más usadas de este control son:

BackColor	Sirve para poner un color de fondo al control. El color más usado es el transparente.
SizeMode	Indica cómo se adecua la imagen ante el control, ya sea estirarlo (o mejor dicho autoajustar), tamaño original, zoom y centrado.
BackgroundImageLayout	Nos indica la forma en que se mostrará la imagen: en mosaico, centrado, estirado o zoom.
Image	Establece la imagen a mostrar. Se puede usar en tiempo de diseño o en tiempo de ejecución. En este control también podemos mostrar imágenes en tiempo de ejecución. Hay dos formas de hacerlo: mediante un arreglo de imágenes o mediante la ruta de la imagen.

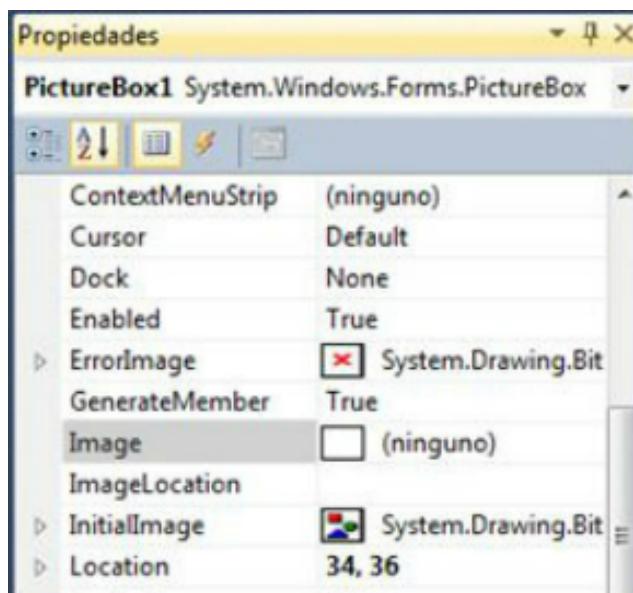
Cargar una imagen en el PictureBox desde un archivo

Necesitamos un archivo con formato gráfico, como jpg, bmp, png, etc, usaremos su nombre y ubicación para la carga del pictureBox:

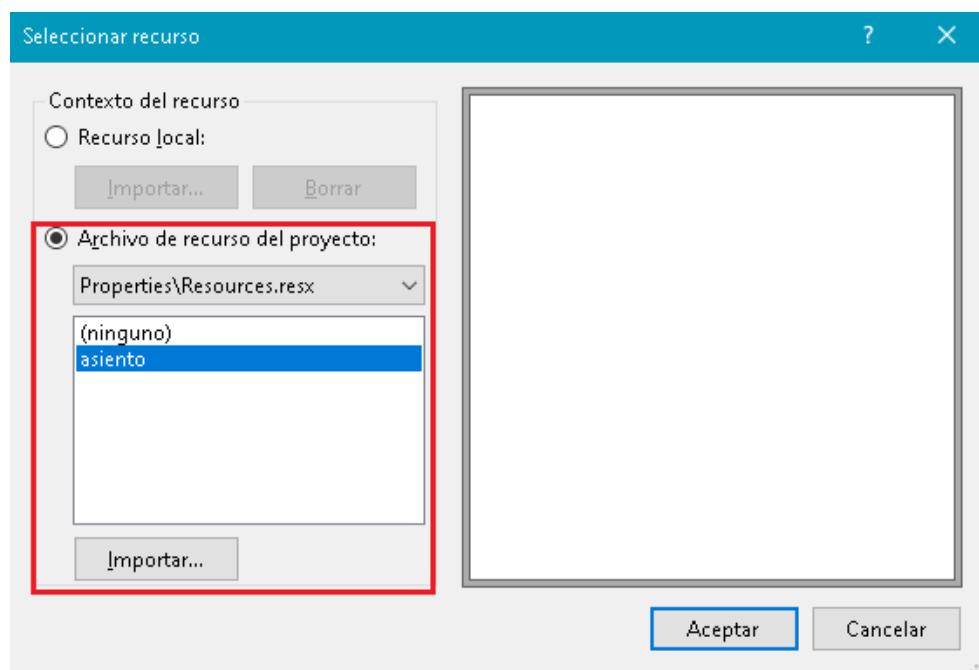
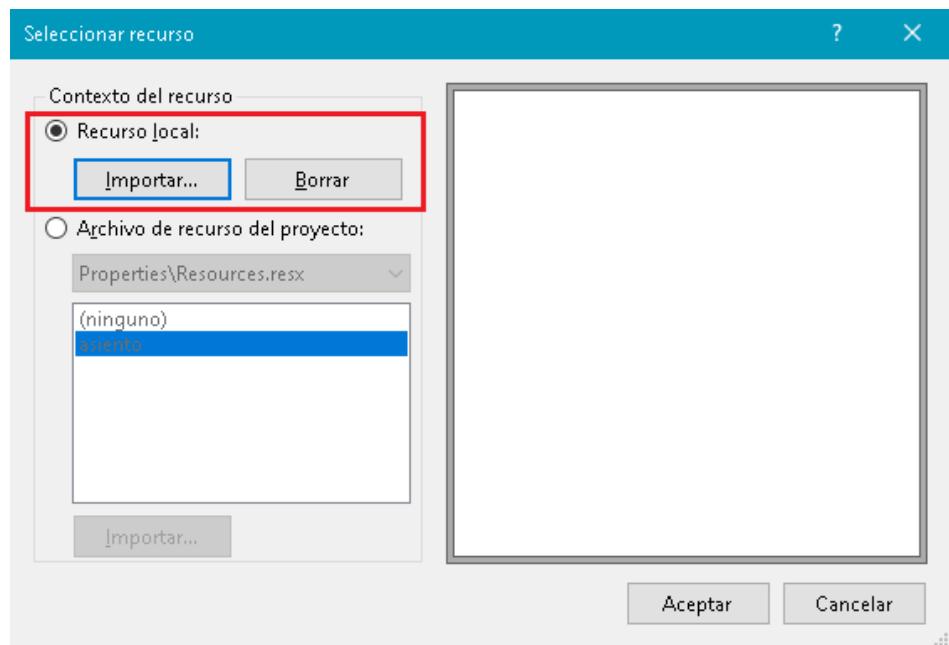
La propiedad “Image” almacena el contenido gráfico del archivo y el método “FromFile” se encarga de transferir la imagen desde el archivo al pictureBox.

```
PictureBox1.Image = Image.FromFile("nombre_archivo");
```

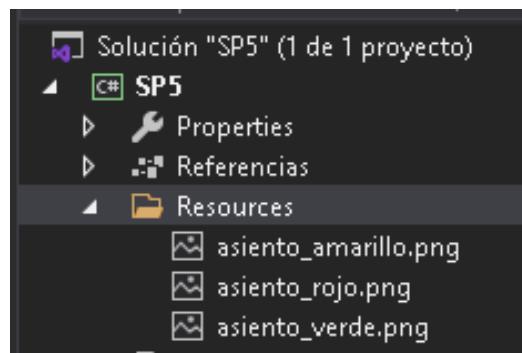
Podemos también cargar imágenes desde arreglos de imágenes. Para hacerlo de esta forma, primero hay que ingresar las imágenes a la sección "Resources", por ejemplo: nos ubicamos en la propiedad image, tal como se muestra en la siguiente imagen:



Una vez ubicado en la propiedad image, haremos clic en el botón de la derecha y se abrirá la siguiente ventana, donde haremos clic en Import e importamos (añadimos) las imágenes que vamos a usar, tal como se muestra a continuación:



Una vez añadidas las imágenes hacemos clic en el botón "Aceptar" y ya tenemos en la sección "Resources" / Recursos, todas las imágenes cargadas e identificadas cada una con su nombre



Para acceder a las imágenes almacenadas por código podemos usar lo siguiente:

```
Image img = Properties.Resources.asiento_verde;  
pictureBox1.Image = img;
```

Donde “asiento_verde” es el nombre del recurso cargado anteriormente.

Además de manejar imágenes, el control PictureBox sirve como contenedor de gráficos generados mediante GDI, o sea que es posible crear un objeto Graphics vinculado al Picturebox y aplicar todos los métodos para dibujar líneas, arcos, círculos, rectángulos, usar diferentes colores, crear texto con distintas fuentes, etc., tal como lo vimos anteriormente.

En el siguiente ejemplo se crea un objeto Graphics desde el mismo control PictureBox y

luego se ejecutan los métodos necesarios para dibujar un texto, y un recuadro. El resultado se muestra en esta imagen:



Código:

```
Graphics gra = pictureBox1.CreateGraphics();
gra.DrawString("Asiento 1", new Font("Arial", 10), Brushes.Green, 10, 15);
gra.DrawRectangle(Pens.Black, 10, 10, 58, 25);
gra.Dispose();
```

SP5/Autoevaluación 1

1. Indique la opción correcta

El control PictureBox sirve para ingresar "mostrar imágenes almacenadas en archivos" solamente.

- Verdadero
- Falso

2. Indique la opción correcta

El método CreateGraphics() del PictureBox crea un objeto de tipo Graphics.

- Verdadero
- Falso

3. Indique la opción correcta

¿Dónde podemos ubicar por código los recursos agregados al proyecto?

- Imports Resources
- Resources
- this.Resources
- Picture.Resources

4. Indique la opción correcta

Para dibujar sobre un PictureBox se necesita crear un objeto de tipo Draw.

- Verdadero
- Falso

5. Indique la opción correcta

¿Cuál es la propiedad que me permite cargar imágenes?

- Picture.Image
- Picture.ImageLoad
- Picture.Image.Load
- Picture.Load

6. Indique la opción correcta

Para mostrar en el PictureBox una imagen almacenada en un archivo se usa la propiedad Picture.

- Verdadero
- Falso

Respuestas correctas²¹

²¹1) Falso. 2) Verdadero. 3) Resources 4) Falso. 5) Picture.Image 6) Falso.

SP5/H2: GDI+. Gráficos

En la situación profesional se plantea la necesidad de trabajar con algunas imágenes y elementos gráficos simples, como texto y líneas, algunos los podremos resolver con el control PictureBox y otros, como en el caso de la impresión, los deberemos resolver con los objetos y métodos que nos aporta el sistema GDI+ (Graphics Device Interface) de Visual Studio.

Nos centraremos ahora en conocer la utilidad de GDI+, luego la utilización en un control PictureBox y, finalmente, la impresión de un resultado que deseemos.

En .NET el CLR (Common Language Runtime) usa una implementación avanzada de la interfaz de diseño de gráficos de Windows, denominada GDI+, que permite crear gráficos, dibujar textos y manejar imágenes gráficas como si fueran objetos.

GDI+ nos permite crear textos y gráficos (líneas, arcos, recuadros, animaciones, etc.) sobre los formularios o sobre otros controles.

Podemos decir que GDI+ es una interfaz de dispositivo gráfico que permite a los programadores escribir aplicaciones independientes del dispositivo.

GDI+ incluye:

Espacio de Nombre. **System.Drawing**

System.Drawing

Proporciona acceso a la funcionalidad de gráficos básica de GDI+.

Se ofrece una funcionalidad más avanzada en los espacios de nombres

- **System.Drawing.Drawing2D**,
- **System.Drawing.Imaging**
- **System.Drawing.Text**.

Clases.

La clase **Graphics**

<https://learn.microsoft.com/es-es/dotnet/api/system.drawing.graphics?view=netframework-4.8.1>

Graphics es la base de la funcionalidad de GDI+. Es la clase la que realmente dibuja líneas, curvas, figuras, imágenes y texto. La clase **Graphics** proporciona métodos para dibujar en el dispositivo de pantalla.

Clases como **Rectangle** y **Point** encapsulan primitivos de GDI+.

La clase **Pen** se utiliza para dibujar líneas y curvas, mientras que las clases derivadas de la clase **Brush** se utilizan para rellenar el interior de las formas.

Enumeraciones.

GDI+ define varias enumeraciones, que son colecciones de constantes relacionadas, por ejemplo, la enumeración **Pens** contiene los elementos Blue, Red, Green y otros, que especifican colores que pueden utilizarse para definir el color del lápiz.

Estructuras.

GDI+ proporciona varias estructuras (por ejemplo, **Rectangle**, **Point** y **Size**) para organizar datos de gráficos. Además, algunas clases sirven principalmente como tipos de datos estructurados. Por ejemplo, la clase **BitmapData** es una clase auxiliar de la clase **Bitmap**, y la clase **PathData** es una clase auxiliar de la clase **GraphicsPath**.

Los servicios de GDI+

Los servicios se exponen a través de un conjunto de clases administradas, se agrupan en tres categorías:

1. Gráficos vectoriales 2D
2. Imágenes
3. Tipografía

1. Gráficos vectoriales 2D

Se refieren al dibujo de líneas, curvas y figuras (tipos primitivos), que se especifican mediante conjuntos de puntos en un sistema de coordenadas (x, y).

2. Imágenes

Existen tipos de imágenes que no se pueden o son muy difíciles de mostrar con las técnicas de gráficos vectoriales.

Por ejemplo:

Las imágenes que aparecen en forma de iconos son complicadas para especificar en forma de líneas y curvas simples. Una fotografía digital de un paisaje resulta aún más difícil de crear con las técnicas vectoriales.

3. Tipografía

Se ocupa de la presentación de textos en diversidad de fuentes, tamaños y estilos. GDI+ proporciona una gran compatibilidad para esta tarea tan compleja.

Método CreateGraphics

Como ya dijimos la generación de objetos gráficos requiere contar con un objeto de la clase “**Graphics**” para poder ejecutar los métodos de dibujo, se deberá partir con la instancia de ese objeto para lo cual disponemos del método “**CreateGraphics**” disponible en todos los objetos que soportan el uso de GDI+, por ejemplo, el formulario, el control PictureBox y los controles de tipo Panel, objetos de tipo Image y la clase PrintDocument entre otros.

```
Graphics gra1 = pictureBox1.CreateGraphics();
Graphics gra2 = this.CreateGraphics();
Graphics gra3 = flwPnlTransporte.CreateGraphics();
Graphics gra4 = cmbTransportes.CreateGraphics();
```

En el ejemplo anterior “this” es la referencia al formulario (Form) donde se ejecuta ese código.

“flwPnlTransporte” es un control de tipo “FlowLayoutPanel”.

“cmbTransportes” es un control de tipo “ComboBox”.

Con todos ellos se puede crear el objeto Graphics para aplicar los métodos gráficos de GDI+.

Una vez creado, se puede usar un objeto Graphics para dibujar líneas y formas, representar texto, o mostrar y manipular imágenes. Los objetos que generalmente se usan con el objeto Graphics son los siguientes:

- **Clase Pen:** (Lápiz) se usa para dibujar líneas, esquematizar formas o representar otras representaciones geométricas.
- **Clase Brush:** (Pincel o Brocha) se usa para llenar áreas de gráficos, como formas llenadas, imágenes o texto.
- **Clase Font:** (Fuente de Texto) proporciona una descripción de las formas que se van a usar al representar texto.
- **Estructura Color:** (Colores) representa los distintos colores que se van a mostrar.

```
Graphics gra = pictureBox1.CreateGraphics();
// dibujar una linea horizontal desde el punto (0,10) al punto (100,10)
gra.DrawLine(Pens.Black, new Point(0, 10), new Point(100, 10));
// dibujar un rectangulo desde el punto (20,30) de ancho 40 y alto 10
gra.DrawRectangle(Pens.Black, 20, 30, 40, 10);
// dibujar un texto a partir del punto (1,50)
gra.DrawString("Texto en font Arial", new Font("Arial", 8), Brushes.Black,
1, 50);
// liberar los recursos
gra.Dispose();
```

En este breve ejemplo creamos el objeto Graphics relacionado al control PictureBox, lo que significa que todos los métodos gráficos que se ejecuten a continuación se verán reflejados sobre la superficie del PictureBox.

El método “**DrawLine**” dibuja una línea con un color de lápiz (Pens.Black), desde el punto (x=0, y=10) hasta el punto (x=100, y=10).

El método “**DrawRectangle**” dibuja un rectángulo con vértice en el punto (x=20, y=30), de ancho 40 y alto 10.

El método “**DrawString**” dibuja el texto “Texto en font Arial”, con tipografía “Arial” de 8 puntos de alto, relleno de color negro (Brushes.Black), a partir del punto (x=1, y=50).

El método “**Dispose**” libera todos los recursos usados por la clase Graphics, de no hacerlo estaríamos ocupando memoria con objetos no usados y perjudicando la ejecución de la aplicación.

En este enlace podrá profundizar sobre las propiedades y métodos de la clase Graphics:

<https://learn.microsoft.com/es-es/dotnet/api/system.drawing.graphics?view=netframework-4.8.1>

El Sistema de coordenadas y los ejes x, y

Cualquier formulario o control contiene tres sistemas de coordenadas:

- **Coordenadas de dispositivo,**
- **Coordenadas de página**
- **coordenadas del mundo.**

Coordenadas de dispositivo (device coordinate system). Son las coordenadas por defecto inamovibles y medidas en píxeles.

Coordenadas de página (page coordinate system). También son inamovibles, pero admiten diferentes unidades de medida, que se establecen llamando al método **PageUnit** del objeto **Graphics**. Este método toma como argumento un miembro de la enumeración **GraphicsUnit**, que especifica la unidad de medida que se utilizará.

Display	1/75 pulgadas.
Document	1/300 pulgadas.
Inch	Pulgada.
Millimeter	Milímetro.

Píxel	Píxel.
Point	Punto de impresión (1/72 pulgadas).
World	La misma que se haya definido para las coordenadas de mundo.

El valor por defecto es píxel, de modo que, si no se modifica, las coordenadas de página coinciden con las del dispositivo.

Coordenadas de mundo (World coordinate system). Salvo en la orientación de los valores X e Y, que en ningún caso se puede cambiar, son totalmente personalizables: pueden moverse y girarse, y se pueden definir unidades de medida personalizadas. Las coordenadas del mundo son las utilizadas por el objeto Graphics. Si no se personalizan, coinciden con las coordenadas de página.

En los tres sistemas el **origen de coordenadas ($x=0, y=0$)** se ubica en la esquina superior izquierda. Los valores X aumentan hacia la derecha y los valores Y aumentan hacia abajo.

Dibujo de Formas Básicas

Las siguientes herramientas, nos permiten profundizar en el uso de las formas básicas que podemos llegar a utilizar en un formulario. Nos será de gran utilidad poder controlar tamaños, colores y formas, para dibujar los gráficos en nuestra situación profesional.

