

## Situación profesional 5: Reserva de pasajes

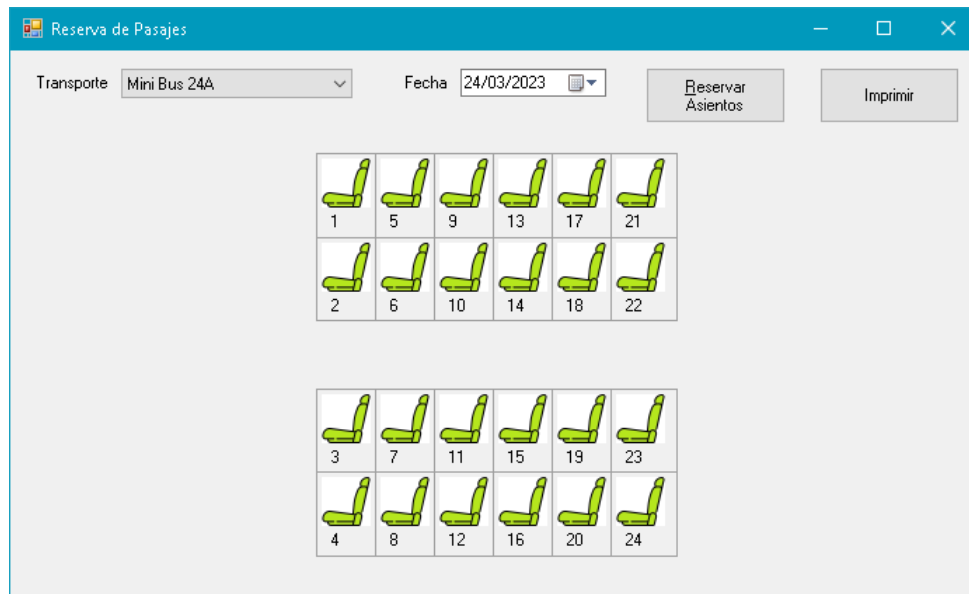
El administrador de una empresa de transporte de pasajeros le ha solicitado a usted que, en calidad de pasante de la misma, desarrolle una aplicación administrativa para registrar las reservas de pasajes que pueden realizar sus clientes para las diferentes unidades de transporte y fechas de viajes. Además, 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 esté diseño:

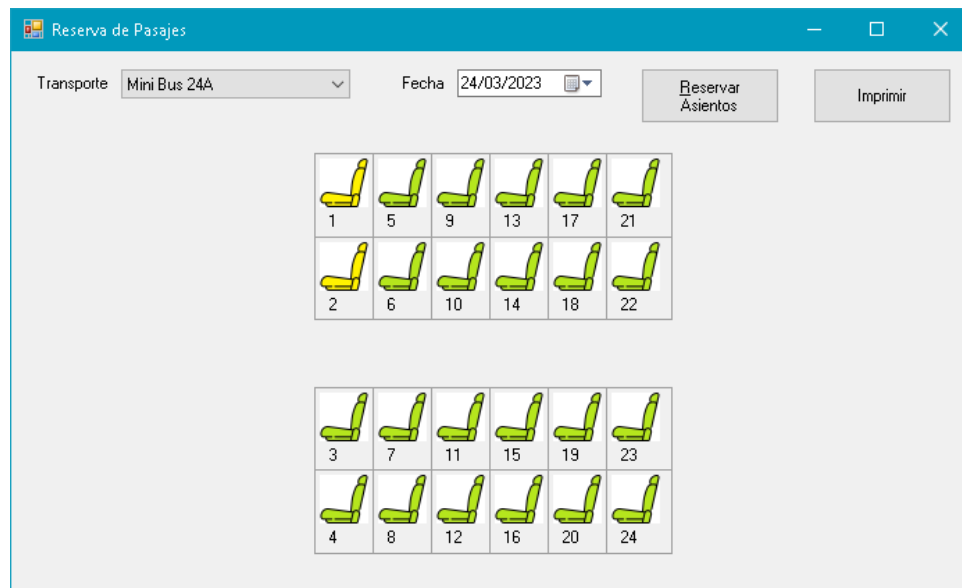
The screenshot displays the 'Reserva de Pasajes' application window. At the top, there's a title bar with standard window controls. Below it, the 'Transporte' dropdown is set to 'Coche Semi Cama 56A', and the 'Fecha' field shows '24/03/2023'. To the right are buttons for 'Reservar Asientos' and 'Imprimir'. The main area contains three grids of seat icons, each with a number, representing the available seating arrangement for the selected transport and date.

Row	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12	Col 13
1	1	5	9	13	17	21	25	29	33	37	41	45	49
2	2	6	10	14	18	22	26	30	34	38	42	46	50
3	3	7	11	15	19	23	27	31	35	39	43	47	51
4	4	8	12	16	20	24	28	32	36	40	44	48	52
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													
21													
22													
23													
24													
25													
26													
27													
28													
29													
30													
31													
32													
33													
34													
35													
36													
37													
38													
39													
40													
41													
42													
43													
44													
45													
46													
47													
48													
49													
50													
51													
52													
53													
54													
55													
56													

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 click 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 'Reserva' window contains the following fields and buttons:

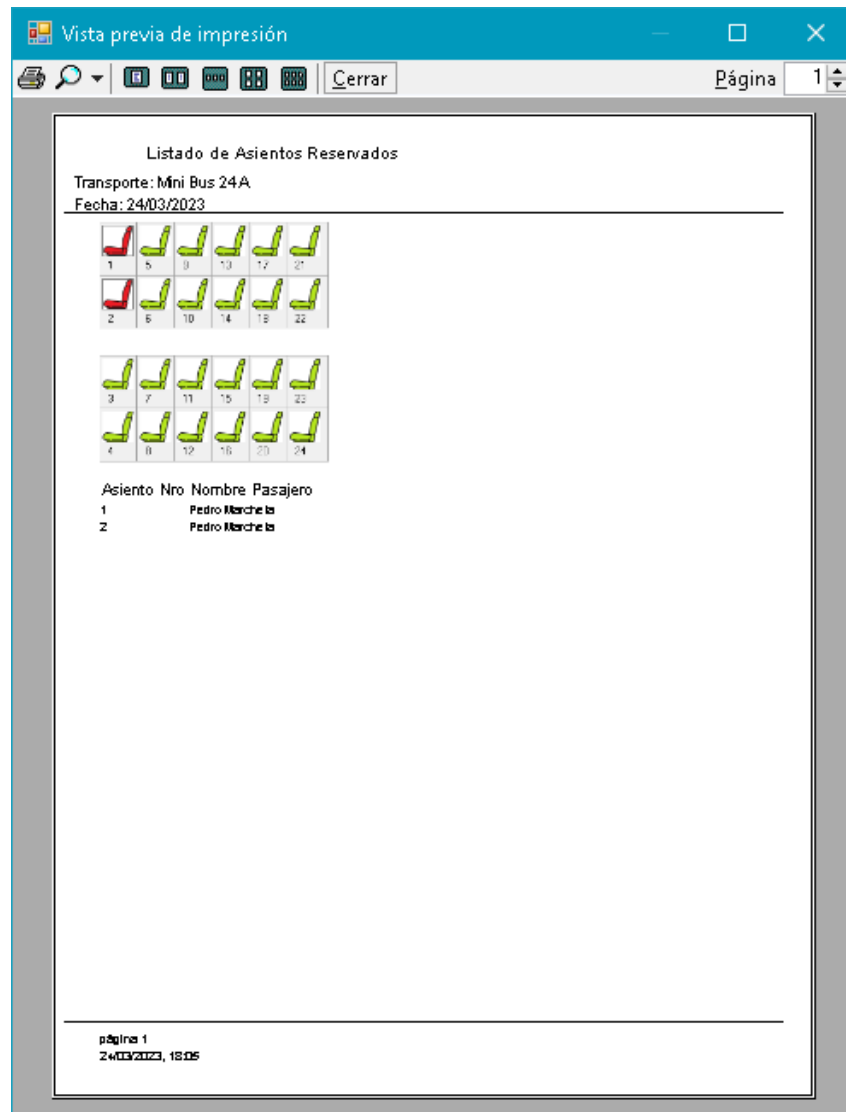
- Transporte: Mini Bus 24A
- Fecha: 24/03/2023
- DNI: 25123987
- Nombre: Pedro Marcheta
- Asientos: 1, 2
- Buttons: Confirmar Reserva, Cancelar

Al confirmar la reserva los asientos seleccionados previamente quedarán de color rojo y ya no se podrán volver a cambiar.