Punto	<p>Un punto es un objeto compuesto por un par de valores de tipo entero, que representan las coordenadas X e Y en el eje de coordenadas. Se construye de la siguiente manera para un punto situado en $x=10$ y $y=10$:</p> <p><code>Dim MiPunto As Point = New Point(10, 10)</code></p>
Línea	<p>Una línea se define por dos puntos y el trazo que los une. El método que dibuja líneas se llama DrawLine:</p> <p><code>Dim Lapiz As New Pen(Color.Blue, 10)</code></p>

	<pre>Dim Punto1 As Point = New Point(10, 10) Dim Punto2 As Point = New Point(100, 100) Lienzo.DrawLine(Lapiz, Punto1, Punto2)</pre> <p>Utilizando la notación sin variables intermedias queda así:</p> <pre>Lienzo.DrawLine(New Pen(Color.Blue, 10), New Point(10, 10), New Point(100, 100))</pre>
Rectángulo	<p>Para Visual Studio, un rectángulo es un polígono de cuatro lados tal que, dado un lado cualquiera, existe otro paralelo a él y, además, todos los vértices forman un ángulo recto. Un cuadrado, por tanto, sólo es un caso particular de rectángulo que se distingue porque todos sus lados son iguales.</p> <p>El objeto Rectangle se compone de las coordenadas de su vértice superior izquierdo, la anchura y la altura. Para dibujarlo invocamos al método DrawRectangle que acepta como parámetros el lápiz y un objeto Rectangle:</p> <pre>Lienzo.DrawRectangle(Lapiz, New Rectangle(100, 100, 200, 100))</pre>
Elipse	<p>El método DrawEllipse toma los mismos argumentos que el método DrawRectangle, pero en vez de dibujar un rectángulo, dibuja una elipse circunscripta en él. Para dibujar un círculo, por tanto, hay que definir un cuadrado:</p> <p>'Dibuja una elipse circunscripta en el rectángulo anterior Lienzo.DrawEllipse(Lapiz, New Rectangle(100, 100, 200, 100))</p> <p>'Dibuja un círculo Lienzo.DrawEllipse(Lapiz, New Rectangle(200, 200, 200, 200))</p>
Arco	<p>Dibujar un arco es igual que dibujar un círculo, añadiéndole dos argumentos más: el punto inicial del arco y el ángulo que forma girando en el sentido de las agujas del reloj, ambos medidos en grados. El método se llama DrawArc:</p> <pre>gr.DrawArc(New Pen(Color.Red, 10), New Rectangle(250, 250, 250, 250), 0, 90)</pre> <p>Entonces, podemos decir que estamos dibujando un arco cuyo punto inicial es el punto 0° y se extiende 90° en el sentido de las agujas del reloj.</p>

Polígono	<p>El método DrawPolygon recibe, además del lápiz, un array de puntos y los une con rectas. DrawPolygon une el primer punto con el último del array formando así un polígono cerrado. El siguiente ejemplo dibuja un rombo:</p> <pre>Lienzo.DrawPolygon(Lapiz, New Point() {New Point(200, 50), New Point(300, 100), New Point(200, 150), New Point(100, 100)})</pre>
-----------------	---

Pincel o brocha

Para conocer en detenimiento cómo podemos modificar el formato de una forma, podemos utilizar pinceles y brochas, que pintan o dibujan de diferentes formas los objetos que realizaremos dentro de un formulario.

Las brochas se encargan de colorear los dibujos. Visual Studio ofrece varios tipos de brochas:

SolidBrush	<p>Rellena la forma con un solo color. Es la brocha más sencilla pues se compone de un solo color:</p> <pre>Dim BrochaSólida As SolidBrush = New SolidBrush(Color.LightGreen)</pre>
HatchBrush	<p>Rellena la forma con dos colores según un modelo predefinido. Define un modelo de distribución de dos colores sin gradación de transición entre ellos, dentro de una forma. Viene a ser un entramado. Los modelos disponibles en VS .NET se encuentran en la enumeración HatchStyle. La clase HatchBrush no se puede heredar, por tanto no se pueden crear modelos personalizados a partir de ella. Con el modelo y dos colores tenemos definido un HatchBrush:</p> <pre>Dim Trama As HatchBrush = New HatchBrush(HatchStyle.Cross, Color.Blue, Color.Cyan)</pre>

TextureBrush	<p>Rellena la forma con una textura.</p> <p>No rellena la forma con colores, sino con una imagen. Una textura consiste en el relleno de una superficie con una imagen repetida tantas veces cuantas sea necesario, hasta cubrir toda la superficie. Los parámetros del constructor de TextureBrush son, por tanto, la imagen y los diferentes modos en que se repiten las imágenes, contenidos en la enumeración WrapMode. He aquí un ejemplo (se supone que tenemos una imagen en un PictureBox):</p> <pre><code>Dim Lienzo As Graphics = Me.CreateGraphics Dim Textura As TextureBrush = New TextureBrush (PictureBox1.Image,_WrapMode.TileFlipX) Lienzo.FillRectangle(Textura, New Rectangle(New Point(0, 0),_New Size(Me.ClientSize.Width, Me.ClientSize.Height))) Textura.Dispose() 'Es conveniente cerrar la brocha cuando no se utilice.</code></pre>
LinearGradientBrush	<p>Rellena un rectángulo con dos colores y transición graduada de uno a otro.</p> <p>Esta brocha rellena un rectángulo con dos colores y transición graduada de uno a otro. Todo lo que tenemos que entregar al constructor es el rectángulo, los dos colores y la orientación de la gradación. La orientación de la gradación está incluida en la enumeración LinearGradientMode.</p>
PathGradientBrush	<p>Rellena la forma con colores que emanan desde el centro y los vértices de la forma, y van graduando su color hasta coincidir con la emanación del color vecino.</p> <p>En vez de definir una gradación de dos colores en una figura cerrada, PathGradientBrush lanza los colores desde el centro y los vértices de la figura.</p> <p>El constructor de PathGradientBrush espera, por tanto, o bien un array de puntos que formarán los vértices de la figura, y de los cuales deducirá el punto central, o bien un trayecto del cual extraerá los vértices y el punto central. Además, también entregar al constructor de debemos PathGradientBrush un color para el punto central y un arreglo de tantos colores como vértices tenga la figura.</p>

Hasta acá hemos visto una introducción al mundo de GDI+, más que suficiente para aplicar a casos simples de programación con gráficos, como sería el caso de nuestra situación profesional, sin embargo, para casos más elaborados habrá que profundizar en más detalles de sus propiedades y métodos, algunas referencias recomendadas para ello son:

Sistemas de Coordenadas y Transformaciones:

[Sistemas de coordenadas y transformaciones - Windows Forms .NET Framework | Microsoft Learn](#)

Uso del lápiz (Pen) para dibujar líneas y rectángulos:

[Utilizar lápiz para dibujar líneas y formas - Windows Forms .NET Framework | Microsoft Learn](#)

Uso del pincel (Brush) para rellenar figuras:

[Utilizar un pincel para llenar formas - Windows Forms .NET Framework | Microsoft Learn](#)

Clase Font:

<https://learn.microsoft.com/es-es/dotnet/api/system.drawing.font?view=netframework-4.8.1>

1. Indique la opción correcta

GDI significa Interfaz de Dispositivo Gráfico.

- Verdadero
- Falso

2. Ordene relaciones

Relacione los conceptos de la columna izquierda con sus características correspondientes en la columna derecha:

a) Pincel	e) Se utiliza con el fin de describir el color que se usa para procesar un objeto determinado.
b) Lápiz	f) Se utiliza para describir la fuente que se usa para procesar textos.
c) Fuente	g) Se utiliza para dibujar líneas y polígonos, junto con rectángulos, arcos y gráficos.
d) Color	h) Se utiliza para llenar superficies delimitadas con esquemas, colores o mapas de bits.

3. Indique la opción correcta

Los valores de los colores a usar se pueden obtener a partir del objeto `Drawing.Color`.

- Verdadero
- Falso

4. Ordene relaciones

Relacione los conceptos de la columna izquierda con sus características correspondientes en la columna derecha:

a) FillRectangle		e) Permite dibujar un cuadrado o rectángulo.
b) FillPolygon		f) Permite dibujar un círculo o elipse.
c) DrawEllipse		g) Permite pintar cualquier polígono.
d) DrawRectangle		h) Permite pintar un cuadrado o rectángulo.

5. Indique la opción correcta

Una vez utilizado el objeto Graphics es conveniente cerrarlo invocando el método:

- Clear
- Changed
- Dispose
- UnLoad

6. Indique la opción correcta

¿Cuál es el espacio de nombre que utilizamos para dibujos lineales?:

- System.Drawing.Line
- System.Drawing.Drawing2D
- System.Drawing.Imaging
- System.Drawing.Text

7. Indique la opción correcta

Para realizar gráficos sobre un formulario se necesita un objeto de tipo Graphics.

- Verdadero
- Falso

8. Indique la opción correcta

El método "FillCircle" del objeto Graphics permite dibujar un círculo.

Verdadero

Falso

9. Indique la opción correcta

El método CreateGraphics sirve para dibujar un nuevo gráfico.

Verdadero

Falso

Respuestas correctas²²

²²1) Verdadero. 2) a→h, b→g, c→f, d→a. 3) Verdadero. 4) a→h, b→g, c→f, d→e. 5) Dispose. 6) System.Drawing.Drawing2D. 7) Verdadero.. 8) Falso. 9) Falso.

SP5/H3: Controles Contenedores de tipo Panel

Revisaremos seguidamente las características y usos de los controles que trabajan como contenedores de otros componentes y que facilitan el maquetado de la interfaz gráfica. Nos referimos a los controles **FlowLayoutPanel**, **TableLayoutPanel** y **Panel**.

Los controles **FlowLayoutPanel** y **TableLayoutPanel** proporcionan formas intuitivas para organizar los controles en un contenedor principal o formulario. Ambos controles proporcionan una capacidad automática y configurable para controlar las posiciones relativas de los controles secundarios que contienen, y lo más importante, ambos ofrecen características de diseño dinámico en tiempo de ejecución, lo que permite cambiar su tamaño y el tamaño y la posición de los controles secundarios a medida que las dimensiones del contenedor primario se modifican. Los paneles de diseño se pueden anidar dentro de otros paneles de diseño para habilitar la creación de interfaces de usuario sofisticadas.

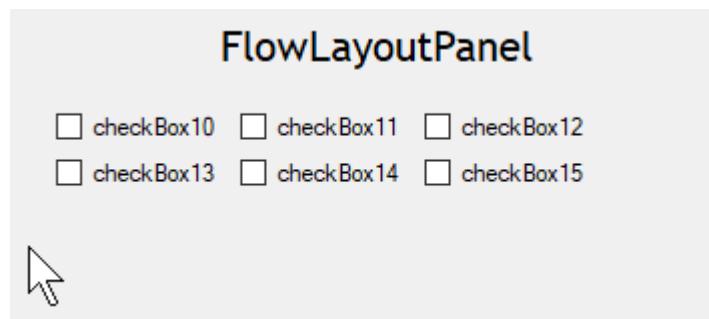
Control FlowLayoutPanel

El control **FlowLayoutPanel** organiza su contenido en una dirección de flujo específica: horizontal o vertical. Su contenido puede ajustarse desde una fila a la siguiente o desde una columna a la siguiente. Además, el contenido puede ajustarse a determinadas dimensiones o puede ajustarse según el caso.

Puede especificar la dirección de flujo estableciendo el valor de la propiedad **FlowDirection**. Puede especificar si el contenido del control **FlowLayoutPanel** se ajustará o se recortará estableciendo el valor de la propiedad **WrapContents**. Con **WrapContents** en **true** el contenido será ajustado y con el valor **false** el contenido será recortado.

El control **FlowLayoutPanel** ajusta automáticamente su tamaño al contenido cuando la propiedad **AutoSize** se establece en **true**.

Cualquier control común de Windows Forms puede ser un control secundario del control FlowLayoutPanel, incluidos otros paneles y otras instancias de FlowLayoutPanel. Con esta característica, se pueden realizar diseños sofisticados de interfaces que se adapten a las dimensiones del contenedor principal o formulario en tiempo de ejecución, ya sea porque se modifica el tamaño del contenedor principal o porque cambia el contenido del FlowLayoutPanel.



Ejemplo de uso del control FlowLayoutPanel

- Se dejó la propiedad FlowDirection con el valor LeftToRight para agregar controles con el flujo de izquierda a derecha.
- Se agregan los controles CheckBox uno a continuación del otro y se van acomodando automáticamente en posiciones consecutivas hasta completar el espacio disponible en una fila, luego el próximo control se ubica en la segunda fila.

El mismo resultado obtenemos ejecutando este código:

```
CheckBox chk = new CheckBox();
chk.Text = "Checkbox 10";
flowLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 11";
flowLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 12";
flowLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 13";
flowLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 14";
flowLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 15";
flowLayoutPanel1.Controls.Add(chk);
```

Se usa la colección de controles que posee como propiedad el control FlowLayoutPanel para agregar uno por uno los controles CheckBox ejecutando el método “Add”.

Propiedades del control FlowLayoutPanel

Entre las propiedades del control tenemos:

AutoSize: indica si el tamaño del control se ajustará automáticamente a su contenido.

AutoSizeMode: indica el comportamiento de ajuste automático de tamaño del control.

BorderStyle: establece el estilo del borde que presenta el control.

Controls: colección de controles contenidos en el FlowLayoutPanel.

Dock: indica si los bordes de control se aplicarán o no a su contenedor primario y el modo en que esto sucede.

FlowDirection: indica la dirección del flujo en el control.

Size: establece el alto y ancho del control

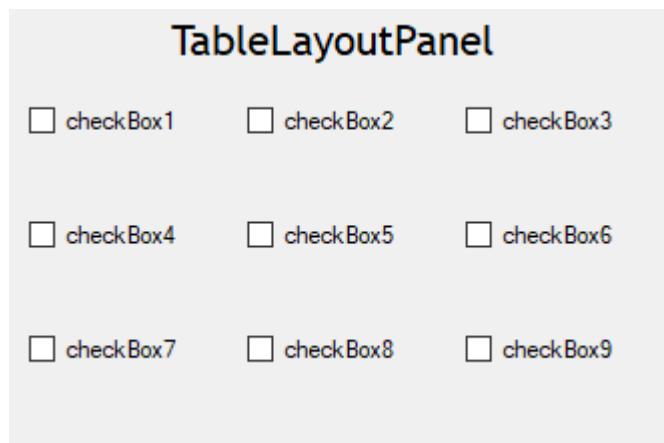
WrapContents: indica si el contenido del control se ajusta a su tamaño o es recortado.

Más detalles en este enlace:

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

Control TableLayoutPanel

El control TableLayoutPanel permite organizar su contenido en una cuadrícula o grilla, lo que proporciona una funcionalidad similar a las tablas de HTML. Las celdas de un control TableLayoutPanel se organizan siempre en filas y columnas, cuya cantidad de elementos se puede configurar y además pueden tener distintos tamaños tanto en alto como en ancho.



Se crea una tabla con tres columnas (ColumnCount = 3) y tres filas (RowCount = 3)

Se ajustó el ancho y alto del control para aprovechar mejor el espacio horizontal y vertical. Se asigna true a la propiedad “AutoSize”.

Se agregó a cada celda un control de tipo CheckBox.

Para lograr el mismo resultado desde el código tenemos:

```
tableLayoutPanel1.ColumnCount = 3;
tableLayoutPanel1.RowCount = 3;
tableLayoutPanel1.CellBorderStyle = TableLayoutPanelCellBorderStyle.Single;
tableLayoutPanel1.AutoSize = true;
chk = new CheckBox();
chk.Text = "Checkbox 1";
tableLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 2";
tableLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 3";
tableLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
chk.Text = "Checkbox 4";
tableLayoutPanel1.Controls.Add(chk);
chk = new CheckBox();
// se repite para el resto de controles CheckBox
```

Cualquier control estándar de Windows Forms puede ser un control secundario del control TableLayoutPanel, incluidas otras instancias de TableLayoutPanel. Esto permite construir diseños de interfaces más elaborados que se adaptan a los cambios en el tiempo de ejecución.

El control TableLayoutPanel puede expandirse para acomodar nuevos controles cuando se agreguen, dependiendo del valor de las propiedades **RowCount**, **ColumnCount** y **GrowStyle**. Establecer las propiedades RowCount o ColumnCount en un valor de 0 especifica que el TableLayoutPanel se desenlaza en la dirección correspondiente.

También se puede controlar la dirección de expansión cuando el contenido del control TableLayoutPanel se llena de controles secundarios. De forma predeterminada, el control TableLayoutPanel se expande hacia abajo (en vertical) agregando filas.

Principales propiedades del control TableLayoutPanel

- **AutoSize**: indica si el tamaño del control se ajustará automáticamente a su contenido.
- **AutoSizeMode**: indica el comportamiento de ajuste automático de tamaño del control.
- **BorderStyle**: establece el estilo del borde que presenta el control.
- **ColumnCount**: cantidad de columnas del control
- **ColumnStyles**: estilos de las columnas
- **Controls**: colección de controles contenidos en el FlowLayoutPanel.
- **Dock**: indica si los bordes de control se aplicarán o no a su contenedor primario y el modo en que esto sucede.
- **GrowStyle**: indica si el control debe expandirse para agregar un nuevo control cuando todas las celdas ya están ocupadas.
- **RowCount**: cantidad de filas del control
- **RowStyles**: colección de estilos para las filas del control.
- **Size**: establece el alto y ancho del control

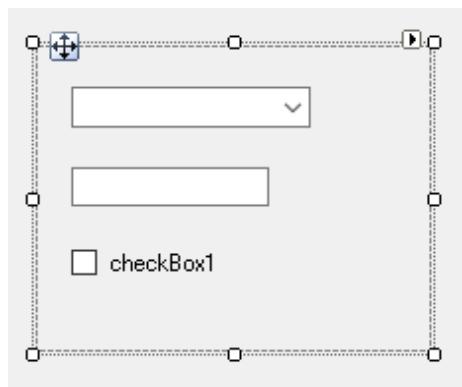
Control Panel

El control de tipo Panel se usa para proporcionar una agrupación para otros controles. Uno de los usos más frecuentes del Panel consiste en subdividir el contenido de un formulario por función. Por ejemplo, las diferentes opciones de

pago que tiene el cliente para realizar una compra. La agrupación de todas las opciones en un panel proporciona al usuario una indicación visual lógica. En tiempo de diseño, todos los controles se pueden mover fácilmente; cuando se mueve el control Panel, también se mueven todos los controles que contenga. Otra facilidad que otorga el uso de un panel es poder ocultar o visualizar su contenido en un solo paso, sin importar la cantidad y tipo de controles que contenga en su interior.

Se puede acceder a los controles incluidos en un panel a través de su propiedad **Controls**. Esta propiedad devuelve una colección de instancias de tipo Control, por lo que normalmente habrá que convertir un control recuperado de esta manera a su tipo específico.

Ejemplo: un panel (panel1) que contiene un control ComboBox, un TextBox y un CheckBox



Podemos recorrer la colección de controles del panel de esta forma:

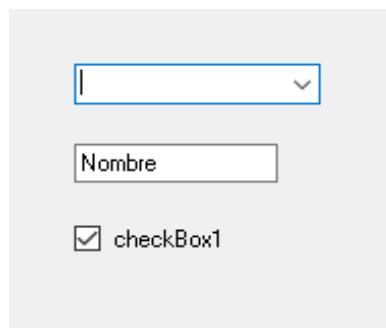
```
foreach(Control c in panel1.Controls)
{
    if(c.Name == "comboBox1")
    {
        ((ComboBox)c).Items.Clear(); // cuando 'c' es un ComboBox
    }
    if(c.Name == "textBox1")
    {
        ((TextBox)c).Text = "Nombre"; // cuando 'c' es un TextBox
    }
    if(c.Name == "checkBox1")
    {
        (c as CheckBox).Checked = true; // cuando 'c' es un CheckBox
    }
}
```

En el interior del ciclo el objeto “c” toma los valores de cada una de las instancias de los controles colocados en el control “panel1”, a partir de eso se puede acceder a cada uno de los controles por su nombre o su tipo y modificar su estado o ejecutar sus métodos.

Se puede usar un “cast” con el nombre del tipo de objeto: ((ComboBox)c), esto considera al objeto “c” que en ese momento es de tipo “Control” como un ComboBox.

El mismo resultado se logra con la expresión “as”: (c as CheckBox), la expresión resultante será tratada como un objeto de tipo CheckBox.

Al ejecutar ese bloque de código obtenemos:



El control Panel es similar al control GroupBox; sin embargo, solo el control Panel puede tener barras de desplazamiento, y solo el control GroupBox puede mostrar un título, por lo demás su funcionalidad para agrupar otros controles es muy similar.

Principales propiedades del control Panel

Para visualizar las barras de desplazamiento, establezca la propiedad **AutoScroll** en true.

Puede personalizar la apariencia del panel estableciendo las propiedades **BackColor**, **BorderStyle** y **BackgroundImage**.

La propiedad `BorderStyle` determina si el panel se presentará sin borde visible (`None`), con una línea simple (`FixedSingle`) o con una línea sombreada (`Fixed3D`).

1. Indique la opción correcta

El control FlowLayoutPanel se usa como contenedor de otros controles.

- Verdadero
- Falso

2. Indique la opción correcta

En un control FlowLayoutPanel se pueden agregar solamente controles de un solo tipo.

- Verdadero
- Falso

3. Indique la opción correcta

El control TableLayoutPanel es un contenedor igual al control GroupBox que posee una propiedad para establecer el título.

- Verdadero
- Falso

4. Indique la opción correcta

El control TableLayoutPanel organiza su contenido en filas columnas.

- Verdadero
- Falso

5. Indique la opción correcta

El control Panel sirve para agrupar controles y lograr una separación lógica de otros controles en el formulario.

- Verdadero
- Falso

6. Indique la opción correcta

El control Panel puede cambiar su tamaño automáticamente según su contenido.

- Verdadero
- Falso

Respuestas correctas²³

²³1) Verdadero 2) Falso. 3) Falso . 4) Verdadero. 5) Verdadero. 6) Verdadero.

SP5/H4: Impresión

Controles PrintDialog, PageSetupDialog y PrintPreviewDialog, objeto PrintDocument

Para realizar la impresión de reportes, gráficos, listados, etc. como por ejemplo el listado de reservas que se solicita en esta situación profesional, utilizaremos un conjunto de elementos que nos provee el lenguaje y framework de .NET, el soporte para impresión se obtiene principalmente de la clase **PrintDocument** (sus métodos, eventos y propiedades), facilitando enormemente la tarea de controlar la impresión desde nuestras aplicaciones. A esto se agregan los diálogos para configurar páginas (**PageSetupDialog**), para pre-visualizar los documentos (**PrintPreviewDialog**) y para seleccionar la impresora a usar (**PrintDialog**).

Todo esto conforma un sólido conjunto que cubre todas las necesidades que normalmente se plantean en las aplicaciones de escritorio Windows.

Se usará el espacio de nombres: **System.Drawing.Printing**

Normalmente, para poder imprimir un documento deberá contar con una instancia del componente **PrintDocument**, luego establecerá las propiedades que describen dónde se va a imprimir mediante las clases **PrinterSettings** (configuraciones de impresora) y **PageSettings** (configuraciones de página), y, finalmente, llamará al método **Print** para imprimir realmente el documento.

Mientras se lleva a cabo el proceso de impresión desde una aplicación Windows, el componente **PrintDocument** mostrará un cuadro de diálogo de anulación de impresión, para avisar a los usuarios que se está produciendo la impresión y para permitir la cancelación del trabajo de impresión.

La base de la impresión en formularios *Windows Forms* es el componente **PrintDocument** y, más específicamente, el evento **PrintPage**. Al escribir código de control para este evento, puede especificar qué se va a imprimir y cómo.

Las actividades para crear un trabajo de impresión son:

1. Agregar un componente PrintDocument al formulario o crear uno por código.
2. Sobre el componente agregado hacer doble clic con el botón del mouse, esta acción creará el evento “PrintPage”. También se puede agregar el evento desde la ventana de propiedades.
3. Escribir el código para el evento PrintPage. Acá tendrá que definir su propia lógica de impresión. Eso incluye el proceso de obtener los datos a imprimir, por ejemplo, de una base de datos. También podrá crear objetos gráficos con GDI, como líneas, recuadros, texto, etc. Se usa el objeto Graphics y sus métodos para dar forma al contenido a imprimir, como si se dibujara en una página en blanco que se mantiene en memoria hasta que se transfiera a la impresora real. Para finalizar la impresión se debe asignar falso a la propiedad “HasMorePages”.
4. Invocar el método “Print” del objeto PrintDocument creado en el paso 1, al ejecutarse este método se dispara el evento “PrintPage” y se inicia la impresión propiamente dicha. De forma predeterminada la impresión se dirige siempre a la impresora que tenga configurada por defecto el sistema de Windows.

Ejemplo: al ejecutar el método “Print” se dispara el evento “PrintPage”

```
private void btnImprimir_Click(object sender, EventArgs e)
{
    printDocument1.Print();
}
```

```
private void printDocument1_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
    // dibujar un texto
    e.Graphics.DrawString("Impresión de ejemplo",
        new Font("Arial", 12),
        Brushes.Black,
        new Point(20, 30));
    // dibujar una imagen a partir de un archivo bmp
    e.Graphics.DrawImage(Image.FromFile("C:/Imagenes/logo.bmp"),
        new Point(20, 50));
    // finaliza la impresión
    e.HasMorePages = false;
}
```

El objeto ‘e’ de tipo “PrintPageEventArgs” contiene el objeto “Graphics” con el que se realizan los dibujos a imprimir.

La Clase PrintDocument

A continuación, se describen las propiedades, métodos y eventos más importantes de la clase PrintDocument:

Constructores públicos	
◆ PrintDocument <i>(Constructor)</i>	Inicializa una nueva instancia de la clase PrintDocument .
Propiedades públicas	
▣ DefaultPageSettings	Obtiene o establece la configuración de página que se utiliza como predeterminada para todas las páginas que se van a imprimir.
▣ DocumentName	Obtiene o establece el nombre del documento que va a aparecer mientras se imprime el documento.
▣ OriginAtMargins	Obtiene o establece un valor que indica si un objeto gráfico asociado a una página está situado justo dentro de los márgenes especificados por el usuario o en la esquina superior izquierda del área de impresión de la página.
▣ PrinterSettings	Obtiene o establece la impresora que imprime el documento.
Métodos públicos más usados	
◆ Dispose <i>(se hereda de Component)</i>	Sobrecargado. Libera los recursos utilizados por Component .
◆ Print	Inicia el proceso de impresión del documento.
◆ ToString	Reemplazado. Vea Object.ToString .

Eventos públicos más usados	
 BeginPrint	Se produce cuando se llama al método Print antes de que se imprima la primera página del documento.
 EndPrint	Se produce cuando se ha impreso la última página del documento.
 PrintPage	Se produce cuando se necesita el resultado que se va a imprimir para la página actual.
Métodos protegidos	
 OnBeginPrint	Provoca el evento BeginPrint . Se llama después de llamar al método Print y antes de que se imprima la primera página del documento.
 OnEndPrint	Provoca el evento EndPrint . Se llama cuando se ha impreso la última página del documento.
 OnPrintPage	Provoca el evento PrintPage . Se llama antes de que se imprima una página.
 OnQueryPageSettings	Provoca el evento QueryPageSettings . Se llama justo antes de cada evento PrintPage .

Otras tareas importantes relacionadas con la impresión son:

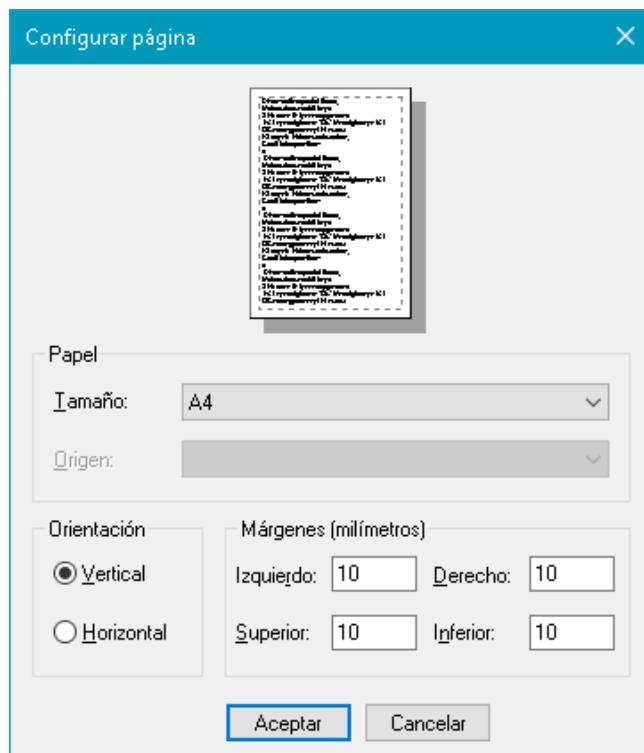
1. Modificar las opciones de impresión seleccionadas mediante programación, por medio del componente **PageSetupDialog**.
2. Cambiar la impresora utilizada para la impresión por medio del componente **PrintDialog**, en tiempo de ejecución.
3. Mostrar a los usuarios una vista preliminar del trabajo a imprimir usando el control **PrintPreviewDialog**.

Veremos entonces las características de estos tres componentes.

1. **PageSetupDialog**

El componente `PageSetupDialog` de formularios Windows Forms es un cuadro de diálogo **preconfigurado** que permite establecer los detalles de la página para imprimir en aplicaciones Windows.

Este diálogo puede permitir que los usuarios establezcan ajustes de borde y margen, encabezados y pies de página, y orientación vertical y horizontal. Al tratarse de diálogos estándar de Windows, las aplicaciones resultan inmediatamente familiares para los usuarios.



Utilice el método `ShowDialog` para mostrar el cuadro de diálogo en tiempo de ejecución. En este componente se pueden establecer las propiedades relacionadas con una sola página (clase `PrintDocument`) o con cualquier documento (clase `PageSettings`).

Además, el componente `PageSetupDialog` puede utilizarse para determinar configuraciones específicas de la impresora, que se almacenan en la clase `PrinterSettings`.

Un aspecto importante del trabajo con el componente `PageSetupDialog` es cómo interactúa con la clase `PageSettings`.

La clase `PageSettings` se utiliza para especificar configuraciones que modifican el modo en que se imprimirá una página, como la orientación del papel, el tamaño de la página y los márgenes.

Cada una de estas configuraciones se representa como una propiedad de la clase `PageSettings`. La clase `PageSetupDialog` modifica estos valores de propiedad para una instancia dada de la clase `PageSettings` que está asociada con el documento (y se representa como la propiedad `PrintDocument.DefaultPageSettings`).

Para definir las propiedades de página utilizando el componente `PageSetupDialog` usted debe utilizar el método `ShowDialog` para mostrar el cuadro de diálogo, especificando el componente `PrintDocument` que se utilizará.

En el ejemplo siguiente, el controlador de eventos `Click` del control `Button` usa el componente `PageSetupDialog1`. En la propiedad `Document` se especifica un documento existente. Luego se ejecuta el método `ShowDialog` para que el usuario haga los ajustes deseados (tamaño de página, orientación, márgenes) y si confirma las selecciones con el botón `Aceptar` se imprime el documento.

En el ejemplo se supone que el formulario tiene un control `btnImprimir`, un componente `printDocument1` y un componente `pageSetupDialog1`.

```
private void btnImprimir_Click(object sender, EventArgs e)
{
    // se asigna el documento a configurar
    pageSetupDialog1.Document = printDocument1;
    // se muestra el diálogo para configurar la página
    if (pageSetupDialog1.ShowDialog() == DialogResult.OK)
    {
        // si se confirman los cambios (botón OK), se imprime
        printDocument1.Print();
    }
}
```

Las principales propiedades de la clase `PageSetupDialog` son:

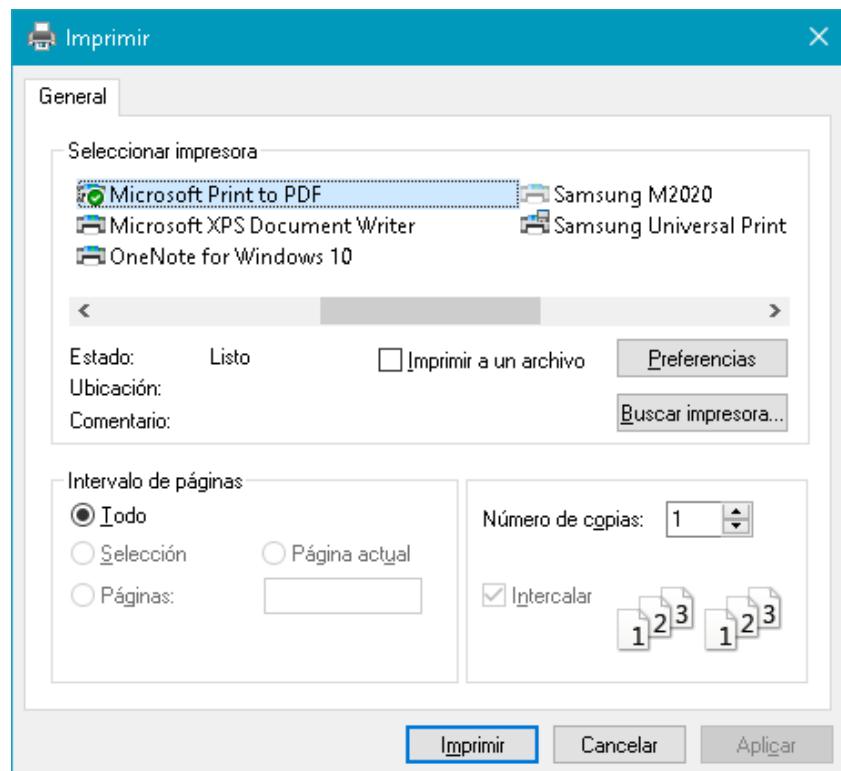
Propiedades públicas	
 AllowMargins	Obtiene o establece un valor que indica si está habilitada la sección de márgenes del cuadro de diálogo.
 AllowOrientation	Obtiene o establece un valor que indica si está habilitada la sección de orientación (horizontal o vertical) del cuadro de diálogo.
 AllowPaper	Obtiene o establece un valor que indica si se habilita la sección de papel (tamaño y origen de papel) del cuadro de diálogo.
 AllowPrinter	Obtiene o establece un valor que indica si el botón Impresora está habilitado.
 Document	Obtiene o establece un valor que indica el <i>PrintDocument</i> del que se obtendrá la configuración de página.
 MinMargins	Obtiene o establece un valor que indica los márgenes mínimos que se permiten seleccionar, expresados en centésimas de pulgadas.
 PageSettings	Obtiene o establece un valor que indica la configuración de la página que se va a modificar.
 PrinterSettings	Obtiene o establece la configuración de impresora que se modifica cuando el usuario hace clic en el botón Impresora del cuadro de diálogo.
 ShowNetwork	Obtiene o establece un valor que indica si está visible el botón Red.

2. PrintDialog

El componente **PrintDialog** de formularios Windows Forms es un cuadro de diálogo preconfigurado, que se utiliza para seleccionar una impresora, elegir las páginas que se van a imprimir y determinar otras configuraciones relacionadas con la impresión en aplicaciones para Windows.

Permite que los usuarios impriman diversas partes de sus documentos:

- **imprimir todo,**
- **imprimir un intervalo de páginas seleccionado**
- **imprimir una selección.**



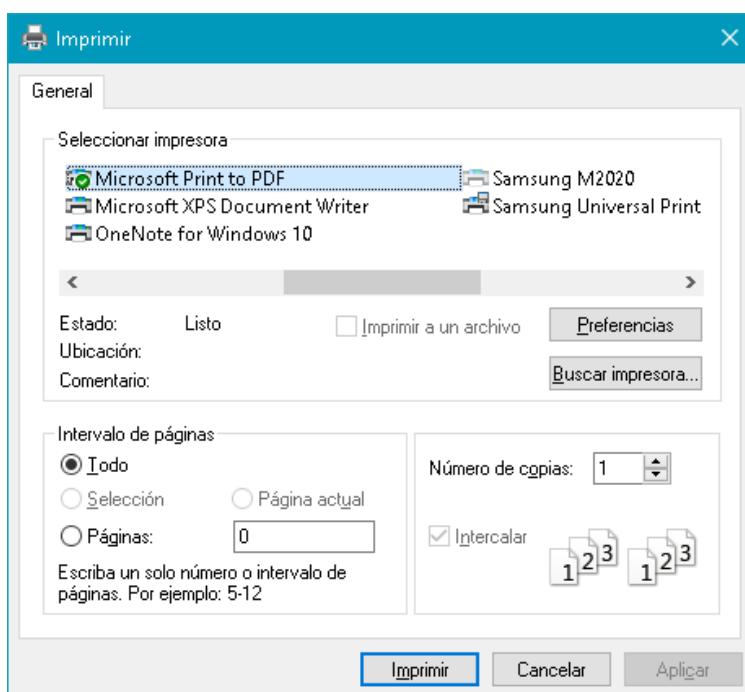
Utilice el método **ShowDialog** para mostrar el cuadro de diálogo en tiempo de ejecución.

Este componente tiene propiedades relacionadas con un único trabajo de impresión (clase **PrintDocument**) o con las configuraciones de una impresora individual (clase **PrinterSettings**). Cualquiera de ellas, a su vez, puede ser compartida por múltiples impresoras.