The 'Reserva de Pasajes' window displays a seat chart for a Mini Bus 24A on 24/03/2023. The chart consists of two rows of six seats each, numbered 1 to 24. Seats 1 and 2 are highlighted in red, indicating they are reserved. The window includes buttons for 'Reservar Asientos' and 'Imprimir'.

Row	Seat	Status
1	1	Reserved
1	2	Reserved
1	3	Available
1	4	Available
1	5	Available
1	6	Available
2	7	Available
2	8	Available
2	9	Available
2	10	Available
2	11	Available
2	12	Available
2	13	Available
2	14	Available
2	15	Available
2	16	Available
2	17	Available
2	18	Available
2	19	Available
2	20	Available
2	21	Available
2	22	Available
2	23	Available
2	24	Available

El reporte de reservas se debe generar con el botón “Imprimir”, y mostrará un cuadro de pre-visualización 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:

Transportes			
	transporte	Descripcion	Asientos
+	1	Coche Semi Cama 56A	56
+	2	Coche 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:

<b>BackColor</b>	Sirve para poner un color de fondo al control. El color más usado es el transparente.
<b>SizeMode</b>	Indica cómo se adecua la imagen ante el control, ya sea estirarlo (o mejor dicho autoajustar), tamaño original, zoom y centrado.
<b>BackgroundImageLayout</b>	Nos indica la forma en que se mostrará la imagen: en mosaico, centrado, estirado o zoom.
<b>Image</b>	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.

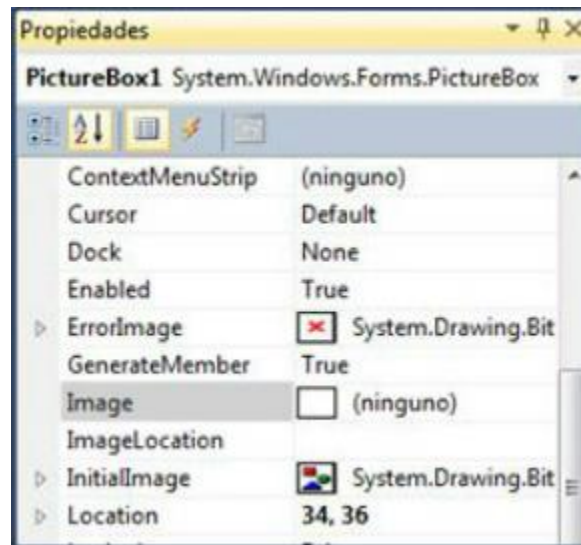
### Cargar una imagen en el PictureBox desde un archivo

Necesitamos un archivo con formato gráfico, como jpg, bmp, png, etc, usaremos su nombre y ubicación para la carga del pictureBox:

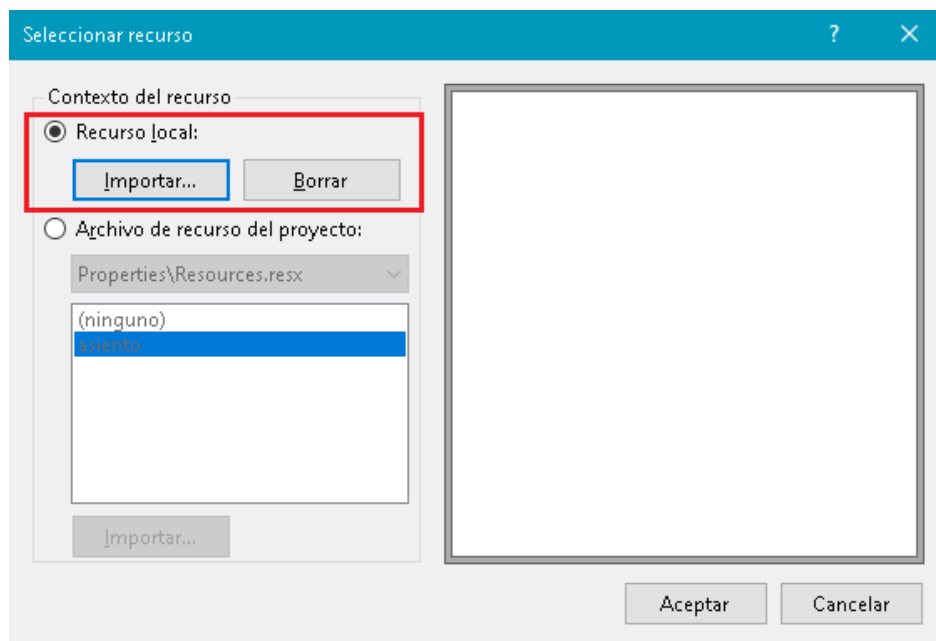
La propiedad “**Image**” almacena el contenido gráfico del archivo y el método “**FromFile**” se encarga de transferir la imagen desde el archivo al pictureBox.

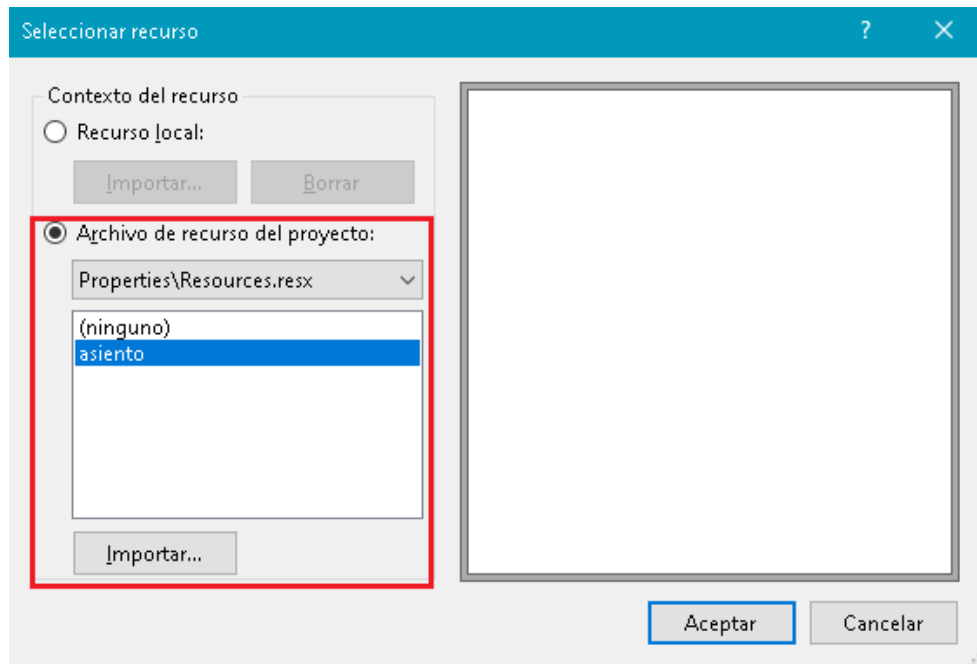
```
PictureBox1.Image = Image.FromFile(“nombre_archivo”);
```

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:

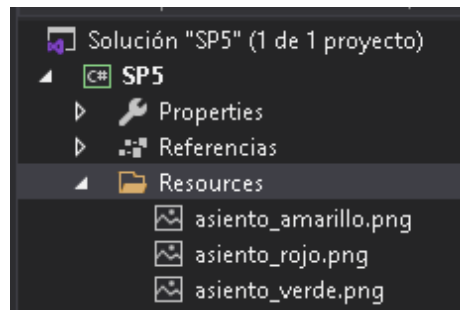


Una vez ubicado en la propiedad image, haremos clic en el botón de la derecha y se abrirá la siguiente ventana, donde haremos clic en Import e importaremos (añadiremos) las imágenes que vamos a usar, tal como se muestra a continuación:





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            X

### 2. Indique la opción correcta

El método CreateGraphics() del PictureBox crea un objeto de tipo Graphics.