Ejemplo:

```
private void btnImprimir_Click(object sender, EventArgs e)
{
    // se asigna el documento a imprimir
    printDialog1.Document = printDocument1;
    // permitir elegir la/s páginas a imprimir
    printDialog1.AllowSomePages = true;
    // deshabilitar la opción de imprimir en un archivo
    printDialog1.PrintToFile = false;
    // se muestra el diálogo para configurar la impresora
    if (printDialog1.ShowDialog() == DialogResult.OK)
    {
        // si se confirman la selección (botón Aceptar), se imprime
        printDocument1.Print();
    }
}
```

Se obtiene esta presentación:

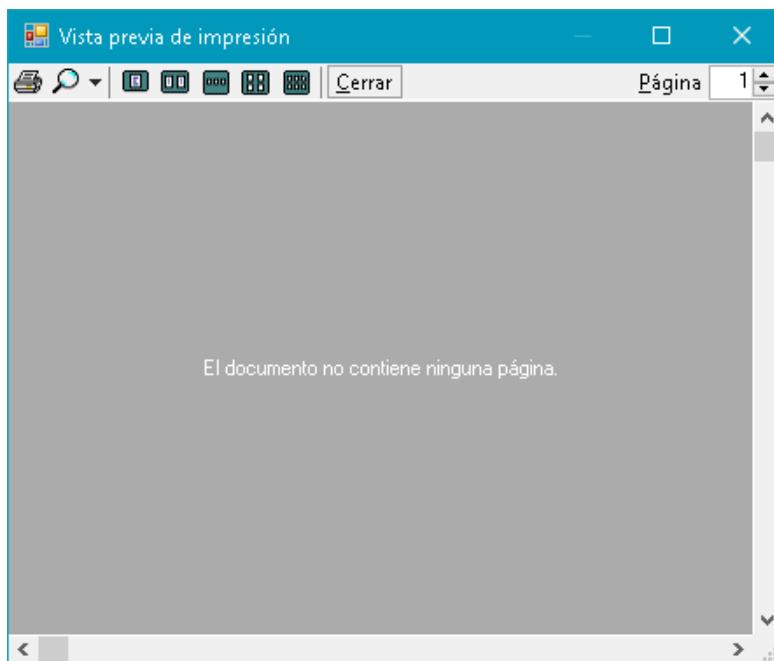


Las principales propiedades de la clase `PrintDialog` son:

Propiedades públicas	
<code>AllowPrintToFile</code>	Obtiene o establece un valor que indica si la casilla de verificación Imprimir a un archivo está activada.
<code>AllowSelection</code>	Obtiene o establece un valor que indica si está habilitado el botón de opción Página desde... hasta...
<code>AllowSomePages</code>	Obtiene o establece un valor que indica si se habilita el botón de opción Páginas.
<code>Document</code>	Obtiene o establece un valor que indica el <code>PrintDocument</code> del que se obtendrá <code>PrinterSettings</code> .
 <code>PrinterSettings</code>	Obtiene o establece la configuración de impresora que se modifica en el cuadro de diálogo.
<code>PrintToFile</code>	Obtiene o establece un valor que indica si la casilla de verificación Imprimir a un archivo está activada.
<code>ShowNetwork</code>	Obtiene o establece un valor que indica si se muestra el botón Red.

3. PrintPreviewDialog

El control **PrintPreviewDialog** se utiliza para mostrar el contenido de un documento, antes de imprimirllo. El control contiene botones para imprimir, acercar, mostrar una o varias páginas y cerrar el cuadro de diálogo. Debe especificar una instancia de la clase **PrintDocument**.



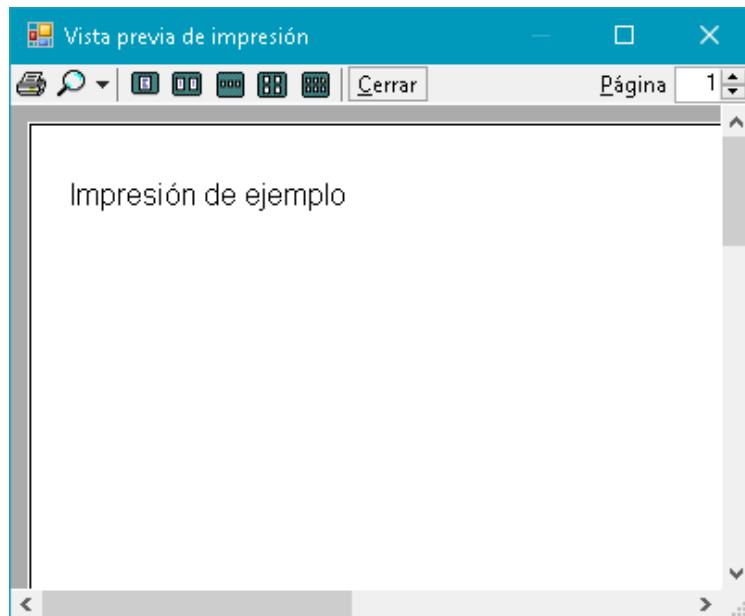
El control **PrintPreviewDialog** utiliza la clase **PrinterSettings**. Asimismo, el control **PrintPreviewDialog** utiliza la clase **PageSettings**, igual que el componente **PageSetupDialog**. El documento para imprimir especificado en la propiedad **Document** hace referencia a instancias de las clases **PrinterSettings** y **PageSettings**, y éstas se utilizan para representar el documento en la ventana de vista previa.

Para pre-visualizar páginas con el control **PrintPreviewDialog** utilice el método **ShowDialog** para mostrar el cuadro de diálogo, especificando el componente **PrintDocument** que se utilizará.

Ejemplo:

```
private void btnImprimir_Click(object sender, EventArgs e)
{
    // asignar el documento a previsualizar
    printPreviewDialog1.Document = printDocument1;
    // mostrar el diálogo
    printPreviewDialog1.ShowDialog();
}
```

Se obtiene:



Las principales propiedades de la clase PrintPreviewDialog son:

Propiedades públicas	
 Bounds (se hereda de Control)	Obtiene o establece el tamaño y la ubicación del control, incluidos los elementos de no cliente.
 CanFocus (se hereda de Control)	Obtiene un valor que indica si el control puede recibir el foco.
 CanSelect (se hereda de Control)	Obtiene un valor que indica si el control se puede seleccionar.
 DialogResult (se hereda de Form)	Obtiene o establece el resultado de cuadro de diálogo para el formulario.
 Document	Obtiene o establece el documento del que se desea la vista previa.
 Name (se hereda de Control)	Obtiene o establece el nombre del control.

Recomendamos revisar con más detalle estos componentes para dotar a las aplicaciones de la mejor funcionalidad que se requiera en cada caso, facilitando la tarea de los usuarios finales.

Algunos enlaces para tener en cuenta:

- **PageSetupDialog:**

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

- **PrintDialog:**

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

- **PrintPreviewDialog:**

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

- **PrintDocument:**

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

1. Indique la opción correcta

¿Cuál de las siguientes sentencias nos permite modificar las opciones de impresión de la página?

- PrintPreviewDialog
- PrintDialog
- PageSetupDialog

2. Indique la opción correcta

El método PrintDoc de un objeto PrintDocument inicia el proceso de impresión.

- Verdadero
- Falso

3. Indique la opción correcta

La clase PrintDocument es la base para controlar los trabajos de impresión.

- Verdadero
- Falso

4. Indique la opción correcta

Para detener el trabajo de impresión se usa el método EndPrint.

- Verdadero
- Falso

5. Indique la opción correcta

¿Cuál de estas sentencias de códigos nos mostrará una ventana para confirmar nuestra impresión?

- ShowDialog();
- PrintDialog1.ShowDialog();
- Print.ShowDialog();
- Dialog1.Dialog();

6. Indique la opción correcta

Para usar el cuadro de diálogo PrintPreviewDialog se necesita tener al menos una impresora configurada.

- Verdadero
- Falso

7. Indique la opción correcta

El evento PrintPage se usa para codificar la lógica del proceso que controla la impresión.

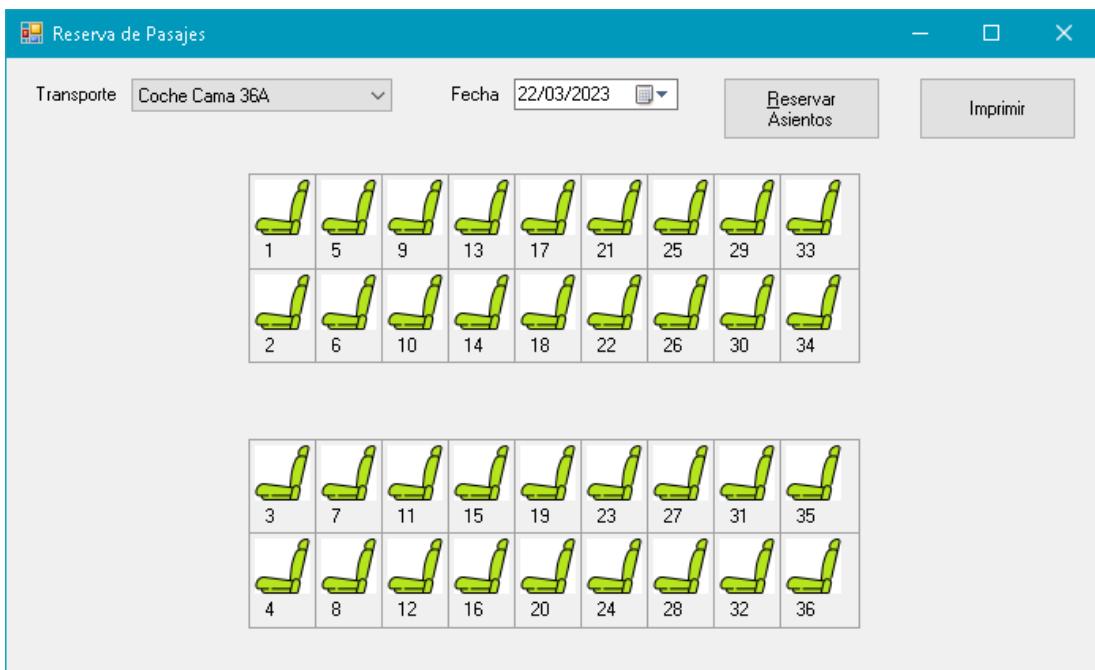
- Verdadero
- Falso

Respuestas correctas²⁴

²⁴2) PageSetupDialog. 2) Falso. 3) Verdadero. 4) Falso. 5)
PrintDialog1.ShowDialog(); 6) Verdadero. 7) Verdadero.

SP5/Ejercicio resuelto

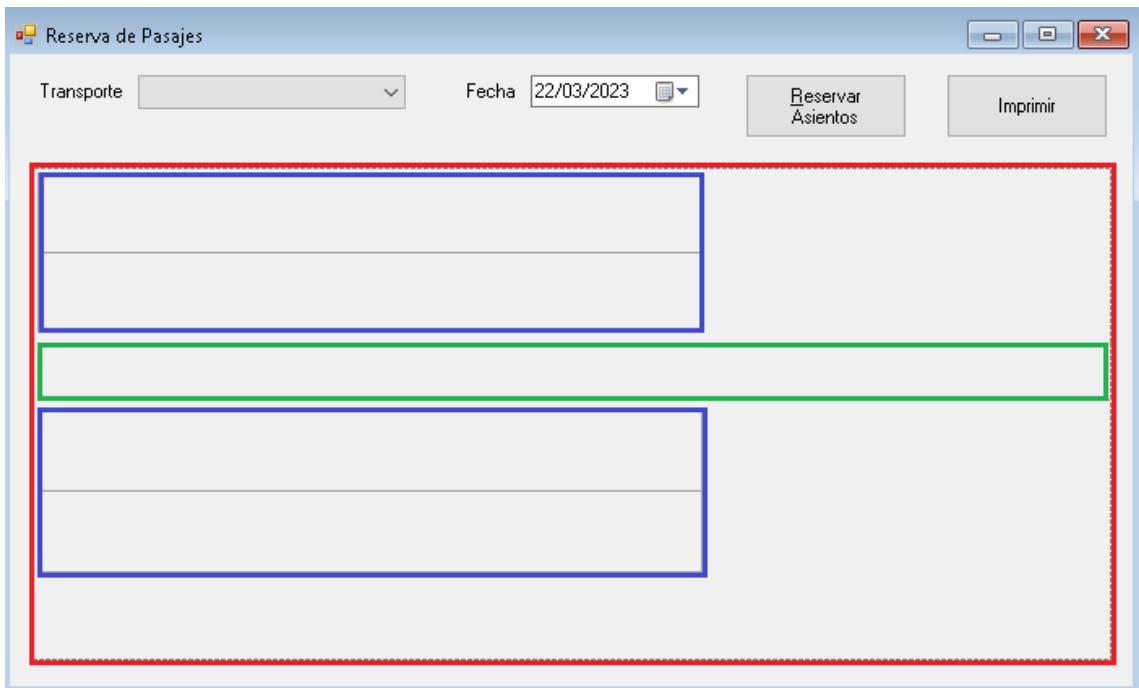
Comenzamos la resolución de esta situación profesional creando un nuevo proyecto con Visual Studio .Net y trabajando en el diseño del primer formulario, tenemos que obtener algo así:



Pero a diferencia de lo visto hasta acá, este formulario debe modificar su contenido de acuerdo a la cantidad de asientos del transporte seleccionado en el control ComboBox, por lo tanto, no podemos dejar fijos los asientos, deberemos recurrir a una configuración por código para crear y distribuir la cantidad de asientos que se requiera en cada caso, nos referimos a construir el contenido del formulario de forma dinámica.

El segundo tema a considerar es qué tipo de controles nos conviene usar para lograr un resultado correcto, como ya vimos en las herramientas de esta situación profesional, disponemos de varios paneles que podemos emplear como contenedores y seguramente se podrá encontrar una solución adecuada con diferentes combinaciones de paneles y controles.

Nuestra propuesta de solución se basa en un control de tipo **FlowLayoutPanel** como contenedor principal, que contenga dos controles de tipo **TableLayoutPanel** y otro de tipo **Panel** como separador de los anteriores. Algo así:



El recuadro de color **rojo** corresponde al control **FlowLayoutPanel**.

Los recuadros de color **azul** son controles de tipo **TableLayoutPanel**, configurados con una sola columna y dos filas cada uno.

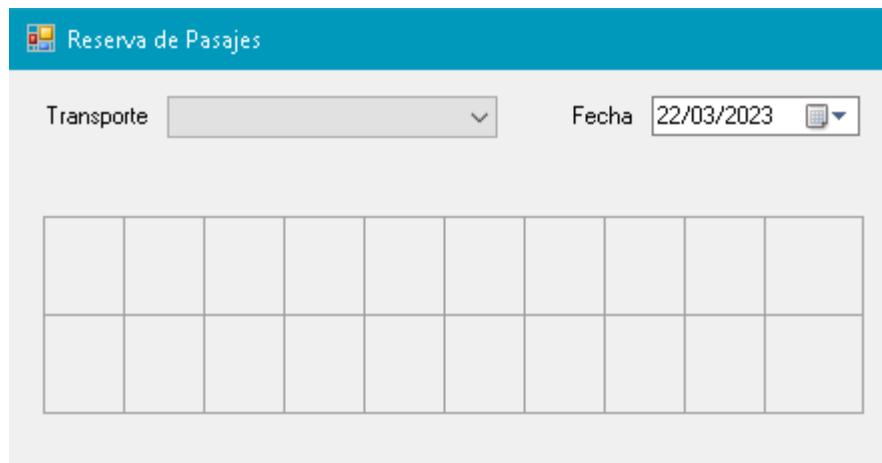
El recuadro de color **verde** es un control de tipo **Panel**, que usaremos solamente como separador de los otros dos controles.

En cada **TableLayoutPanel** deberemos establecer la cantidad de columnas necesarias de acuerdo a la cantidad de asientos que tengamos que mostrar, eso lo hacemos con la propiedad “**ColumnCount**” y “**ColumnStyles**”, por ejemplo, si queremos 10 lugares, en cada fila del panel:

```
tblPnlAsientosSup.ColumnCount = 10;
tblPnlAsientosSup.ColumnStyles.Clear();
for (int i = 0; i < 10; i++)
{
    tblPnlAsientosSup.ColumnStyles.Add(new ColumnStyle(SizeType.Percent,
        (float)10));
}
```

El estilo de las columnas es: "sizeType.Percent" (porcentaje) y el valor es de 10, o sea cada columna tendrá un 10% del ancho del TableLayoutPanel.

Resulta de esta forma:



De esa forma ya podemos cambiar la configuración de los controles TableLayoutPanel a voluntad, la cantidad total de asientos del transporte es un dato que podemos recuperar de la base de datos sin mayores problemas ya que tendremos seleccionado en el control ComboBox el transporte a consultar.

Ahora tenemos que resolver cómo formamos el contenido de cada celda que se muestra en el TableLayoutPanel, en cada celda vemos un ícono que representa el asiento y un texto con el número de asiento:



Para manejar esto de forma segura usaremos 3 componentes:

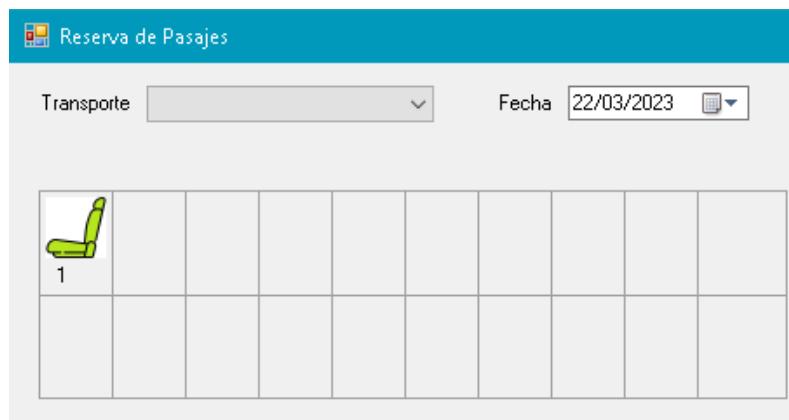
- Un control de tipo Panel como contenedor principal (recuadro de color rojo).
- Un control PictureBox para mostrar el ícono o imagen del asiento (recuadro azul)
- Un control Label para escribir el número de asiento (recuadro verde)

Estos 3 elementos se van a crear en el código y luego se insertarán en la celda correspondiente del TableLayoutPanel, simplificando un poco, sería así:

```
// panel contenedor para poner en una celda
Panel panelAsiento = new Panel();

int nro = 1; // número que tendrá el asiento
// crear el PictureBox
PictureBox pic = new PictureBox();
// agregar la imagen desde un archivo
pic.Image = Image.FromFile("asiento_verde.png");
// agregar el pictureBox al panel
panelAsiento.Controls.Add(pic);
// crear el Label
Label lbl = new Label();
// asignar el texto a mostrar
lbl.Text = nro.ToString();
// agregar el Label al panel
panelAsiento.Controls.Add(lbl);
// agregar el panel al TableLayoutPanel en la fila 0, columna 0
tblPnlAsientosSup.Controls.Add(panelAsiento, 0, 0);
```

Y obtenemos el primer asiento con su ícono y número colocados en la celda:



Este procedimiento lo podemos incluir en un ciclo repetitivo que itere la cantidad de veces correspondiente a la cantidad de asientos y tendremos el panel cargado en todas sus celdas.

Evento sobre el PictureBox

Cada asiento se debe poder seleccionar para luego ser reservado a nombre de una persona, para ello al hacer “Clic” sobre el asiento se deberá cambiar el ícono de color verde por otro igual, pero de color amarillo y así indicar que el asiento se quiere reservar.

Esta funcionalidad implica que cada PictureBox que agregamos al panel debe ser capaz de procesar el evento “Click” y por lo tanto ese evento lo debemos agregar también por código en el momento de crear cada uno de los controles PictureBox.

Modificamos el código visto en el punto anterior para agregar el evento “Click”:

```
// panel contenedor para poner en una celda
Panel panelAsiento = new Panel();

int nro = 1; // número que tendrá el asiento
// crear el PictureBox
PictureBox pic = new PictureBox();
// agregar la imagen desde un archivo
pic.Image = Image.FromFile("../Iconos/asiento_verde.png");
pic.Click += Pic_Click; // agregar el controlador del evento Click
// agregar el pictureBox al panel
panelAsiento.Controls.Add(pic);
// crear el Label
Label lbl = new Label();
// asignar el texto a mostrar
lbl.Text = nro.ToString();
// agregar el Label al panel
panelAsiento.Controls.Add(lbl);
// agregar el panel al TableLayoutPanel en la fila 0, columna 0
tblPnlAsientosSup.Controls.Add(panelAsiento, 0, 0);
```

En el momento que agreguemos esta línea de código (`pic.Click += Pic_Click;`), el editor va a crear el bloque con el evento para que completamos su funcionalidad, nos mostrará algo así:

```
private void Pic_Click(object sender, EventArgs e)
{
    throw new NotImplementedException();
}
```

Observe que contiene una línea donde se dispara una excepción de tipo “`NotImplementedException`”, a modo de recordatorio de que el código del evento todavía no ha sido implementado por nosotros.

Deberemos entonces eliminar esa línea de código y escribir nuestro procedimiento, en este caso las acciones a realizar serán modificar la imagen que contiene el PictureBox sobre el que se disparó el evento, si es un asiento de color verde lo cambiaremos por otro de color amarillo y si es de color amarillo lo cambiaremos por otro de color verde, como ayuda para identificar el estado de la imagen podemos usar la propiedad “Tag” de la imagen, asignando las letras “V” (Verde) y “A” (Amarillo) según el caso.

Lo más importante en la implementación de este evento, es el hecho que será el mismo para todos los asientos del formulario, es decir que la acción del “clic” sobre cualquier control PictureBox no llevará siempre a este único evento, por lo tanto, debemos encontrar la forma de determinar de qué objeto proviene el evento, de otra manera no sabríamos a cuál de todos los controles PictureBox tenemos que cambiar la imagen.

Esto lo solucionamos usando el parámetro “**sender**” que recibe el evento en el momento de ser ejecutado, “**sender**” es una referencia al objeto que generó el evento. El código del evento resulta así:

```
private void Pic_Click(object sender, EventArgs e)
{
    PictureBox pic = (PictureBox)sender; // sender es un PictureBox
    String Tag = pic.Image.Tag.ToString(); // obtener el valor de Tag
    if (pic.Image.Tag.ToString() == "V")
    {
        // si es verde se cambia por amarillo
        pic.Image = Image.FromFile("asiento_amarillo.png");
        pic.Image.Tag = "A";
    }
    else //si es amarillo se cambia por verde
    {
        pic.Image = Image.FromFile("asiento_verde.png");
        pic.Image.Tag = "V";
    }
}
```

Una vez completada la creación de los paneles con los asientos habrá que marcar los asientos que ya han sido reservados de color rojo, para informar al usuario que solamente podrá seleccionar asientos libres, la información de las reservas está en la base de datos por lo que necesitamos un procedimiento que consulte las reservas para el transporte y la fecha que se haya seleccionado en los controles de la interfaz. Una vez conseguida la lista de números de asientos reservados se procederá a actualizar las imágenes previamente cargadas por otras de color rojo, y además de dejarán deshabilitados esos PictureBox para que ya no se puedan volver a modificar.

El código, algo simplificado, de ese proceso sería como el siguiente:

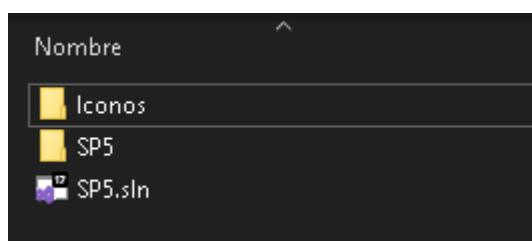
El parámetro de tipo entero: “**asiento**” es el número de asiento que obtenemos de la consulta sobre la base de datos y que ya está reservado. El procedimiento “**ActualizarEstado**” se encarga de cambiar la imagen por un asiento rojo, su Tag y de deshabilitarlas.

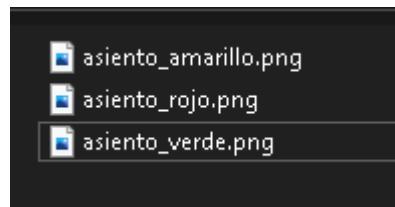
```
private void ActualizarEstado(int asiento)
{
    // se recorren todos los controles del TableLayoutPanel
    foreach (Control c in tableLayoutPanelAsientos.Controls)
    {
        // c.Controls[1] es el control Label que tiene el número de asiento
        if (c.Controls[1].Text == asiento.ToString())
        {
            // c.Controls[0] es el PictureBox
            PictureBox pic = (PictureBox)c.Controls[0];
            // se cambia la imagen
            pic.Image = Image.FromFile("asiento_rojo.png");
            // se cambia el Tag
            pic.Image.Tag = "R";
            // y se deshabilita
            pic.Enabled = false;
        }
    }
}
```

El procedimiento “ActualizarEstado” se ejecutará tantas veces como asientos reservados existan para el transporte y fecha consultados.

Todos estos procedimientos están relacionados con la presentación de datos en el estado inicial del formulario y la posible interacción por parte del usuario para seleccionar asientos libres en los paneles, la implementación final de todo esto la veremos más adelante ya que previamente debemos tener definidas las clases que manejan los datos de la aplicación y que interactúan con la base de datos.

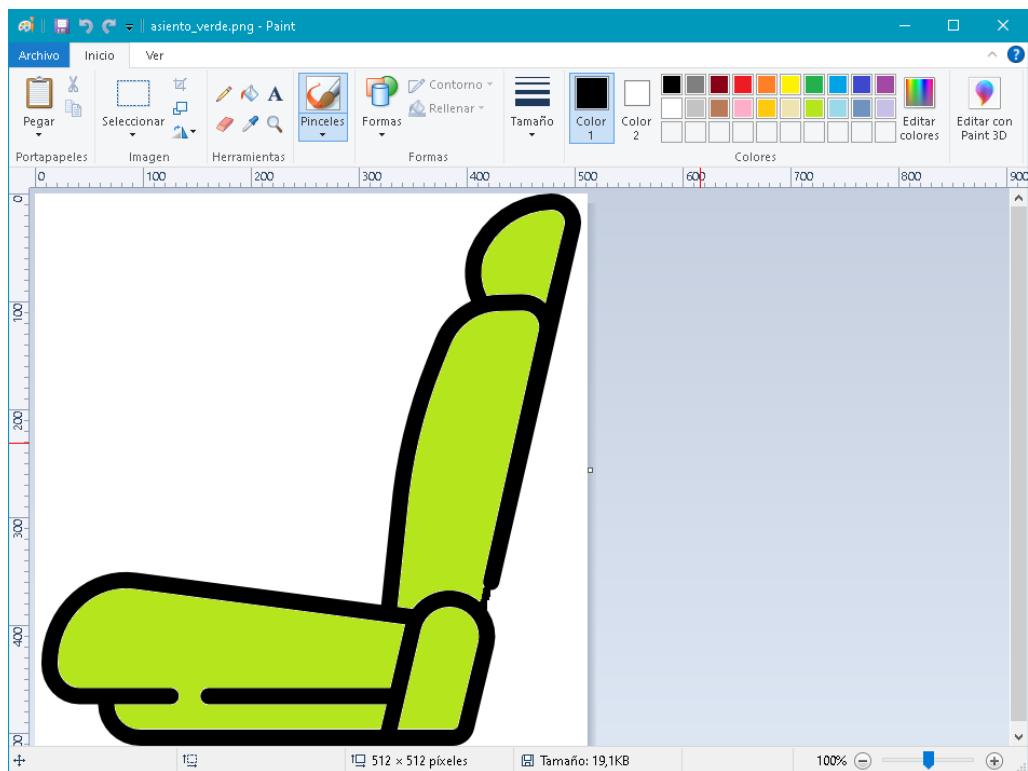
Como parte del proyecto vamos a incluir una carpeta que contenga las imágenes necesarias para los asientos, serán 3 archivos: “asiento_verde.png”, “asiento_amarillo.png” y “asiento_rojo.png”, la carpeta que los contiene se llama “Iconos” y se ubicará a nivel de la raíz del proyecto:





Cuando necesitemos acceder a estas imágenes debemos tener en cuenta su ubicación relativa al ejecutable de la aplicación.

A modo de ejemplo se muestra el diseño de la imagen usada para esta resolución editada desde la aplicación “Paint”:

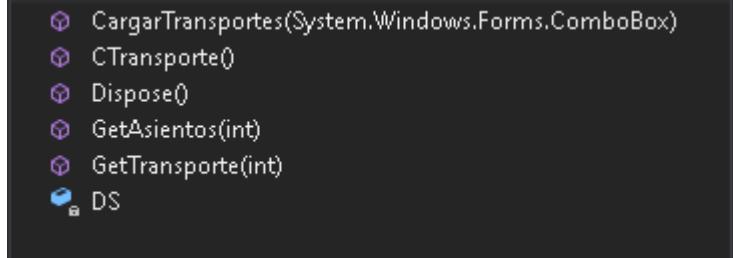


El tamaño de la imagen es de 512 píxeles x 512 píxeles y se puede guardar como archivo bmp, jpg o png.

Comenzaremos con la implementación de la clase relacionada con los transportes.

Clase CTransporte

Tendrá estos métodos



El método **constructor** se encarga de la conexión con la base de datos y de obtener los datos de la tabla Transportes, además define la clave primaria para facilitar las búsquedas.

- El método “**CargarTransportes**” rellena los datos de los transportes en un control ComboBox.
- El método “**GetAsientos**” obtiene la capacidad de asientos que posee un transporte.
- El método “**GetTransporte**” devuelve la descripción del transporte.
- El método “**Dispose**” libera los recursos usados por el DataSet.

El código completo de la clase CTransporte resulta así:

```
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 SP5
{
    public class CTransporte
    {
        private DataSet DS;

        public CTransporte() // constructor
        {
            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=Transporte.mdb";
                cnn.Open();

                // Proceso de la tabla Transportes
            }
        }
    }
}
```

```

        OleDbCommand cmdTr = new OleDbCommand();
        cmdTr.Connection = cnn;
        cmdTr.CommandType = CommandType.TableDirect;
        cmdTr.CommandText = "Transportes";

        OleDbDataAdapter daTr = new OleDbDataAdapter(cmdTr);
        daTr.Fill(DS, "Transportes");
        // se agrega la clave primaria
        DataColumn[] dcT = new DataColumn[1];
        dcT[0] = DS.Tables["Transportes"].Columns["Transporte"];
        DS.Tables["Transportes"].PrimaryKey = dcT;

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

// Obtiene la descripción del transporte pasado por parámetro
public String GetTransporte(int Transporte)
{
    String Descripcion = "";
    DataRow t = DS.Tables["Transportes"].Rows.Find(Transporte);
    if(t != null)
    {
        Descripcion = t["Descripcion"].ToString();
    }
    return Descripcion;
}

// carga los datos de los transportes en el ComboBox
public void CargarTransportes(ComboBox cmb)
{
    // rellena un ComboBox con los nombres de los transportes
    cmb.Items.Clear();
    cmb.DisplayMember = "Descripcion";
    cmb.ValueMember = "Transporte";
    cmb.DataSource = DS.Tables["Transportes"];
}

// devuelve la cantidad total de asientos del transporte
public int GetAsientos(int transporte)
{
    int Asientos = 0;
    try
    {
        DataRow tr = DS.Tables["Transportes"].Rows.Find(transporte);
        if(tr == null)
        {
            throw new Exception("CTransporte: El transporte no existe");
        }
        Asientos = (int)tr["Asientos"];
    }
    catch(Exception ex)
    {
        throw new Exception("CTransporte: " + ex.Message);
    }
    return Asientos;
}

```

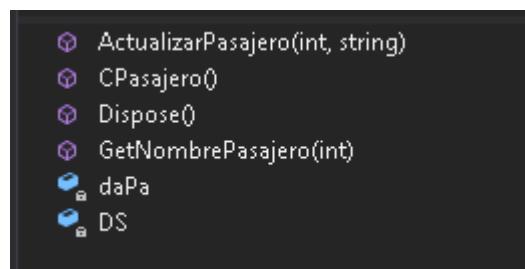
```

        // libera el DataSet
        public void Dispose()
        {
            DS.Dispose();
        }
    }
}

```

Continuamos con la clase que trabaja con los datos de los pasajeros

Clase CPasajero



El método **constructor** se encarga de la conexión con la base de datos y de obtener los datos de la tabla Pasajeros, además define la clave primaria para facilitar las búsquedas.

- El método “ActualizarPasajero” graba en la tabla el DNI y Nombre de un pasajero.
- El método “GetNombrePasajero” obtiene el nombre del pasajero cuyo DNI se pasa por parámetro.
- El método “Dispose” libera los recursos usados por el DataSet.

Código completo de la clase CPasajero

```

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 SP5
{
    public class CPasajero
    {

```

```

private DataSet DS;
private OleDbDataAdapter daPa;

public CPasajero()
{
    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=Transporte.mdb";
        cnn.Open();

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

        daPa = new OleDbDataAdapter(cmdPa);
        daPa.Fill(DS, "Pasajeros");
        // se agrega la clave primaria
        DataColumn[] dcPa = new DataColumn[1];
        dcPa[0] = DS.Tables["Pasajeros"].Columns["Pasajero"];
        DS.Tables["Pasajeros"].PrimaryKey = dcPa;
        //
        OleDbCommandBuilder cbpa = new OleDbCommandBuilder(daPa);

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

// devuelve el Nombre de un pasajero
public String GetNombrePasajero(int pasajero)
{
    String Nombre = "";
    DataRow p = DS.Tables["Pasajeros"].Rows.Find(pasajero);
    if(p != null)
    {
        Nombre = p["Nombre"].ToString();
    }
    return Nombre;
}

// Agrega un registro nuevo en la tabla de Pasajeros
public void ActualizarPasajero(int pasajero, String nombre)
{
    DataRow p = DS.Tables["Pasajeros"].Rows.Find(pasajero);
    if (p == null)
    {
        try
        {
            // agregar
            DataRow nuevo = DS.Tables["Pasajeros"].NewRow();
            nuevo["Pasajero"] = pasajero;
            nuevo["Nombre"] = nombre;
            DS.Tables["Pasajeros"].Rows.Add(nuevo);
        }
    }
}

```

```

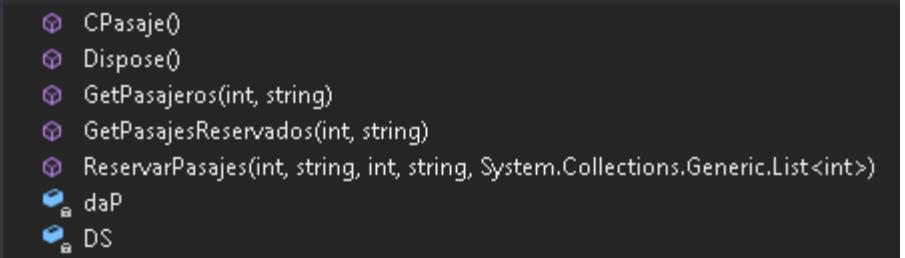
        daPa.Update(DS, "Pasajeros");
    }
    catch(Exception ex)
    {
        throw new Exception("CPasajero: " + ex.Message);
    }
}

// Libera los recursos del DataSet
public void Dispose()
{
    DS.Dispose();
}
}
}

```

Clase para el manejo de la tabla de Pasajes

Clase CPasaje



```

    CPasaje()
    Dispose()
    GetPasajeros(int, string)
    GetPasajesReservados(int, string)
    ReservarPasajes(int, string, int, string, System.Collections.Generic.List<int>)
    daP
    DS

```

- El método **constructor** se encarga de la conexión con la base de datos y de obtener los datos de la tabla Pasajes, además define la clave primaria para facilitar las búsquedas.
- El método “**GetPasajeros**” devuelve una lista de objetos Pasajeros para un transporte y una fecha determinada.
- El método “**GetPasajesReservados**” obtiene una lista con los números de asiento reservados para un transporte y una fecha determinada.
- El método “**ReservarPasajes**” permite agregar un registro nuevo a la tabla de Pasajes por cada número de asiento de un determinado transporte y fecha, también actualiza los datos del pasajero que está haciendo la reserva.
- El método “**Dispose**” libera los recursos usados por el DataSet.