Verdadero    X

Falso

### 3. Indique la opción correcta

¿Dónde podemos ubicar por código los recursos agregados al proyecto?

Imports Resources

Resources    X

this.Resources

Picture.Resources

### 4. Indique la opción correcta

Para dibujar sobre un PictureBox se necesita un crear un objeto de tipo Draw.

Verdadero

Falso            X

### 5. Indique la opción correcta

¿Cuál es la propiedad que me permite cargar imágenes?

Picture.Image            X

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      X

## 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***

Proporciona acceso a 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 texto 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 rellenar áreas de gráficos, como formas rellenas, 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 línea horizontal desde el punto (0,10) al punto (100,10)
gra.DrawLine(Pens.Black, new Point(0, 10), new Point(100, 10));
// dibujar un rectángulo 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 de 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.

<i>Display</i>	1/75 pulgadas.
<i>Document</i>	1/300 pulgadas.
<i>Inch</i>	Pulgada.
<i>Millimeter</i>	Milímetro.
<i>Pixel</i>	Píxel.
<i>Point</i>	Punto de impresión (1/72 pulgadas).
<i>World</i>	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 de 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 de 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.

<b>Punto</b>	<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> <pre>Dim MiPunto As Point = New Point(10, 10)</pre>
<b>Línea</b>	<p>Una línea se define por dos puntos y el trazo que los une. El método que dibuja líneas se llama <b>DrawLine</b>:</p> <pre>Dim Lápiz As New Pen(Color.Blue, 10) Dim Punto1 As Point = New Point(10, 10) Dim Punto2 As Point = New Point(100, 100) Lienzo.DrawLine(Lápiz, 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>
<b>Rectángulo</b>	<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 <b>DrawRectangle</b> que acepta como parámetros el lápiz y un objeto Rectangle:</p> <pre>Lienzo.DrawRectangle(Lápiz, New Rectangle(100, 100, 200, 100))</pre>

<b>Elipse</b>	<p>El método <b>DrawEllipse</b> toma los mismos argumentos que el método <b>DrawRectangle</b>, pero en vez de dibujar un rectángulo, dibuja una elipse circunscrita en él. Para dibujar un círculo, por tanto, hay que definir un cuadrado:</p> <p>'Dibuja una elipse circunscrita 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>
<b>Arco</b>	<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 <b>DrawArc</b>:</p> <p>gr.DrawArc(New Pen(Color.Red, 10), New Rectangle(250, 250, 250, 250), 0, 90)</p> <p>Entonces, podemos decir que estamos dibujando un arco cuyo punto inicial es el punto 0º y se extiende 90º en el sentido de las agujas del reloj.</p>
<b>Polígono</b>	<p>El método <b>DrawPolygon</b> recibe, además del lápiz, un array de puntos y los une con rectas. <b>DrawPolygon</b> une el primer punto con el último del array formando así un polígono cerrado. El siguiente ejemplo dibuja un rombo:</p> <p>Lienzo.DrawPolygon(Lapiz, New Point() {New Point(200, 50), New Point(300, 100), New Point(200, 150), New Point(100, 100)})</p>

## 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:

<b><i>SolidBrush</i></b>	<p>Rellena la forma con un solo color.</p> <p>Es la brocha más sencilla pues se compone de un solo color:</p> <pre>Dim BrochaSólida As SolidBrush = New SolidBrush(Color.LightGreen)</pre>
<b><i>HatchBrush</i></b>	<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>
<b><i>TextureBrush</i></b>	<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>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.</pre>

<b><i>LinearGradientBrush</i></b>	<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>
<b><i>PathGradientBrush</i></b>	<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, <b>PathGradientBrush</b> lanza los colores desde el centro y los vértices de la figura. El constructor de <b>PathGradientBrush</b> 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 debemos entregar al constructor de <b>PathGradientBrush</b> 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:**

<https://learn.microsoft.com/es-es/dotnet/desktop/winforms/advanced/coordinate-systems-and-transformations?view=netframeworkdesktop-4.8>

#### **Uso del lápiz (Pen) para dibujar líneas y rectángulos:**

<https://learn.microsoft.com/es-es/dotnet/desktop/winforms/advanced/using-a-pen-to-draw-lines-and-shapes?view=netframeworkdesktop-4.8>

#### **Uso del pincel (Brush) para rellenar figuras:**

<https://learn.microsoft.com/es-es/dotnet/desktop/winforms/advanced/using-a-brush-to-fill-shapes?view=netframeworkdesktop-4.8>

**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    ☒ X

Falso

**2. Ordene relaciones**

Relacione los conceptos de la columna izquierda con sus características correspondientes en la columna derecha:

Pincel	Se utiliza con el fin de describir el color que se usa para procesar un objeto determinado. (Color)
Lápiz	Se utiliza para describir la fuente que se usa para procesar textos. (Fuente)
Fuente	Se utiliza para dibujar líneas y polígonos, junto con rectángulos, arcos y gráficos. (Lápiz)
Color	Se utiliza para rellenar superficies delimitadas Con esquemas, colores o mapas de bits. (Pincel)

**3. Indique la opción correcta**

Los valores de los colores a usar se pueden obtener a partir del objeto **Drawing.Color**.

Verdadero    ☒ X

Falso

**4. Ordene relaciones**

Relacione los conceptos de la columna izquierda con sus características correspondientes en la columna derecha:

FillRectangle	Permite dibujar un cuadrado o rectángulo. (DrawRectangle)
---------------	---

FillPolygon	Permite dibujar un círculo o elipse. (DrawEllipse)
DrawEllipse	Permite pintar cualquier polígono. (FillPolygon)
DrawRectangle	Permite pintar un cuadrado o rectángulo. (FillRectangle)

**5. Indique la opción correcta**

Una vez utilizado el objeto Graphics es conveniente cerrarlo invocando el método:

Clear

Changed

Dispose ☒ X

UnLoad

**6. Indique la opción correcta**

¿Cuál es el espacio de nombre que utilizamos para dibujos lineales?:

System.Drawing.Line

System.Drawing.Drawing2D ☒ X

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 ☒ X

Falso

**8. Indique la opción correcta**

El método "FillCircle" del objeto Graphics permite dibujar un círculo.

Verdadero

Falso ☒ X

**9. Indique la opción correcta**

El método CreateGraphics sirve para dibujar un nuevo gráfico.

Verdadero

Falso            X



## 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

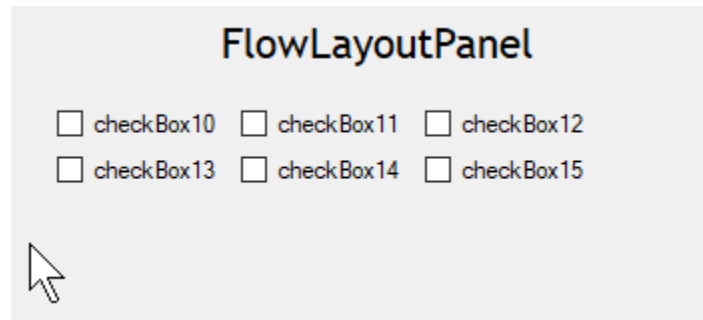
El control FlowLayoutPanel organiza su contenido en una dirección de flujo específica: horizontal o vertical. Su contenido puede ajustarse desde una fila a la siguiente o desde una columna a la siguiente. Además, el contenido puede ajustarse a determinadas dimensiones o puede ajustarse según el caso.