La implementación completa de la clase CPasaje es esta:

```
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 SP5
{
    public class CPasaje
    {
        private DataSet DS;
        private OleDbDataAdapter daP;

        public CPasaje() // constructor
        {
            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=Transporte.mdb";
                cnn.Open();

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

                daP = new OleDbDataAdapter(cmdP);
                daP.Fill(DS, "Pasajes");
                // se agrega la clave primaria
                DataColumn[] dcP = new DataColumn[3];
                dcP[0] = DS.Tables["Pasajes"].Columns["Transporte"];
                dcP[1] = DS.Tables["Pasajes"].Columns["Fecha"];
                dcP[2] = DS.Tables["Pasajes"].Columns["Asiento"];
                DS.Tables["Pasajes"].PrimaryKey = dcP;
                //
                OleDbCommandBuilder cbp = new OleDbCommandBuilder(daP);

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

        // Graba en la tabla de Pasajes los registros para los asientos reservados
        public void ReservarPasajes(int transporte, String fecha,
            int pasajero, String nombre, List<int> asientos)
        {
            try
            {
                foreach (int asiento in asientos)
                {

```

```

        DataRow nuevo = DS.Tables["Pasajes"].NewRow();
        nuevo["Transporte"] = transporte;
        nuevo["Fecha"] = fecha;
        nuevo["Asiento"] = asiento;
        nuevo["Pasajero"] = pasajero;
        DS.Tables["Pasajes"].Rows.Add(nuevo);
    }
    daP.Update(DS, "Pasajes");
    //
    CPasajero pa = new CPasajero();
    pa.ActualizarPasajero(pasajero, nombre);
    pa.Dispose();
}
catch(Exception ex)
{
    throw new Exception("CPasajes: " + ex.Message);
}
}

public List<int> GetPasajesReservados(int transporte, String fecha)
{
    List<int> asientos = new List<int>();
    foreach(DataRow dr in DS.Tables["Pasajes"].Rows)
    {
        if(transporte == (int)dr["Transporte"] && fecha ==
dr["Fecha"].ToString())
        {
            asientos.Add((int)dr["Asiento"]);
        }
    }

    return asientos;
}

// devuelve la lista de Reservas para el transporte y fecha pasados por
parámetro
public List<CReserva> GetPasajeros(int transporte, String fecha)
{
    List<CReserva> reservas = new List<CReserva>();
    CPasajero pasajero = new CPasajero();
    // se recorre la tabla de Pasajes
    foreach (DataRow dr in DS.Tables["Pasajes"].Rows)
    {
        // se buscan los pasajes del transporte y fecha recibidos
        if (transporte == (int)dr["Transporte"] && fecha ==
dr["Fecha"].ToString())
        {
            // se obtiene el nombre del pasajero que realizó la reserva
            String Nombre =
pasajero.GetNombrePasajero((int)dr["Pasajero"]);
            // se crea el objeto Reserva
            CReserva reserva = new CReserva((int)dr["Asiento"], Nombre);
            // se agrega a la lista
            reservas.Add(reserva);
        }
    }
    pasajero.Dispose();
    return reservas; // resultado con todas las reservas existentes
}

// Liberar los recursos del DataSet

```

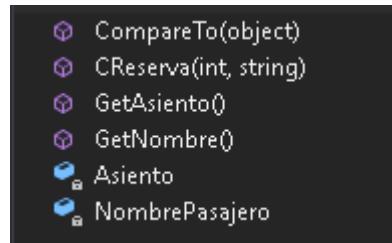
```

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

```

Agregamos también dos clases auxiliares para facilitar el manejo de los datos que no están en la base de datos y para manejar la impresión solicitada en la situación profesional.

La primera clase auxiliar se denomina “**CReserva**” y contiene solamente los datos que forman una reserva, que son el nombre de la persona que realiza la reserva y el número de asiento que está reservando. También agregamos un método para poder comparar dos objetos de esta clase por número de asiento ya que posteriormente necesitaremos tener la información ordenada por ese atributo cuando se genere la impresión.



Clase CReserva:

```

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

namespace SP5
{
    public class CReserva: IComparable // hereda de IComparable
    {
        private int Asiento;
        private String NombrePasajero;

        // constructor de la clase
        public CReserva(int asiento, String nombre)
        {
            Asiento = asiento;
            NombrePasajero = nombre;
        }

        public int GetAsiento()
    }
}

```

```

    {
        return Asiento;
    }

    public String GetNombre()
    {
        return NombrePasajero;
    }

    // método de comparación entre dos objetos CReserva
    // es la implementación de la interfaz IComparable
    public int CompareTo(object obj)
    {
        CReserva a = obj as CReserva;
        // si el objeto actual es mayor al comparado
        if (this.Asiento > a.Asiento)
            return 1; // devuelve positivo
        // si el objeto actual es menor al comparado
        if (this.Asiento < a.Asiento)
            return -1; // devuelve negativo
        return 0; // si son iguales devuelve cero
    }
}

```

El detalle a considerar en esta clase es que la misma hereda de “**IComparable**”, “**IComparable**” es una interfaz que define el método llamado “**CompareTo**”, recibe un objeto que será comparado con la instancia actual de la clase y que siempre devuelve un valor entero con esta lógica:

- Si ambos valores comparados son iguales devuelve cero
- Si el valor del objeto actual es mayor al valor del objeto recibido por parámetro, devuelve un valor mayor a cero.
- Si el valor del objeto actual es menor al valor del objeto recibido por parámetro, devuelve un valor menor a cero.

Tener este método implementado nos habilita a usar el método “**Sort**” sobre cualquier colección de objetos de tipo “CReserva” y así obtener fácilmente esa colección ordenada por el campo que definimos en “**CompareTo**”.

Finalmente nos queda por revisar la clase encargada de la impresión del listado de reservas.

Clase ClImpresion

```
    Ⓛ ClImpresion(System.Drawing.Printing.PrintPageEventArgs, int, string, System.)
    Ⓛ Dispose()
    Ⓛ ImprimirEncabezado(int, string)
    Ⓛ ImprimirPanel(System.Windows.Forms.TableLayoutPanel, int, int)
    Ⓛ ImprimirPasajeros(int, string)
    Ⓛ ImprimirPiePagina()
    Ⓛ ImprimirReservas()
    Ⓜ e
    Ⓜ Fecha
    Ⓜ FontDatos
    Ⓜ FontSubtitulo
    Ⓜ FontTitulo
    Ⓜ GRAF
    Ⓜ PanelInferior
    Ⓜ PanelSuperior
    Ⓜ PGSET
    Ⓜ Transporte
```

Esta clase será usada desde el evento **PrintPage** de nuestro documento a imprimir y necesitará varios parámetros en su constructor ya que además del parámetro de tipo “**PrintPageEventArgs**” necesario para acceder al objeto “Graphics”, también serán necesarios el transporte, la fecha del viaje y los dos paneles que contienen los asientos.

En el constructor también se crearán los objetos de tipo “Font” con distintos tamaños de letra para usar en los títulos, subtítulos y texto con datos en el contenido de la página a imprimir.

El método “**ImprimirReservas**” es el principal método de la clase y se encarga de gestionar todo el proceso de impresión invocando a los demás métodos para generar los datos de la cabecera, los paneles y el pie de página.

Observar que en el método “**ImprimirReservas**” finaliza con:

```
e.HasMorePages = false;
```

Esa asignación indica que no hay más páginas para imprimir y desencadena el envío de datos a la impresora, o al diálogo de pre-visualización, como ocurre en este caso particular.

El método “**ImprimirEncabezado**” genera la impresión del título de la página, el nombre del transporte y la fecha del viaje que se está consultando.

El método “**ImprimirPanel**” construye un gráfico de tipo Bitmap a partir del contenido del panel (TableLayoutPanel) generando una imagen que resulta una copia de la distribución de los asientos tal como se ve en el formulario, este método será ejecutado una vez para el panel superior y otra vez para el panel inferior, cambiando los valores de las coordenadas X, Y.

El método “**ImprimirPasajeros**” generará el listado de asientos y pasajeros que realizaron las reservas, los datos se muestran ordenados por número de asiento y se distribuyen en dos zonas, a la izquierda las primeras 28 reservas y a la derecha de la página las restantes reservas. Cada zona lleva un encabezado con los nombres de las columnas: “Asiento Nro” y “Nombre Pasajero”.

El método “**ImprimirPiePagina**” imprimirá el número de la página, y más abajo la fecha y la hora en que se generó la impresión.

Código completo de la clase “**CImpresion**”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Drawing.Printing;
using System.Windows.Forms;

namespace SP5
{
    public class CImpresion
    {
        private Graphics GRAF;
        private PageSettings PGSET;
        private PrintPageEventArgs e;
```

```

private Font FontTitulo;
private Font FontSubtitulo;
private Font FontDatos;

private int Transporte;
private String Fecha;
private TableLayoutPanel PanelSuperior;
private TableLayoutPanel PanelInferior;

// Constructor
public CImpresion(PrintPageEventArgs e, int transporte, String fecha,
                    TableLayoutPanel panelSuperior, TableLayoutPanel panelInferior)
{
    this.e = e;
    GRAF = e.Graphics;
    PGSET = e.PageSettings;
    Transporte = transporte;
    Fecha = fecha;
    PanelSuperior = panelSuperior;
    PanelInferior = panelInferior;
    //
    FontTitulo = new Font("Arial", 14);
    FontSubtitulo = new Font("Arial", 12);
    FontDatos = new Font("Arial", 10);
}

// método principal que imprime la página completa de las reservas
public void ImprimirReservas()
{
    // imprimir el encabezado de la página
    ImprimirEncabezado(Transporte, Fecha);
    ImprimirPanel(PanelSuperior, 50, 120); // panel superior
    ImprimirPanel(PanelInferior, 50, 270); // panel inferior
    // números de asientos y nombres de pasajeros
    ImprimirPasajeros(Transporte, Fecha);
    ImprimirPiePagina();
    // fin de la impresión
    e.HasMorePages = false;
}

// método para imprimir el encabezado con el título
private void ImprimirEncabezado(int transporte, String fecha)
{
    CTransporte tra = new CTransporte();
    String Nombre = tra.GetTransporte(transporte); // obtiene el nombre del
transporte
    tra.Dispose();
    // imprime el título, nombre del transporte y fecha del viaje
    GRAF.DrawString("Listado de Asientos Reservados", FontTitulo,
Brushes.Black, 100, 30);
    GRAF.DrawString("Transporte: " + Nombre, FontSubtitulo, Brushes.Black,
20, 65);
    GRAF.DrawString("Fecha: " + fecha, FontSubtitulo, Brushes.Black, 20,
90);
    // imprime una línea horizontal de separación
    GRAF.DrawLine(Pens.Black, 10, 110, PGSET.PaperSize.Width - 10, 110);
}

// método para imprimir el gráfico de los paneles con los asientos
private void ImprimirPanel(TableLayoutPanel panel, int X, int Y)

```

```

    {
        // crea un objeto de tipo Bitmap para almacenar el gráfico
        Bitmap bmp = new Bitmap(panel.Width, panel.Height,
panel.CreateGraphics());
        // convierte el contenido del panel en un gráfico y lo guarda en el
Bitmap
        panel.DrawToBitmap(bmp, new Rectangle(0, 0, panel.Width,
panel.Height));
        // imprime el Bitmap
        GRAF.DrawImage(bmp, X, Y, panel.Width, panel.Height);
        bmp.Dispose(); // liberar recursos del Bitmap
    }

    // método para imprimir el listado de asientos y pasajeros
private void ImprimirPasajeros(int transporte, String fecha)
{
    int cantidad = 0;
    CPasaje pasajes = new CPasaje();
    List<CReserva> reservas = pasajes.GetPasajeros(transporte, fecha);
    // ordenar la lista por número de asiento
    reservas.Sort();

    int X = 50; // posición inicial para los datos
    int Y = 410;
    // imprimir títulos de las columnas
    GRAF.DrawString("Asiento Nro", FontSubtitulo, Brushes.Black, X, Y);
    GRAF.DrawString("Nombre Pasajero", FontSubtitulo, Brushes.Black, X +
100, Y);
    Y += 25;
    // se recorren todas las reservas
    foreach (CReserva reserva in reservas) {
        // de cada reserva se imprime el número de asiento y el nombre del
pasajero
        GRAF.DrawString(reserva.GetAsiento().ToString(), FontDatos,
Brushes.Black, X, Y);
        GRAF.DrawString(reserva.GetNombre(), FontDatos, Brushes.Black, X +
100, Y);
        Y += 20; // posición Y para la próxima fila
        cantidad++;
        if(cantidad == 28) // 28: máxima cantidad de filas en la hoja
        {
            X = 400; // se continúa imprimiendo en la parte derecha de
la hoja
            Y = 410;
            // se repiten los títulos de las columnas
            GRAF.DrawString("Asiento Nro", FontSubtitulo, Brushes.Black, X,
Y);
            GRAF.DrawString("Nombre Pasajero", FontSubtitulo,
Brushes.Black, X + 100, Y);
            Y += 25;
        }
    }
}

// método para imprimir el pie de página
private void ImprimirPiePagina()
{
    // imprime una línea horizontal de separación
    GRAF.DrawLine(Pens.Black, 10, PGSET.PrintableArea.Height - 80,
PGSET.PaperSize.Width - 10, PGSET.PrintableArea.Height - 80);
    // imprime el número de página

```

```

        GRAF.DrawString("página 1", FontDatos, Brushes.Black, 50,
PGSET.PrintableArea.Height - 70);
        // imprime la fecha y la hora actual
        String fecha = DateTime.Now.ToShortDateString() + ", " +
DateTime.Now.ToShortTimeString();
        GRAF.DrawString(fecha, FontDatos, Brushes.Black, 50,
PGSET.PrintableArea.Height - 50);
    }

    // dispose: libera todos los recursos de los objetos usados
public void Dispose()
{
    FontTitulo.Dispose();
    FontSubtitulo.Dispose();
    FontDatos.Dispose();
    GRAF.Dispose();
}
}
}
}

```

Finalizada la implementación de todas las clases de nuestra aplicación continuamos ahora con el código del primer formulario, veremos ahora con más detalle los métodos que ya anticipamos al comienzo de la resolución.

Código de Form1:

El evento “Load” del formulario: acá se usa un objeto de la clase CTransporte y se carga el comboBox con todos los transportes disponibles:

```

private void Form1_Load(object sender, EventArgs e)
{
    CTransporte transporte = new CTransporte();
    // carga el ComboBox con los transportes disponibles
    transporte.CargarTransportes(cmbTransportes);
    transporte.Dispose();
}

```

Eventos “SelectedIndexChanged” del control ComboBox y “ValueChanged” del control DateTimePicker, estos eventos se disparan automáticamente cuando el usuario selecciona un transporte o cambia la fecha del viaje, desde acá se ejecutará el proceso principal: “MostrarPasajes” encargado de construir los paneles con los asientos y marcar los que ya están reservados.

```

private void cmbTransportes_SelectedIndexChanged(object sender, EventArgs e)
{
    MostrarPasajes(); // si cambia el transporte seleccionado
}

private void dtpFecha_ValueChanged(object sender, EventArgs e)

```

```
{
    MostrarPasajes(); // si cambia la fecha del viaje
}
```

Método “**MostrarPasajes**”: se encarga de construir los paneles con los asientos del transporte seleccionado, para ello primero obtiene la cantidad total de asientos del transporte y con ese dato ejecuta el método “ConfigurarAsientos”, a continuación, obtiene la lista de reservas realizadas para ese viaje con el método “GetPasajesResrvados” y actualiza el estado de los asientos modificando la imagen por iconos de color rojo con el método “ActualizarEstado”

```
// método principal del formulario, se ejecuta cada vez que el usuario
// selecciona un transporte o cambia la fecha del viaje.
private void MostrarPasajes()
{
    CTransporte transporte = new CTransporte();
    // obtiene el transporte seleccionado en el ComboBox
    int Transporte = (int)cmbTransportes.SelectedValue;
    // obtiene la cantidad de asientos del transporte
    int asientos = transporte.GetAsientos(Transporte);
    // arma los paneles con la cantidad de asientos obtenida
    ConfigurarAsientos(asientos);
    CPasaje pasajes = new CPasaje();
    // busca la lista de reservas
    List<int> Reservados =
    pasajes.GetPasajesReservados((int)cmbTransportes.SelectedValue,
        dtpFecha.Value.Date.ToShortDateString());
    foreach (int asiento in Reservados)
    {
        // actualizar el estado de los asientos reservados
        ActualizarEstado(asiento);
    }
    pasajes.Dispose();
    transporte.Dispose();
}
```

Método “**ConfigurarAsientos**”: este método es el que construye el contenido de los paneles creando los objetos PictureBox y Label para cada asiento y al completar los dos paneles los ubica de forma centrada con respecto al ancho del formulario

```
// configura los paneles con los asientos necesarios
private void ConfigurarAsientos(int asientos)
{
    asientos = asientos / 4;
    int filas = 2;
    //
    flwPnlTransporte.Visible = false;
    ConfigurarPanel(tblPnlAsientosSup, asientos, filas, 0);
    ConfigurarPanel(tblPnlAsientosInf, asientos, filas, 2);

    // centrar el panel en el formulario
    flwPnlTransporte.Width = tblPnlAsientosSup.Width + 20;
```

```

        flwPnlTransporte.Location = new Point((int)(this.Width -
flwPnlTransporte.Width) / 2, flwPnlTransporte.Location.Y);
        flwPnlTransporte.Visible = true;
    }
}

```

La cantidad total de asientos se divide en 4 porque esa es la cantidad de asientos por cada fila, dos asientos en el panel superior y dos asientos en el panel inferior, además cada panel tiene 2 filas.

Método “ConfigurarPanel”: acá se crea un panel en particular, en base a la cantidad de asientos, filas y un último parámetro, denominado “offset” que sirve para diferenciar si se trata del panel superior o del panel inferior y se usa para determinar el número de asiento a colocar.

```

// configura el contenido completo de un panel
private void ConfigurarPanel(TableLayoutPanel panel, int asientos, int filas, int
offset)
{
    panel.Visible = false;
    panel.Controls.Clear(); // limpiar el contenido previo del panel
    // configurar el tableLayoutPanel
    // el tamaño depende de la cantidad de asientos
    panel.Size = new System.Drawing.Size((35 + 8) * asientos, (35 + 25) * filas);
    panel.ColumnCount = asientos; // cantidad de columnas del panel
    panel.RowCount = 2; // cantidad de filas
    panel.CellBorderStyle = TableLayoutPanelCellBorderStyle.Single;
    panel.ColumnStyles.Clear(); // se crea el estilo para cada columna de la tabla
    for (int i = 0; i < asientos; i++)
    {
        panel.ColumnStyles.Add(new ColumnStyle(SizeType.Percent, (float)asientos));
    }
    // asientos de tableLayoutPanel
    for (int f = 0; f < asientos; f++) // se recorren las filas y columnas
    {
        for (int c = 0; c < 2; c++)
        {
            // en cada celda se agrega un panel que contendrá un PictureBox y un
Label
            Panel panelAsiento = new Panel(); // se crea el panel contendor
            panelAsiento.AutoSize = true;
            panelAsiento.Dock = DockStyle.Fill;

            int nro = (f * 4) + (c + 1) + offset; // se determina el número del
asiento
            PictureBox pic = new PictureBox(); // se crea el PictureBox
            pic.Image = Image.FromFile("../Iconos/asiento_verde.png");
            pic.SizeMode = PictureBoxSizeMode.StretchImage;
            pic.BorderStyle = BorderStyle.None;
            pic.Image.Tag = "V"; // color Verde
            pic.Visible = true;
            pic.Location = new Point(0, 0);
            pic.Size = new Size(35, 35);
            pic.Click += PictureBox_Click; // se agrega el evento Click
        }
    }
}

```

```

        //
        panelAsiento.Controls.Add(pic); // se inserta el PictureBox en el panel
        //
        Label lbl = new Label(); // se crea el Label
        lbl.Text = nro.ToString();
        lbl.AutoSize = false;
        lbl.Location = new Point(3, 38);

        panelAsiento.Width = pic.Width;
        panelAsiento.Height = pic.Height + lbl.Height;
        panelAsiento.Controls.Add(lbl); // se inserta el Label en el panel
        panel.Controls.Add(panelAsiento, f, c); // se inserta el panel en la
celda
    }

}
panel.Visible = true;
}

```

Si comparamos con el proceso comentado al comienzo de la resolución vemos que hemos agregado más código para establecer algunas propiedades del Panel, del PictureBox y del Label, por ejemplo, para fijar ancho, alto y posición que debe adoptar cada elemento más algunas otras propiedades complementarias para lograr una correcta ubicación y visualización de cada asiento dentro del TableLayoutPanel. Sin embargo, la lógica del proceso es la misma.

Método “ActualizarEstado”: recibe por parámetro el número de asiento que debe actualizar, el proceso recorre los controles de cada panel hasta encontrar el que posee un Label con el mismo número de asiento, recordemos que cada celda del TableLayoutPanel contiene a su vez un panel simple con un PictureBox y un Label, el número de asiento está almacenado en el Label, y una vez localizado hay que modificar la imagen del PictureBox que está en el mismo panel que el Label, se asignará una imagen de asiento color rojo.