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, incluidas 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:

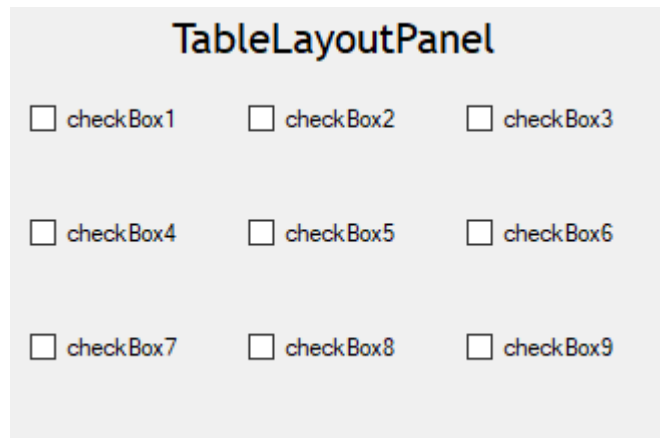
- **AutoSize**: indica si el tamaño del control se ajustará automáticamente a su contenido.
- **AutoSizeMode**: indica el comportamiento de ajuste automático de tamaño del control.
- **BorderStyle**: establece el estilo del borde que presenta el control.
- **Controls**: colección de controles contenidos en el FlowLayoutPanel.
- **Dock**: indica si los bordes de control se acoplarán 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.flowlayoutpanel?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 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 desenlazar  en la direcci n correspondiente.

Tambi n se puede controlar la direcci n de expansi n cuando el contenido del control TableLayoutPanel se llene 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 tama o del control se ajustar  autom ticamente a su contenido.
- **AutoSizeMode:** indica el comportamiento de ajuste autom tico de tama o del control.
- **BorderStyle:** establece el estilo del borde que presenta el control.
- **ColumnCount:** cantidad de columnas del control
- **ColumnStyles:** estilos de las columnas
- **Controls:** colecci n de controles contenidos en el FlowLayoutPanel.
- **Dock:** indica si los bordes de control se acoplar n 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 est n ocupadas.
- **RowCount:** cantidad de filas del control
- **RowStyles:** colecci n de estilos para las filas del control.
- **Size:** establece el alto y ancho del control

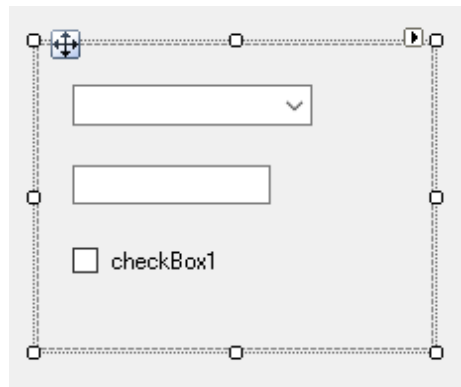
### **Control Panel**

El control de tipo Panel se usa para proporcionar una agrupaci n para otros controles. Uno de los usos m s frecuentes del Panel consiste en subdividir el contenido de un formulario por funci n. Por ejemplo, las diferentes opciones de pago que tiene el cliente para realizar una compra. La agrupaci n de todas las opciones en un panel proporciona al usuario una indicaci n visual l gica. En tiempo de dise o, 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:

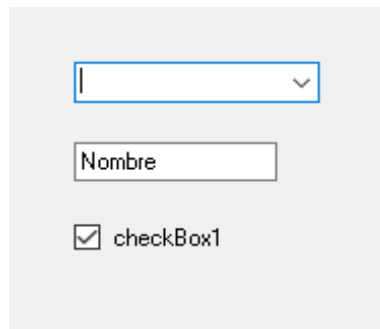
```
foreach(Control c in panel1.Controls)
{
    if(c.Name == "comboBox1")
    {
        ((ComboBox)c).Items.Clear(); // cuando 'c' es un ComboBox
    }
    if(c.Name == "textBox1")
    {
        ((TextBox)c).Text = "Nombre"; // cuando 'c' es un TextBox
    }
    if(c.Name == "checkBox1")
    {
        (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).

## SP5/Autoevaluación 3

### 1. Indique la opción correcta

El control FlowLayoutPanel se usa como contenedor de otros controles.

Verdadero      X

Falso

### 2. Indique la opción correcta

En un control FlowLayoutPanel se pueden agregar solamente controles de un solo tipo.

Verdadero

Falso            X

**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            X

**4. Indique la opción correcta**

El control TableLayoutPanel organiza su contenido en filas columnas.

Verdadero      X

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      X

Falso

**6. Indique la opción correcta**

El control Panel puede cambiar su tamaño automáticamente según su contenido.

Verdadero      X

Falso



## 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, esa acción creará el evento “PrintPage”. También se puede agregar el evento desde la ventana de propiedades.
3. Escribir el código para el evento PrintPage. Aquí 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	
 <b>PrintDocument</b> (Constructor)	Inicializa una nueva instancia de la clase <b>PrintDocument</b> .
Propiedades públicas	
 <b>DefaultPageSettings</b>	Obtiene o establece la configuración de página que se utiliza como predeterminada para todas las páginas que se van a imprimir.
 <b>DocumentName</b>	Obtiene o establece el nombre del documento que va a aparecer mientras se imprime el documento.
 <b>OriginAtMargins</b>	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.
 <b>PrinterSettings</b>	Obtiene o establece la impresora que imprime el documento.
Métodos públicos más usados	
 <b>Dispose</b> (se hereda de <b>Component</b> )	Sobrecargado. Libera los recursos utilizados por <b>Component</b> .
 <b>Print</b>	Inicia el proceso de impresión del documento.
 <b>ToString</b>	Reemplazado. Vea <b>Object.ToString</b> .

Eventos públicos más usados	
 <b>BeginPrint</b>	Se produce cuando se llama al método <b>Print</b> antes de que se imprima la primera página del documento.
 <b>EndPrint</b>	Se produce cuando se ha impreso la última página del documento.
 <b>PrintPage</b>	Se produce cuando se necesita el resultado que se va a imprimir para la página actual.
Métodos protegidos	
 <b>OnBeginPrint</b>	Provoca el evento <b>BeginPrint</b> . Se llama después de llamar al método <b>Print</b> y antes de que se imprima la primera página del documento.
 <b>OnEndPrint</b>	Provoca el evento <b>EndPrint</b> . Se llama cuando se ha impreso la última página del documento.
 <b>OnPrintPage</b>	Provoca el evento <b>PrintPage</b> . Se llama antes de que se imprima una página.
 <b>OnQueryPageSettings</b>	Provoca el evento <b>QueryPageSettings</b> . Se llama justo antes de cada evento <b>PrintPage</b> .

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**.

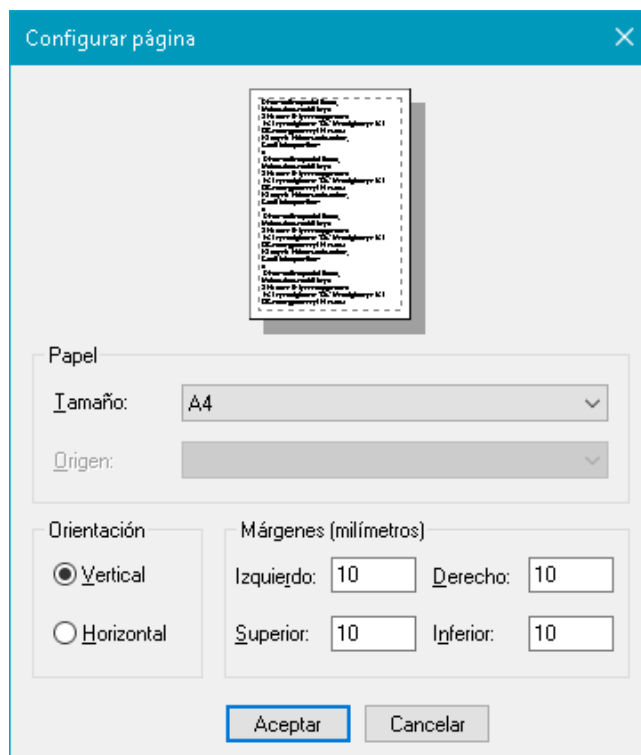
Veremos entonces las características de estos tres componentes.

### 1. PageSetupDialog

El componente PageSetupDialog de formularios Windows Forms es un cuadro de diálogo Pre-configurado 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 resultarán 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.

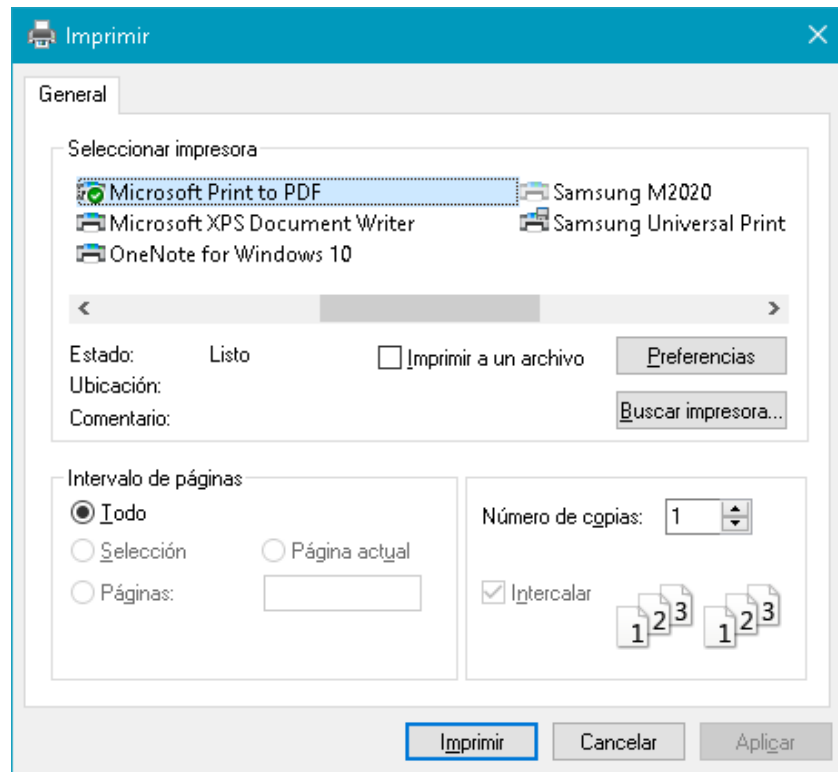
```
private void btnImprimir_Click(object sender, EventArgs e)
{
    // se asigna el documento a configurar
    pageSetupDialog1.Document = printDocument1;
    // se muestra el diálogo para configurar la página
    if (pageSetupDialog1.ShowDialog() == DialogResult.OK)
    {
        // si se confirman los cambios (botón OK), se imprime
        printDocument1.Print();
    }
}
```

Las principales propiedades de la clase PageSetupDialog son:

Propiedades públicas	
 <b>AllowMargins</b>	Obtiene o establece un valor que indica si está habilitada la sección de márgenes del cuadro de diálogo.
 <b>AllowOrientation</b>	Obtiene o establece un valor que indica si está habilitada la sección de orientación (horizontal o vertical) del cuadro de diálogo.
 <b>AllowPaper</b>	Obtiene o establece un valor que indica si se habilita la sección de papel (tamaño y origen de papel) del cuadro de diálogo.
 <b>AllowPrinter</b>	Obtiene o establece un valor que indica si el botón Impresora está habilitado.