```

// el parámetro de método es un número de asiento reservado
// se debe ubicar en el panel y modificar el icono del pictureBox
// por un ícono de color rojo
private void ActualizarEstado(int asiento)
{
    // recorrer los controles del panel
    foreach(Control c in tblPnlAsientosSup.Controls)
    {
        // es el número de asiento ? Controls[1] es el Label
        if(c.Controls[1].Text == asiento.ToString())
        {
            PictureBox pic = (PictureBox)c.Controls[0];
            // cambia la imagen
            pic.Image = Image.FromFile("../Iconos/asiento_rojo.png");
            pic.Image.Tag = "R"; // cambia el Tag
        }
    }
}

```

```

        pic.BorderStyle = BorderStyle.FixedSingle; // se agrega el borde
        pic.Enabled = false; // se deshabilita
        break; // salir del recorrido
    }
}
// se repite lo mismo para el segundo panel
foreach (Control c in tblPnlAsientosInf.Controls)
{
    if (c.Controls[1].Text == asiento.ToString())
    {
        PictureBox pic = (PictureBox)c.Controls[0];
        pic.Image = Image.FromFile("../Iconos/asiento_rojo.png");
        pic.Image.Tag = "R";
        pic.BorderStyle = BorderStyle.FixedSingle;
        pic.Enabled = false;
        break;
    }
}
}

```

Observa que además de modificar la imagen, se asigna la letra “R” (Rojo) a la propiedad Tag, se agrega un borde simple a la imagen y lo más importante, el pictureBox se deshabilita para que el usuario no pueda volver a reservar el asiento.

Evento Click del PictureBox: en los controles PictureBox que estén habilitados el usuario podrá hacer clic con el botón del mouse cambiando el color de verde a amarillo o inversamente si el asiento ya está en amarillo, estos cambios son controlados por el código en el evento Click:

```

// cambia la imagen del PictureBox
private void PictureBox_Click(object sender, EventArgs e)
{
    PictureBox pic = (PictureBox)sender; // obtiene el objeto del evento Click
    String Tag = pic.Image.Tag.ToString(); // valor de la propiedad Tag
    if (Tag == "V") // es Verde?
    {
        pic.Image = Image.FromFile("../Iconos/asiento_verde.png");
        pic.Image.Tag = "A"; // Amarillo
    }
    else // es Amarillo
    {
        pic.Image = Image.FromFile("../Iconos/asiento_amarillo.png");
        pic.Image.Tag = "V"; // Verde
    }
}

```

Lo importante acá es el uso del parámetro llamado “sender” que contiene una referencia al objeto que genera el evento, en este caso un control PictureBox, a

partir de ese parámetro se puede acceder a las propiedades del control para modificar sus valores.

Evento Click del botón Reservar: en este evento se construye una lista de enteros con los números de asiento que estén seleccionados para reservar, es decir los que están con su imagen de color amarillo. Luego se invoca al formulario de confirmación donde se piden los datos de documento y nombre de la persona que quiere reservar, al Form2 se pasan por parámetro el transporte seleccionado, la fecha de viaje y la lista de asientos a reservar.

```
private void btnReservar_Click(object sender, EventArgs e)
{
    // buscar los asientos para reservar, (los que están en amarillo)
    List<int> asientos = new List<int>();

    // recorrer los controles del panel superior
    foreach (Panel panel in tblPnlAsientosSup.Controls)
    {
        PictureBox pic = (PictureBox)panel.Controls[0];
        // controlar si el Tag de la imagen es "A" Amarillo
        if (pic.Image.Tag.ToString() == "A")
        {
            // obtener el número de asiento
            int nro = int.Parse(panel.Controls[1].Text);
            asientos.Add(nro); // se agrega el número a la lista
        }
    }
    // recorrer los controles del panel inferior
    foreach (Panel panel in tblPnlAsientosInf.Controls)
    {
        PictureBox pic = (PictureBox)panel.Controls[0];
        if (pic.Image.Tag.ToString() == "A")
        {
            int nro = int.Parse(panel.Controls[1].Text);
            asientos.Add(nro);
        }
    }
    // si la lista contiene elementos
    if (asientos.Count > 0)
    {
        // invocar al formulario de confirmación de reservas
        Form2 frm = new Form2((int)cmbTransportes.SelectedValue,
            dtpFecha.Value.Date.ToShortDateString(),
            asientos);
        frm.ShowDialog();
        // actualizar los asientos reservados
        CPasaje pasajes = new CPasaje();
        List<int> Reservados =
            pasajes.GetPasajesReservados((int)cmbTransportes.SelectedValue,
                dtpFecha.Value.Date.ToShortDateString());
        foreach (int asiento in Reservados)
        {
            ActualizarEstado(asiento);
        }
        pasajes.Dispose();
    }
}
```

```

        }
    else
    {
        MessageBox.Show("Debe seleccionar uno o más asientos para reservar",
"Atención");
    }
}

```

Si no hay asientos marcados para reservar se muestra un mensaje de aviso.

Botón Imprimir y evento PrintPage: en el evento Click del botón Imprimir se crea un objeto de tipo PrintDocument, se asocia con el evento PrintPage, luego se establecen las propiedades básicas de la página. Seguidamente, se crea el diálogo de pre-visualización (PrintPreviewDialog), se asigna la propiedad “Document” y se ejecuta el método “ShowDialog” para visualizar el diálogo. La llamada a ese método dispara el evento PrintPage que vemos a continuación.

```

private void btnImprimir_Click(object sender, EventArgs e)
{
    PrintDocument pd = new PrintDocument(); // se crea el documento a imprimir
    pd.PrintPage += Pd_PrintPage; // se agrega el evento PrintPage
    // se fija la orientación del papel, la cantidad de copias y el tamaño del
    papel
    pd.PrinterSettings.DefaultPageSettings.Landscape = false;
    pd.PrinterSettings.Copies = 1;
    pd.PrinterSettings.DefaultPageSettings.PaperSize = new PaperSize("A4", 210,
297);
    // se crea el diálogo de Previsualización
    PrintPreviewDialog PrintDlg = new PrintPreviewDialog();
    PrintDlg.Document = pd;
    PrintDlg.ShowDialog(); // acá se dispara el evento PrintPage y se muestra la
    página
}

private void Pd_PrintPage(object sender, PrintPageEventArgs e)
{
    int Transporte = (int)cmbTransportes.SelectedValue;
    String Fecha = dtpFecha.Value.Date.ToShortDateString();
    // se construye el objeto CImpresion
    impresion = new CImpresion(e, Transporte, Fecha,
        tblPnlAsientosSup, tblPnlAsientosInf);
    impresion.ImprimirReservas(); // genera la página a imprimir
    impresion.Dispose();
}

```

En el evento PrintPage tomamos el transporte seleccionado en el ComboBox y la fecha del DateTimePicker y con esos valores se crea un objeto de la clase “CImpresion”, se pasan también los dos paneles de tipo TableLayoutPanel para poder imprimir su contenido. Una vez creado el objeto impresión, ejecutamos el método

“ImprimirReservas” encargado de construir todo el contenido de la página. Finalmente, con el método Dispose se liberan los recursos gráficos empleados en la impresión.

El código completo del formulario Form1 resulta así:

```
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.Drawing.Printing;
using System.Drawing.Configuration;

namespace SP5
{
    public partial class Form1 : Form
    {
        private CImpresion impresion;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            CTransporte transporte = new CTransporte();
            // carga el ComboBox con los transportes disponibles
            transporte.CargarTransportes(cmbTransportes);
            transporte.Dispose();
        }

        private void cmbTransportes_SelectedIndexChanged(object sender, EventArgs e)
        {
            MostrarPasajes(); // si cambia el transporte seleccionado
        }

        private void dtpFecha_ValueChanged(object sender, EventArgs e)
        {
            MostrarPasajes(); // si cambia la fecha del viaje
        }

        // método principal del formulario, se ejecuta cada vez que el usuario
        // selecciona un transporte o cambia la fecha del viaje.
        private void MostrarPasajes()
        {
            CTransporte transporte = new CTransporte();
            // obtiene el transporte seleccionado en el ComboBox
            int Transporte = (int)cmbTransportes.SelectedValue;
            // obtiene la cantidad de asientos del transporte
            int asientos = transporte.GetAsientos(Transporte);
```

```

        // arma los paneles con la cantidad de asientos obtenida
        ConfigurarAsientos(asientos);
        CPasaje pasajes = new CPasaje();
        // busca la lista de reservas
        List<int> Reservados =
    pasajes.GetPasajesReservados((int)cmbTransportes.SelectedValue,
        dtpFecha.Value.Date.ToShortDateString());
        foreach (int asiento in Reservados)
    {
        // actualizar el estado de los asientos reservados
        ActualizarEstado(asiento);
    }
    pasajes.Dispose();
    transporte.Dispose();
}

// configura los paneles con los asientos necesarios
private void ConfigurarAsientos(int asientos)
{
    asientos = asientos / 4;
    int filas = 2;
    //
    flwPnlTransporte.Visible = false;
    ConfigurarPanel(tblPnlAsientosSup, asientos, filas, 0);
    ConfigurarPanel(tblPnlAsientosInf, asientos, filas, 2);

    // centrar el panel en el formulario
    flwPnlTransporte.Width = tblPnlAsientosSup.Width + 20;
    flwPnlTransporte.Location = new Point((int)(this.Width -
flwPnlTransporte.Width) / 2, flwPnlTransporte.Location.Y);
    flwPnlTransporte.Visible = true;

}

// configura el contenido completo de un panel
private void ConfigurarPanel(TableLayoutPanel panel, int asientos, int
filas, int offset)
{
    panel.Visible = false;
    panel.Controls.Clear(); // limpiar el contenido previo del panel
    // configurar el tableLayoutPanel
    // el tamaño depende de la cantidad de asientos
    panel.Size = new System.Drawing.Size((35 + 8) * asientos, (35 + 25) *
filas);
    panel.ColumnCount = asientos; // cantidad de columnas del panel
    panel.RowCount = 2; // cantidad de filas
    panel.CellBorderStyle = TableLayoutPanelCellBorderStyle.Single;
    panel.ColumnStyles.Clear(); // se crea el estilo para cada columna de
la tabla
    for (int i = 0; i < asientos; i++)
    {
        panel.ColumnStyles.Add(new ColumnStyle(SizeType.Percent,
(float)asientos));
    }
    // asientos de tableLayoutPanel
    for (int f = 0; f < asientos; f++) // se recorren las filas y columnas
    {
        for (int c = 0; c < 2; c++)
        {

```

```

y un Label
    // en cada celda se agrega un panel que contendrá un PictureBox
    Panel panelAsiento = new Panel(); // se crea el panel contendor
    panelAsiento.AutoSize = true;
    panelAsiento.Dock = DockStyle.Fill;

    int nro = (f * 4) + (c + 1) + offset; // se determina el número
    del asiento
    PictureBox pic = new PictureBox(); // se crea el PictureBox
    pic.Image =
    Image.FromFile("../Iconos/asiento_verde.png");
    pic.SizeMode = PictureBoxSizeMode.StretchImage;
    pic.BorderStyle = BorderStyle.None;
    pic.Image.Tag = "V"; // color Verde
    pic.Visible = true;
    pic.Location = new Point(0, 0);
    pic.Size = new Size(35, 35);
    pic.Click += PictureBox_Click; // se agrega el evento Click
    //
    panelAsiento.Controls.Add(pic); // se inserta el PictureBox en
    el panel
    //
    Label lbl = new Label(); // se crea el Label
    lbl.Text = nro.ToString();
    lbl.AutoSize = false;
    lbl.Location = new Point(3, 38);

    panelAsiento.Width = pic.Width;
    panelAsiento.Height = pic.Height + lbl.Height;
    panelAsiento.Controls.Add(lbl); // se inserta el Label en el
    panel
    panel.Controls.Add(panelAsiento, f, c); // se inserta el panel
    en la celda
}

}
panel.Visible = true;
}

// el parámetro de método es un número de asiento reservado
// se debe ubicar en el panel y modificar el icono del pictureBox
// por un ícono de color rojo
private void ActualizarEstado(int asiento)
{
    // recorrer los controles del panel
    foreach(Control c in tblPnlAsientosSup.Controls)
    {
        // es el número de asiento ? Controls[1] es el Label
        if(c.Controls[1].Text == asiento.ToString())
        {
            PictureBox pic = (PictureBox)c.Controls[0];
            // cambia la imagen
            pic.Image = Image.FromFile("../Iconos/asiento_rojo.png");
            pic.Image.Tag = "R"; // cambia el Tag
            pic.BorderStyle = BorderStyle.FixedSingle; // se agrega el
            borde
            pic.Enabled = false; // se deshabilita
            break; // salir del recorrido
        }
    }
    // se repite lo mismo para el segundo panel
    foreach (Control c in tblPnlAsientosInf.Controls)

```

```

    {
        if (c.Controls[1].Text == asiento.ToString())
        {
            PictureBox pic = (PictureBox)c.Controls[0];
            pic.Image = Image.FromFile("../Iconos/asiento_rojo.png");
            pic.Image.Tag = "R";
            pic.BorderStyle = BorderStyle.FixedSingle;
            pic.Enabled = false;
            break;
        }
    }
}

// cambia la imagen del PictureBox
private void PictureBox_Click(object sender, EventArgs e)
{
    PictureBox pic = (PictureBox)sender; // obtiene el objeto del evento
Click
    String Tag = pic.Image.Tag.ToString(); // valor de la propiedad Tag
    if (Tag == "V") // es Verde?
    {
        pic.Image = Image.FromFile("../Iconos/asiento_amarillo.png");
        pic.Image.Tag = "A"; // Amarillo
    }
    else // es Amarillo
    {
        pic.Image = Image.FromFile("../Iconos/asiento_verde.png");
        pic.Image.Tag = "V"; // Verde
    }
}

private void btnReservar_Click(object sender, EventArgs e)
{
    // buscar los asientos para reservar, (los que están en amarillo)
    List<int> asientos = new List<int>();

    // recorrer los controles del panel superior
    foreach (Panel panel in tblPnlAsientosSup.Controls)
    {
        PictureBox pic = (PictureBox)panel.Controls[0];
        // controlar si el Tag de la imagen es "A" Amarillo
        if (pic.Image.Tag.ToString() == "A")
        {
            // obtener el número de asiento
            int nro = int.Parse(panel.Controls[1].Text);
            asientos.Add(nro); // se agrega el número a la lista
        }
    }
    // recorrer los controles del panel inferior
    foreach (Panel panel in tblPnlAsientosInf.Controls)
    {
        PictureBox pic = (PictureBox)panel.Controls[0];
        if (pic.Image.Tag.ToString() == "A")
        {
            int nro = int.Parse(panel.Controls[1].Text);
            asientos.Add(nro);
        }
    }
    // si la lista contiene elementos
    if (asientos.Count > 0)

```

```

    {
        // invocar al formulario de confirmación de reservas
        Form2 frm = new Form2((int)cmbTransportes.SelectedValue,
            dtpFecha.Value.Date.ToShortDateString(),
            asientos);
        frm.ShowDialog();
        // actualizar los asientos reservados
        CPasaje pasajes = new CPasaje();
        List<int> Reservados =
    pasajes.GetPasajesReservados((int)cmbTransportes.SelectedValue,
dtpFecha.Value.Date.ToShortDateString());
        foreach (int asiento in Reservados)
        {
            ActualizarEstado(asiento);
        }
        pasajes.Dispose();
    }
    else
    {
        MessageBox.Show("Debe seleccionar uno o más asientos para
reservar", "Atención");
    }
}

private void btnImprimir_Click(object sender, EventArgs e)
{
    PrintDocument pd = new PrintDocument(); // se crea el documento a
imprimir
    pd.PrintPage += Pd_PrintPage; // se agrega el evento PrintPage
    // se fija la orientación del papel, la cantidad de copias y el tamaño
del papel
    pd.PrinterSettings.DefaultPageSettings.Landscape = false;
    pd.PrinterSettings.Copies = 1;
    pd.PrinterSettings.DefaultPageSettings.PaperSize = new PaperSize("A4",
210, 297);
    // se crea el diálogo de Previsualización
    PrintPreviewDialog PrintDlg = new PrintPreviewDialog();
    PrintDlg.Document = pd;
    PrintDlg.ShowDialog(); // acá se dispara el evento PrintPage y se
muestra la página
}

private void Pd_PrintPage(object sender, PrintPageEventArgs e)
{
    int Transporte = (int)cmbTransportes.SelectedValue;
    String Fecha = dtpFecha.Value.Date.ToShortDateString();
    // se construye el objeto CImpresion
    impresion = new CImpresion(e, Transporte, Fecha,
        tblPnlAsientosSup, tblPnlAsientosInf);
    impresion.ImprimirReservas(); // genera la página a imprimir
    impresion.Dispose();
}

}
}

```

Continuamos ahora con el **código del segundo formulario: Form2**

Hemos agregado un segundo método **constructor** para poder con los valores del transporte, la fecha y la lista de asientos a reservar:

```
public Form2(int transporte, String fecha, List<int> asientos)
{
    InitializeComponent();
    Transporte = transporte;
    Fecha = fecha;
    Asientos = asientos;
    txtDni.MaxLength = 8;
    txtNombre.MaxLength = 30;
}
```

Evento Click del botón Confirmar:

```
private void btnConfirmar_Click(object sender, EventArgs e)
{
    if(txtDni.Text != "" && txtNombre.Text != "")
    {
        try
        {
            int transporte = Transporte;
            String fecha = Fecha;
            int pasajero = int.Parse(txtDni.Text);
            String nombre = txtNombre.Text;
            CPasaje pasajes = new CPasaje();
            // grabar las reservas
            pasajes.ReservarPasajes(transporte, fecha,
                                   pasajero, nombre, Asientos);
            MessageBox.Show("Reserva registrada correctamente", "Reservas");
            pasajes.Dispose();
        }
        catch(Exception ex)
        {
            MessageBox.Show("Error agregando la reserva: " + ex.Message, "Error");
        }
        Close();
    }
    else
    {
        MessageBox.Show("Complete los datos", "Atención");
    }
}
```

En este evento primero se validan que todos los datos necesarios sean correctos, luego se crea un objeto de la clase “CPasaje” y se ejecuta el método “ReservarPasajes” usamos el transporte, la fecha, el número de pasajero, su nombre y la lista de asientos a reservar. Al finalizar el formulario se cierra y se retorna al primer formulario.

El código completo del formulario 2: Form2 es este:

```
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 SP5
{
    public partial class Form2 : Form
    {
        private int Transporte;
        private String Fecha;
        private List<int> Asientos;

        public Form2()
        {
            InitializeComponent();
        }
        public Form2(int transporte, String fecha, List<int> asientos)
        {
            InitializeComponent();
            Transporte = transporte;
            Fecha = fecha;
            Asientos = asientos;
            txtDni.MaxLength = 8;
            txtNombre.MaxLength = 30;
        }

        private void Form2_Load(object sender, EventArgs e)
        {
            CTransporte tr = new CTransporte();
            txtTransporte.Text = tr.GetTransporte(Transporte);
            tr.Dispose();
            txtFecha.Text = Fecha;
            foreach (int a in Asientos)
            {
                lstAsientos.Items.Add(a.ToString());
            }
        }

        private void btnConfirmar_Click(object sender, EventArgs e)
        {
            if(txtDni.Text != "" && txtNombre.Text != "")
            {
                try
                {
                    int transporte = Transporte;
                    String fecha = Fecha;
                    int pasajero = int.Parse(txtDni.Text);
                    String nombre = txtNombre.Text;
                    CPasaje pasajes = new CPasaje();
                    // grabar las reservas
                    pasajes.ReservarPasajes(transporte, fecha,
                        pasajero, nombre, Asientos);
                    MessageBox.Show("Reserva registrada correctamente",
                        "Reservas");
                }
            }
        }
    }
}
```

```

        pasajes.Dispose();
    }
    catch(Exception ex)
    {
        MessageBox.Show("Error agregando la reserva: " + ex.Message,
"Error");
    }
    Close();
}
else
{
    MessageBox.Show("Complete los datos", "Atención");
}
}

private void txtDni_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!Char.IsDigit(e.KeyChar) &&
        e.KeyChar != (char)Keys.Enter &&
        e.KeyChar != (char)Keys.Back)
    {
        e.Handled = true;
    }
    else
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            CPasajero p = new CPasajero();
            String Nombre = p.GetNombrePasajero(int.Parse(txtDni.Text));
            txtNombre.Text = "";
            if (Nombre != "")
            {
                txtNombre.Text = Nombre;
            }
            txtNombre.Focus();
        }
    }
}

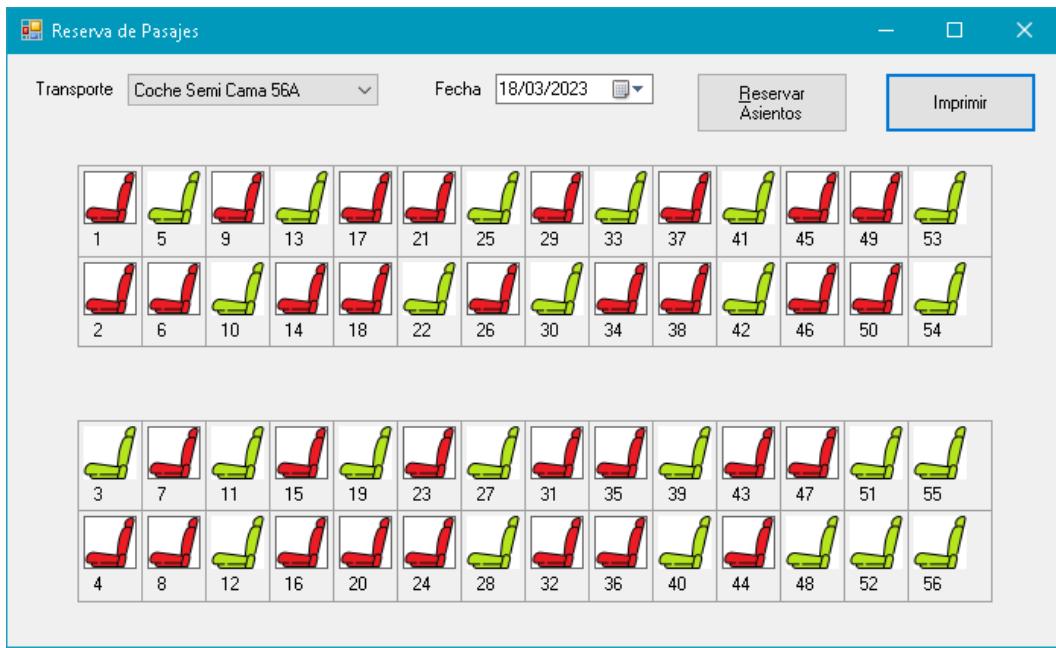
private void btnCancelar_Click(object sender, EventArgs e)
{
    Close();
}
}
}

```

En el evento “**KeyPress**” del TextBox correspondiente al DNI de la persona se agregó un filtrado de teclas para permitir solamente el ingreso de números y que además al presionar la tecla <Enter>, con ese DNI se realice una búsqueda del nombre del pasajero para mostrarlo en el control txtNombre y evitar que el usuario tenga que escribirlo nuevamente, si el DNI ingresado no está registrado para ningún pasajero el control txtNombre permanecerá vacío.

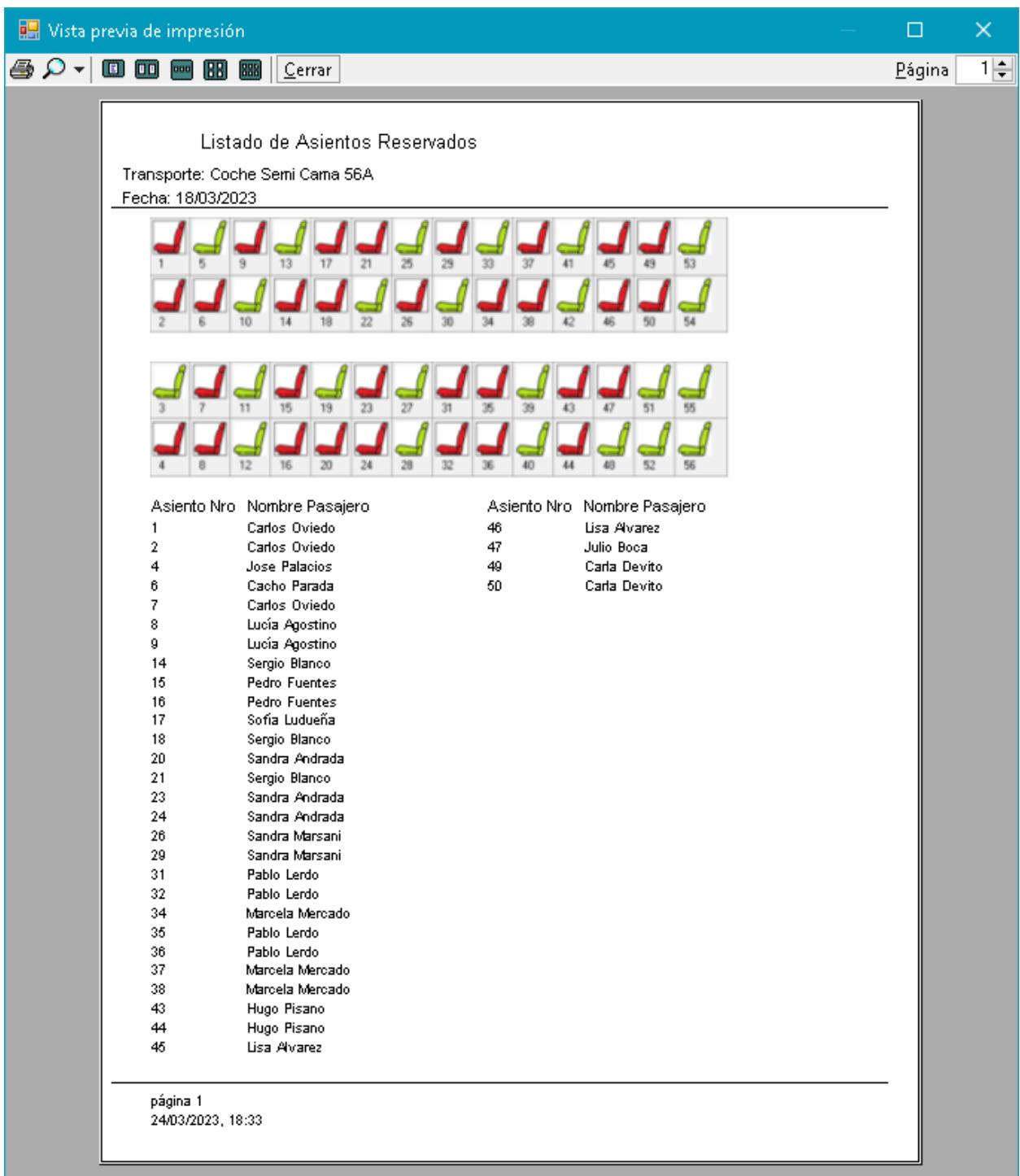
De esta forma concluimos el desarrollo de la aplicación solicitada en la situación profesional.

A continuación, dejamos algunos ejemplos de la aplicación luego de registrar varias reservas en un transporte:



Y el reporte para imprimir se vería así desde la ventana de previsualización:

Vista previa de impresión



Transporte: Coche Semi Cama 56A
Fecha: 18/03/2023

Listado de Asientos Reservados

Asiento Nro Nombre Pasajero Asiento Nro Nombre Pasajero

1	Carlos Oviedo	46	Lisa Alvarez
2	Carlos Oviedo	47	Julio Boca
4	Jose Palacios	49	Carla Devito
6	Cacho Parada	50	Carla Devito
7	Carlos Oviedo		
8	Lucía Agostino		
9	Lucía Agostino		
14	Sergio Blanco		
15	Pedro Fuentes		
16	Pedro Fuentes		
17	Sofía Ludueña		
18	Sergio Blanco		
20	Sandra Andrada		
21	Sergio Blanco		
23	Sandra Andrada		
24	Sandra Andrada		
26	Sandra Marsani		
29	Sandra Marsani		
31	Pablo Lledo		
32	Pablo Lledo		
34	Marcela Mercado		
35	Pablo Lledo		
36	Pablo Lledo		
37	Marcela Mercado		
38	Marcela Mercado		
43	Hugo Pisano		
44	Hugo Pisano		
45	Lisa Alvarez		

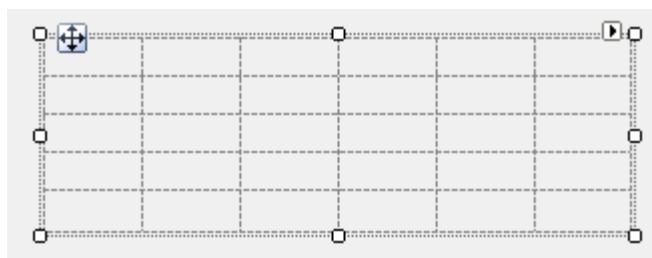
página 1
24/03/2023, 18:33

SP5/Ejercicio por resolver

El administrador de una cerrajería le ha solicitado a usted que, en calidad de pasante de la misma, desarrolle una aplicación administrativa para consultar el nombre, el precio de venta, el stock y la distribución física de los artículos que comercializa.

Además, le solicita que la aplicación genere un reporte con el stock de cada artículo.

Los artículos de la cerrajería se encuentran almacenados en un salón que tiene 30 estanterías distribuidas en 5 filas, cada fila tiene 6 estanterías y las mismas se encuentran numeradas del 1 al 30 de izquierda a derecha y de arriba hacia abajo.



Tenga presente que un mismo artículo puede encontrarse en más de una estantería y en una estantería puede haber distintos tipos de artículos.

Al iniciar la aplicación el usuario ingresa en una caja de texto el código del artículo y al pulsar el botón de comando “consultar” se visualiza en etiquetas el nombre del artículo, el precio del artículo y el stock total del artículo; además, se grafica en un control “PictureBox” o equivalente la distribución física de todas las estanterías del galpón y en qué estanterías se encuentra el artículo consultado.

Cada estantería se grafica con un rectángulo y con su número en el interior del mismo. Las estanterías en las que se encuentra el artículo consultado se dibujan con un rectángulo que tiene un color de relleno para que visualmente se pueda conocer en qué estanterías se encuentra el artículo.

El reporte tiene un título con el nombre de la empresa y una descripción referente a los datos impresos. Las columnas del reporte son: código del artículo, nombre del artículo y stock del artículo.

Al final del reporte se muestra la cantidad total de artículos impresos.

Los datos de la aplicación se encuentran en una base de datos que contiene dos tablas denominadas “Artículos” y “Estanterías”.

Las columnas de la tabla Artículos son: código del artículo (numérico), nombre del artículo (texto) y precio del artículo (numérico), la clave principal de la tabla es el código del artículo.

Las columnas de la tabla Estanterías son: código del artículo (numérico), número de estantería (numérico) y stock del artículo en la estantería (numérico), la clave principal de la tabla está compuesta por las columnas código de artículo y número de estantería.

Ambas tablas se deberán cargar en forma manual con datos suficientes para probar el funcionamiento de la aplicación.

SP5/Evaluación de paso

1. Indique la opción correcta

Para trabajar con GDI y gráficos simples en un proyecto de .Net se debe incluir la cláusula: “using System.Drawing”.

- Verdadero
- Falso

2. Indique la opción correcta

El objeto “Graphics” posee los métodos para controlar la impresión de texto y gráficos.

- Verdadero
- Falso

3. Indique la opción correcta

El sistema de coordenadas usado en GDI permite establecer el origen (0,0) en cualquier punto del contenedor de gráficos.

- Verdadero
- Falso

4. Indique la opción correcta

El control de tipo FlowLayoutPanel se puede usar para distribuir otros controles y organizar el contenido del formulario.

- Verdadero
- Falso

5. Indique la opción correcta

Para agregar controles en un panel de tipo TableLayoutPanel se deben crear solamente en tiempo de diseño del formulario.

- Verdadero
- Falso

6. Indique la opción correcta

Para iniciar el proceso de impresión de un objeto PrintDocument se debe ejecutar el método:

- PrintDocument
- Print
- PrintDoc
- StartPrint

7. Indique la opción correcta

La imagen de un control PictureBox se puede asignar en tiempo de diseño o en tiempo de ejecución, indistintamente.

- Verdadero
- Falso

Respuestas correctas²⁵

²⁵1) Verdadero. 2) Falso. 3) Falso. 4) Verdadero. 5) Falso. 6) StartPrint.
7) Verdadero.

Cierre

A lo largo de este texto hemos profundizado varios conceptos, técnicas y procedimientos relacionados con la funcionalidad de las aplicaciones de escritorio con especial énfasis en la facilidad de uso y en el valor agregado que le otorgan el uso de componentes especializados. Podemos mencionar la creación y uso de excepciones propias o personalizadas, logrando código más robusto, la aplicación de procedimientos temporizados y/o repetitivos y automáticos, el manejo de transacciones con operaciones sobre la base de datos logrando de esa forma asegurar la integridad de los datos, o la programación de gráficos estadísticos y el manejo de procesos de impresión de datos y gráficos entre otros temas desarrollados en este material.

Nuestro objetivo ha sido incentivar el estudio y la investigación de estos temas para que los pueda incorporar en futuros desarrollos en el ámbito profesional, y que sirvan como punto de referencia para continuar avanzando en el conocimiento del lenguaje y la tecnología de desarrollo con Visual Studio .Net, esperamos haber logrado, aunque sea parcialmente, ese cometido.

El Autor

Bibliografía

- *Graphics Clase*,
<https://learn.microsoft.com/es-es/dotnet/api/system.drawing.graphics?view=netframework-4.8.1>.
- Chand, M., and M. Gold. *A Programmer's Guide to ADO.NET in C#*. Apress, 2002.
- D'Andrea, Edgar. *Programación C# con Visual Studio*. Independently published., 2019.
- Hugon, Jérôme. *C# 9 Desarrolle aplicaciones Windows con Visual Studio*. Ediciones Eni, 2019.
- Microsoft. *CultureInfo Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo?view=netframework-4.8>.
- Microsoft. *Trace Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.trace?view=netframework-4.8.1>.
- Microsoft. *EventLog Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.eventlog?view=netframework-4.8.1>.
- Microsoft. *String.Format Method*,
<https://learn.microsoft.com/en-us/dotnet/api/system.string.format?view=netframework-4.8.1>.
- Microsoft. *DateTimePicker Clase*,
<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.datetimepicker?view=windowsdesktop-7.0>.
- Microsoft. *Timer Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.timers.timer?view=netframework-4.8.1>.
- Microsoft. *Transacciones locales*,
<https://learn.microsoft.com/es-es/dotnet/framework/data/adonet/local-transactions>.
- Microsoft. *DirectoryInfo Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.io.directoryinfo?view=netframework-4.8.1>.
- Microsoft. *Path Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.io.path?view=netframework-4.8.1>.
- Microsoft. *Directory Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.io.directory?view=netframework-4.8.1>.

- Microsoft. *FileInfo Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo?view=netframeworkdesktop-4.8.1>.
- Microsoft. *ImageList Class*,
<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.imagelist?view=netframework-4.8.1>.
- Microsoft. *TreeView Class*,
<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.treeview?view=netframework-4.8.1>.
- Microsoft. *ListView Clase*,
<https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.listview?view=netframework-4.8.1>.
- Microsoft. *Información general del control StatusStrip*,
<https://learn.microsoft.com/es-es/dotnet/desktop/winforms/controls/statusstrip-control-overview?view=netframeworkdesktop-4.8>.
- Microsoft. *ToolStrip Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.toolstrip?view=netframework-4.8.1>.
- Microsoft. *Chart Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8.1>.
- Microsoft. *TabControl Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.tabcontrol?view=netframework-4.8.1>.
- Microsoft. *Graphics Clase*,
<https://learn.microsoft.com/es-es/dotnet/api/system.drawing.graphics?view=netframework-4.8.1>.
- Microsoft. *ToolStrip Class*,
<https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.toolstrip?view=netframework-4.8.1>.
- Microsoft. *ProgressBar Control (Windows Forms)*.
<https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/progressbar-control-windows-forms?view=netframeworkdesktop-4.8>.