 <b>Document</b>	Obtiene o establece un valor que indica el <b>PrintDocument</b> del que se obtendrá la configuración de página.
 <b>MinMargins</b>	Obtiene o establece un valor que indica los márgenes mínimos que se permiten seleccionar, expresados en centésimas de pulgadas.
 <b>PageSettings</b>	Obtiene o establece un valor que indica la configuración de la página que se va a modificar.
 <b>PrinterSettings</b>	Obtiene o establece la configuración de impresora que se modifica cuando el usuario hace clic en el botón Impresora del cuadro de diálogo.
 <b>ShowNetwork</b>	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 pre-configurado, 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 o imprimir una selección.



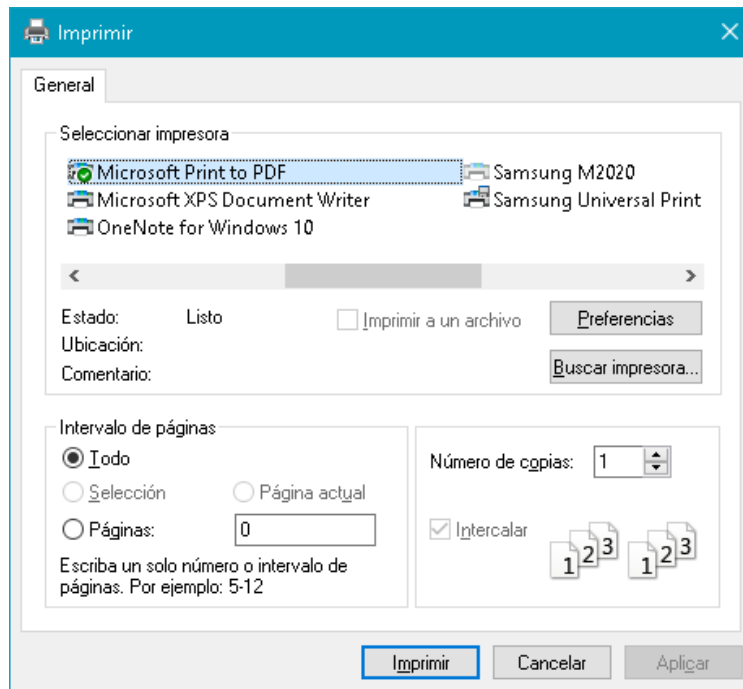
Utilice el método **ShowDialog** para mostrar el cuadro de diálogo en tiempo de ejecución. Este componente tiene propiedades relacionadas con un único trabajo de impresión (clase `PrintDocument`) o con las configuraciones de una impresora individual (clase `PrinterSettings`). Cualquiera de ellas, a su 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:



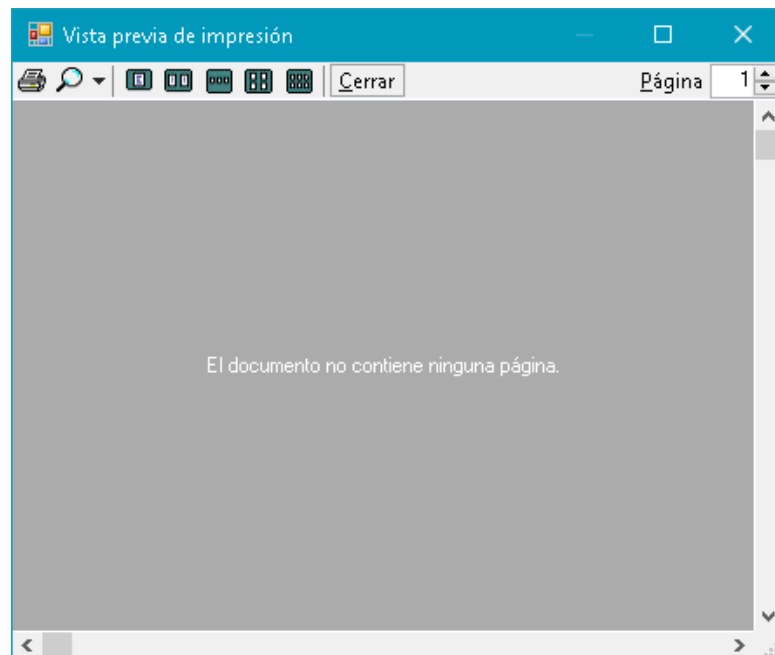


Las principales propiedades de la clase PrintDialog son:

Propiedades públicas	
 <b>AllowPrintToFile</b>	Obtiene o establece un valor que indica si la casilla de verificación Imprimir a un archivo está activada.
 <b>AllowSelection</b>	Obtiene o establece un valor que indica si está habilitado el botón de opción Página desde... hasta...
 <b>AllowSomePages</b>	Obtiene o establece un valor que indica si se habilita el botón de opción Páginas.
 <b>Document</b>	Obtiene o establece un valor que indica el <i>PrintDocument</i> del que se obtendrá <i>PrinterSettings</i> .
 <b>PrinterSettings</b>	Obtiene o establece la configuración de impresora que se modifica en el cuadro de diálogo.
 <b>PrintToFile</b>	Obtiene o establece un valor que indica si la casilla de verificación Imprimir a un archivo está activada.
 <b>ShowNetwork</b>	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 imprimirlo. 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.

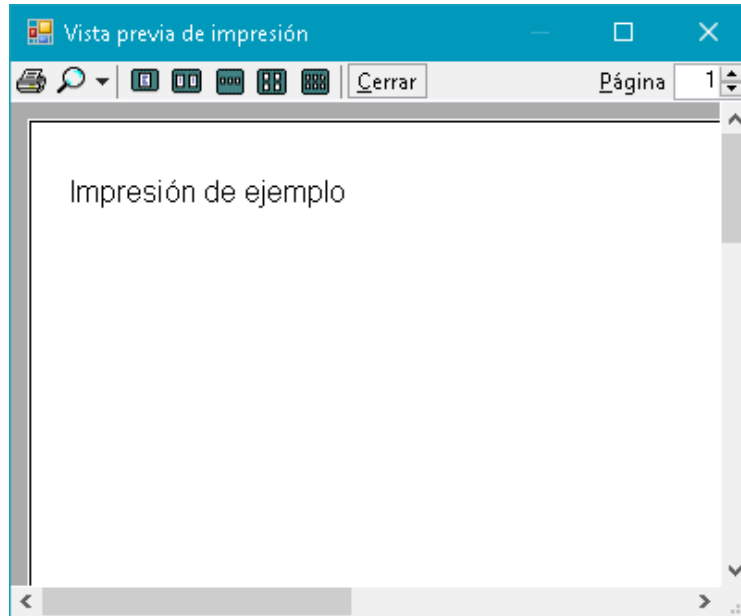
Para pre-visualizar páginas con el control **PrintPreviewDialog** utilice el método **ShowDialog** para mostrar el cuadro de diálogo, especificando el componente **PrintDocument** que se utilizará.

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	
 <b>Bounds</b> (se hereda de <i>Control</i> )	Obtiene o establece el tamaño y la ubicación del control, incluidos los elementos de no cliente.
 <b>CanFocus</b> (se hereda de <i>Control</i> )	Obtiene un valor que indica si el control puede recibir el foco.
 <b>CanSelect</b> (se hereda de <i>Control</i> )	Obtiene un valor que indica si el control se puede seleccionar.
 <b>DialogResult</b> (se hereda de <i>Form</i> )	Obtiene o establece el resultado de cuadro de diálogo para el formulario.
 <b>Document</b>	Obtiene o establece el documento del que se desea la vista previa.
 <b>Name</b> (se hereda de <i>Control</i> )	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      X

**2. Indique la opción correcta**

El método PrintDoc de un objeto PrintDocument inicia el proceso de impresión.

Verdadero

Falso      X

**3. Indique la opción correcta**

La clase PrintDocument es la base para controlar los trabajos de impresión.

Verdadero      X

Falso

**4. Indique la opción correcta**

Para detener el trabajo de impresión se usa el método EndPrint.

Verdadero

Falso      X

**5. Indique la opción correcta**

¿Cuál de estas sentencias de códigos nos mostrará una ventana para confirmar nuestra impresión?

ShowDialog();

PrintDialog1.ShowDialog();      X

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    ☒ X

Falso

**7. Indique la opción correcta**

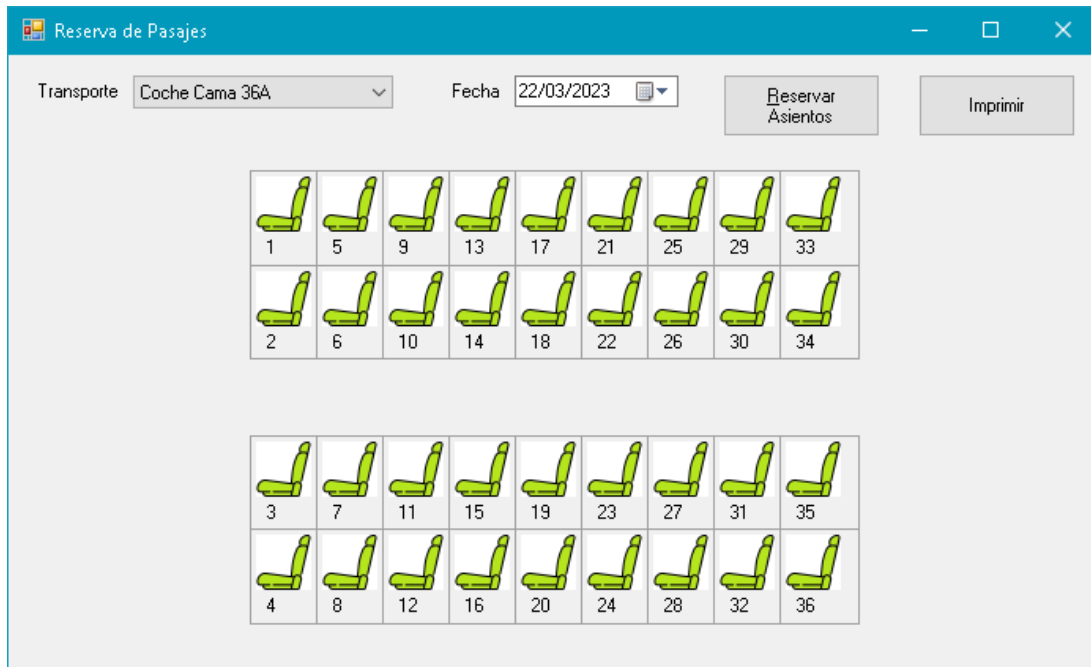
El evento PrintPage se usa para codificar la lógica del proceso que controla la impresión.

Verdadero    ☒ X

Falso

## SP5/Ejercicio resuelto

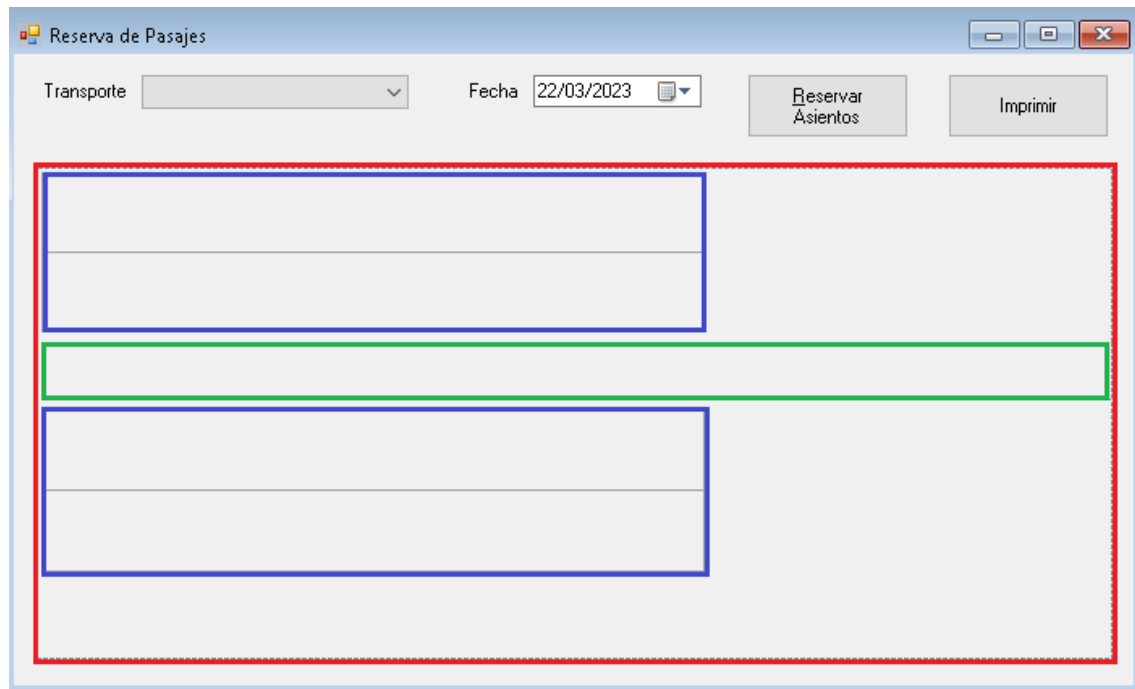
Comenzamos la resolución de esta situación profesional creando un nuevo proyecto con Visual Studio .Net y trabajando en el diseño del primer formulario, tenemos que obtener algo así:



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.

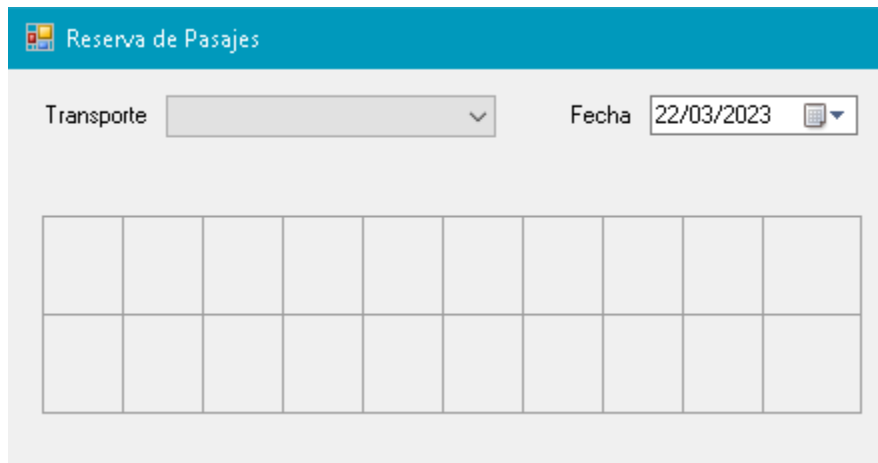
El recuadro de color **verde** es un control de tipo **Panel**, que usaremos solamente como separador de los otros dos controles.

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));
}
```

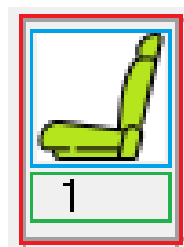
El estilo de las columnas es: “sizeType.Percent” (porcentaje) y el valor es de 10, o sea cada columna tendrá un 10% del ancho del TableLayoutPanel.

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 icono 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 icono 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 Lable
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 icono 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 “Click” sobre el asiento se deberá cambiar el icono 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 Lable
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 completemos 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, asignado 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 “Click” 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 deshabilitarlo.

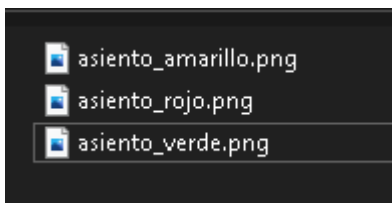
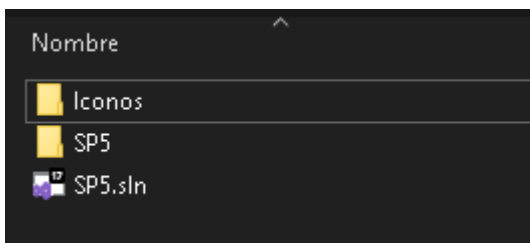
```
private void ActualizarEstado(int asiento)
{
    // se recorren todos los controles del TableLayoutPanel
    foreach (Control c in tablePanelAsientos.Controls)
    {
        // c.Controls[1] es el control Label que tiene el número de asiento
        if (c.Controls[1].Text == asiento.ToString())
        {
            // 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.

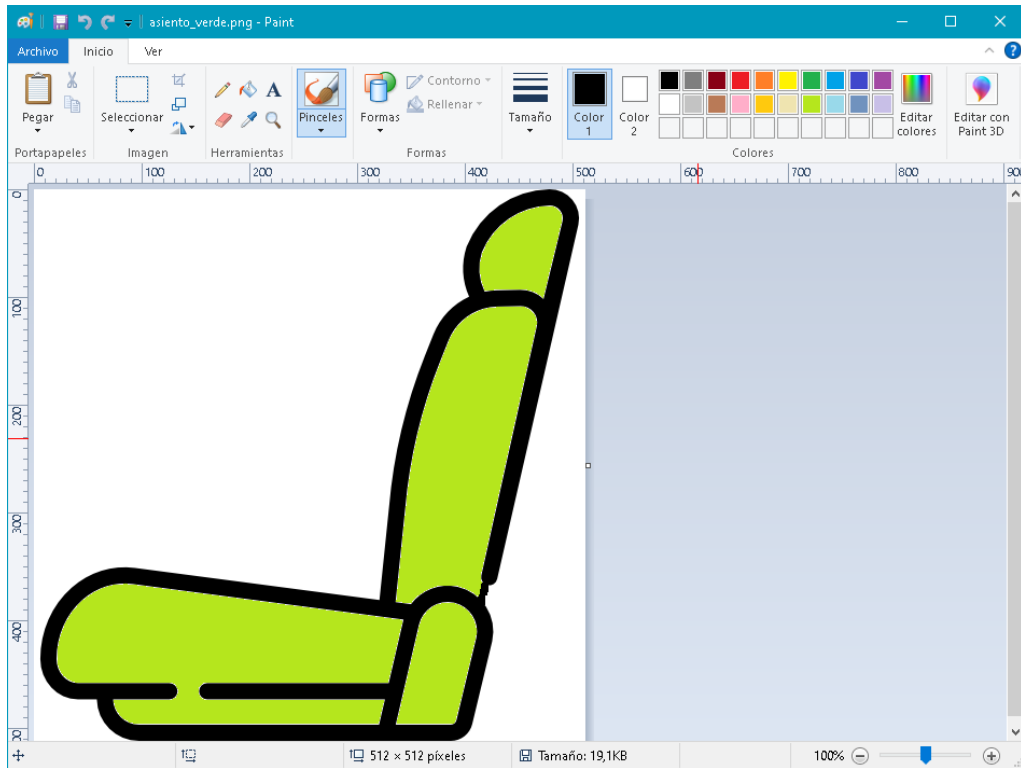
Todos estos procedimientos están relacionados con la presentación de datos en el estado inicial del formulario y la posible interacción por parte del usuario para seleccionar asientos libres en los paneles, la implementación final de todo esto la veremos más adelante ya que previamente debemos tener 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 pixels x 512 pixels 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

```
CargarTransportes(System.Windows.Forms.ComboBox)
CTransporte()
Dispose()
GetAsientos(int)
GetTransporte(int)
DS
```

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"];
    }

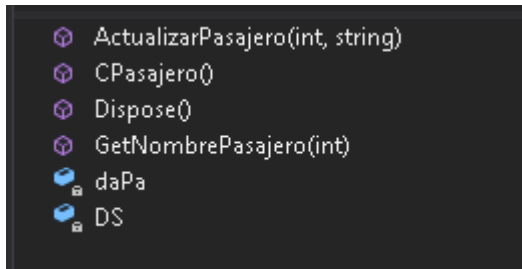
    // devuelve la cantidad total de asientos del transporte
    public int GetAsientos(int transporte)
    {
        int Asientos = 0;
        try
        {
            DataRow tr = DS.Tables["Transportes"].Rows.Find(transporte);
            if(tr == null)
            {
                throw new Exception("CTransporte: 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";
            }
            catch { }
        }

        public void ActualizarPasajero(int dni, string nombre)
        {
            // Implementación del método ActualizarPasajero
        }

        public string GetNombrePasajero(int dni)
        {
            // Implementación del método GetNombrePasajero
        }

        public void Dispose()
        {
            // Implementación del método Dispose
        }
    }
}
```

```

        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 determinados.

El método “**GetPasajesReservados**” obtiene una lista con los números de asiento reservados para un transporte y una fecha determinados.

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;
}

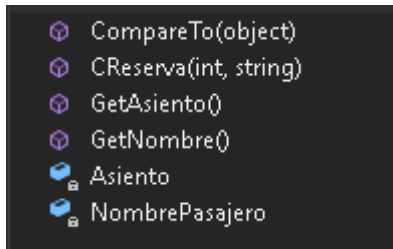
// devuelve la lista de Reservas para el transporte y fecha pasados por
parámetro
public List<CReserva> GetPasajeros(int transporte, String fecha)
{
    List<CReserva> reservas = new List<CReserva>();
    CPasajero pasajero = new CPasajero();
    // se recorre la tabla de Pasajes
    foreach (DataRow 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:

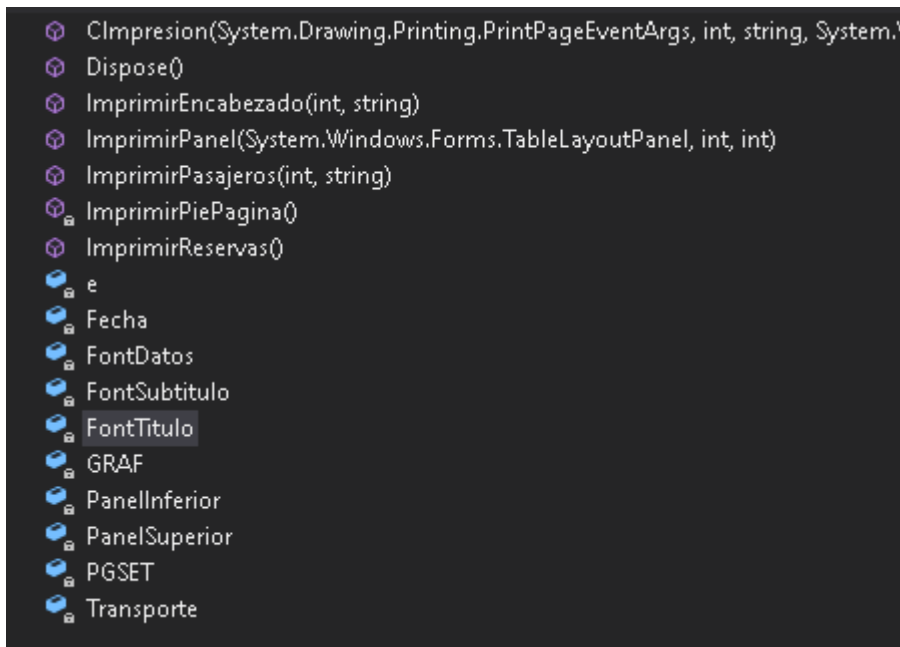
- Si ambos valores comparados son iguales devuelve cero
- Si el valor del objeto actual es mayor al valor del objeto recibido por parámetro, devuelve un valor mayor a cero.
- Si el valor del objeto actual es menor al valor del objeto recibido por parámetro, devuelve un valor menor a cero.

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 CImpresion**





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 cómo 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 numero de asiento
        reservas.Sort();

        int X = 50; // posición inicial para los datos
        int Y = 410;
        // imprimir titulos 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 proxima 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 icono 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 Click 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_amarillo.png");
        pic.Image.Tag = "A"; // Amarillo
    }
    else // es Amarillo
    {
        pic.Image = Image.FromFile("../.../Iconos/asiento_verde.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++)
            {
                // 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;
    }

    // el parámetro de método es un número de asiento reservado
    // se debe ubicar en el panel y modificar el icono del pictureBox
    // por un icono 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
    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 le 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 != "")

```

```

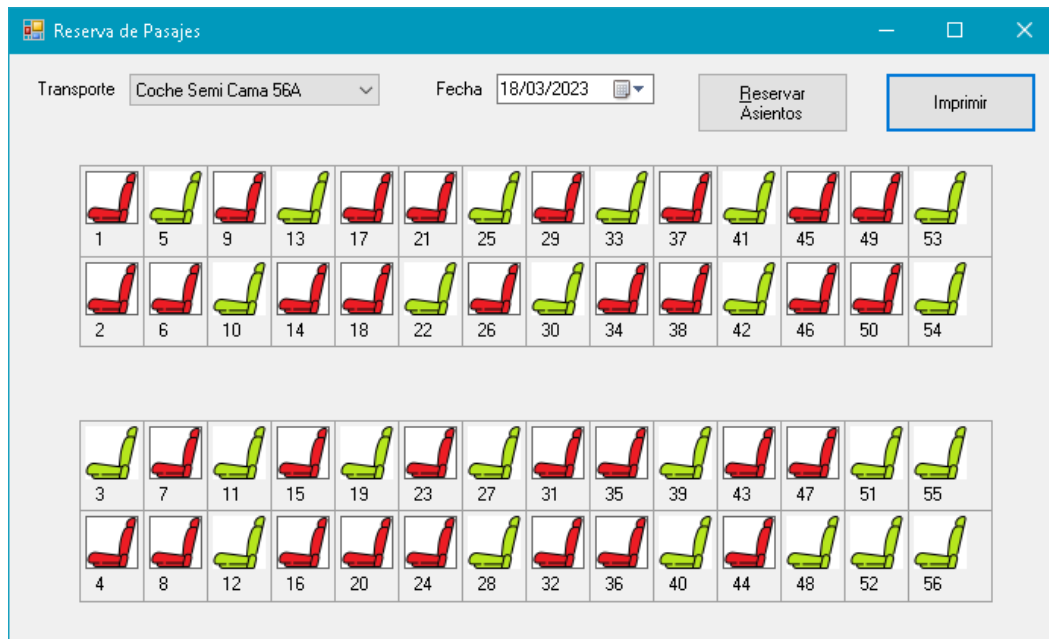
        {
            txtNombre.Text = Nombre;
        }
        txtNombre.Focus();
    }
}

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 pre-visualización:

## Listado de Asientos Reservados

Transporte: Coche Semi Cama 56A

Fecha: 18/03/2023

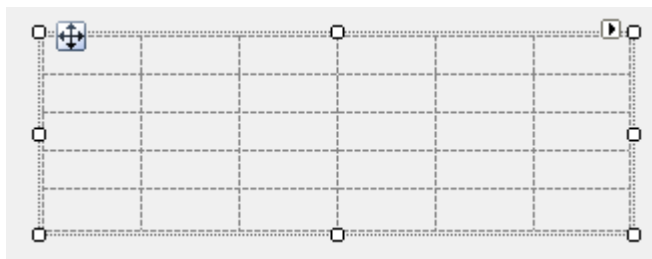


Asiento Nro	Nombre Pasajero	Asiento Nro	Nombre Pasajero
1	Carlos Oviedo	46	Lisa Alvarez
2	Carlos Oviedo	47	Julio Boca
4	Jose Palacios	49	Carla Devito
6	Cacho Parada	50	Carla Devito
7	Carlos Oviedo		
8	Lucía Agostino		
9	Lucía Agostino		
14	Sergio Blanco		
15	Pedro Fuentes		
16	Pedro Fuentes		
17	Sofía Ludueña		
18	Sergio Blanco		
20	Sandra Andrada		
21	Sergio Blanco		
23	Sandra Andrada		
24	Sandra Andrada		
26	Sandra Marsani		
29	Sandra Marsani		
31	Pablo Lendo		
32	Pablo Lendo		
34	Marcela Mercado		
35	Pablo Lendo		
36	Pablo Lendo		
37	Marcela Mercado		
38	Marcela Mercado		
43	Hugo Pisano		
44	Hugo Pisano		
45	Lisa Alvarez		

## 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.

**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      X

Falso

**2. Indique la opción correcta**

El objeto “Graphics” posee los métodos para controlar la impresión de texto y gráficos.

Verdadero

Falso      X

**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      X

**4. Indique la opción correcta**

El control de tipo FlowLayoutPanel se puede usar para distribuir otros controles y organizar el contenido del formulario.

Verdadero      X

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      X



**6. Indique la opción correcta**

Para iniciar el proceso de impresión de un objeto PrintDocument se debe ejecutar el método:

PrintDocument

Print ☒ X

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 ☒ X

Falso