Programación Visual en C# .NET

Guion de la práctica 5

OBJETIVOS

- Aplicaciones que soportan múltiples documentos
- Barras de herramientas y de estado
- Uso de imágenes

TEMPORIZACIÓN

Recogida del enunciado:	Semana del 26 de octubre
Desarrollo de la práctica:	**2 semanas**
Fecha de entrega:	Semana del 16 de noviembre junto con la práctica 6
Fecha límite de entrega:	Semana del 23 de noviembre

PRÁCTICA 5

MDI

TABLA DE CONTENIDOS:

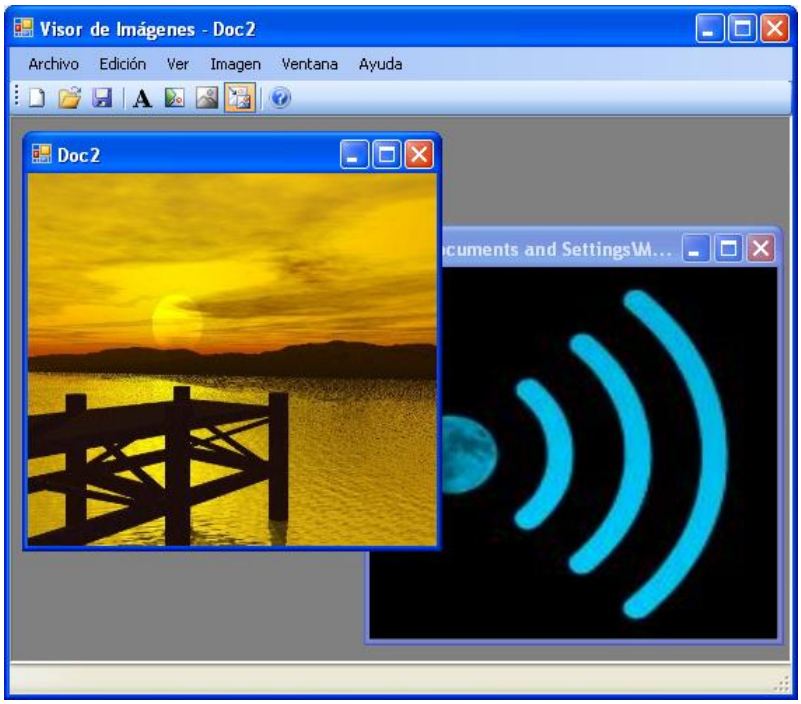
Introducción **3**

9. Esqueleto de una aplicación MDI **4**

9.1 Crear una ventana hija nueva 4
9.2 Cargar una imagen desde un recurso 6
9.3 Ajustar la imagen a la ventana hija 6
9.4 Rotar la imagen 90º 7
9.5 Añadir texto a la imagen 7
9.6 Convertir la imagen a escala de grises 8
9.7 Abrir y guardar imágenes 9
9.8 Añadir soporte para "arrastrar y soltar" 11
9.9 Barras de herramientas y de estado 12

Introducción

Se creará una nueva aplicación MDI que mostrará imágenes en sus ventanas hijas:



El programa podrá mostrar las imágenes en su resolución original (con barras de desplazamiento si fueran necesarias), o adaptadas al tamaño de la ventana hija. Además, será posible realizar transformaciones a las imágenes, tales como añadir una rotación de 90º o un texto predefinido.

Las imágenes se cargarán desde los recursos del propio programa, o bien desde archivos bmp, jpg, gif, etc., durante la ejecución. También se permitirá guardar las imágenes.

9. Esqueleto de una aplicación MDI

Una vez haya creado un nuevo proyecto, cambie el nombre del formulario principal por `VisorImágenes`. Para que este formulario sea capaz de gestionar múltiples ventanas hijas, es necesario activar su propiedad **IsMdiContainer.** Hecho esto, cree la barra de menús principal con los menús "Archivo", "Ver", "Ventana" y "Ayuda".

La barra de menús posee una propiedad llamada **MdiWindowListItem** donde deberá introducir el identificador del menú "Ventana", para que dicho menú pueda mostrar automáticamente una lista con las ventanas hijas existentes e indicar con una señal cuál es la activa.

Para crear el diálogo *Acerca de*, puede hacer uso de la plantilla existente **AboutBox** (*Agregar Windows Forms… > Cuadro Acerca de*). A continuación, añada la orden "Acerca de..." correspondiente al menú "Ayuda" y su manejador del evento **Click**.

Por último, añada a "Ventana" las órdenes "Cascada", "Mosaico horizontal" *y* "Mosaico vertical" y asócieles los siguientes manejadores:

```
private void VentanaCascada_Click (object sender, EventArgs e)
{
   this.LayoutMdi(MdiLayout.Cascade);
}

private void VentanaMosaicoHorizontal_Click(object sender, EventArgs e)
{
   //. . .
}

private void VentanaMosaicoVertical_Click(object sender, EventArgs e)
{
   //. . .
}
```

Agregue al menú "Archivo" las órdenes "Salir" (cerrará el programa en su totalidad), "Cerrar" (cerrará la ventana hija activa) y "Nuevo" (creará una nueva ventana hija). Los manejadores de las dos últimas órdenes los implementaremos posteriormente, mientras que el de "Salir" lo puede implementar ya.

Compilar y ejecutar para ver el resultado.

9.1 Crear una ventana hija nueva

Añada un manejador para el evento **Click** de la orden "Nuevo" del menú "Archivo", y modifíquelo como se muestra a continuación:

```
private void ArchivoNuevo_Click(object sender, EventArgs e)
{
   int númeroHijas = this.MdiChildren.Length;
   string título = "Doc" + (númeroHijas+1);
   NuevaHija(título);
}
```

Haga clic con el botón derecho sobre la llamada a `NuevaHija`, y seleccione la opción *Generar código auxiliar del método*. Modifique el método generado según se indica a continuación:

```
private void NuevaHija(string título)
{
   VentanaHija hija = new VentanaHija(título);
   hija.MdiParent = this;
   hija.PictureBox.SizeMode = PictureBoxSizeMode.AutoSize;
   hija.AutoScroll = true;
   hija.Show();
}
```

La clase `VentanaHija` no ha sido creada aún. Para hacerlo, vaya al menú *Proyecto* y seleccione la opción *Agregar Windows Forms…> Windows Forms*. Introduzca "VentanaHija.cs" como nombre para el nuevo archivo de código. Arrastre un control `PictureBox` sobre el formulario de nombre `m_PictureBox` y asigne a su propiedad `Location` el valor (0,0). Añada la siguiente propiedad a la clase `VentanaHija` recién creada:

```
public PictureBox PictureBox
{
   get { return m_PictureBox; }
}
```

A continuación, modifique el constructor de `VentanaHija` para que tome un parámetro `título` de tipo `string`. Dentro del cuerpo de dicho constructor, añada la siguiente línea tras la llamada a `InitializeComponent`:

```
this.Text = título;
```

Compilar y ejecutar para ver el resultado.

Añada la siguiente propiedad pública a la clase `VisorImágenes`:

```
public VentanaHija HijaActiva
{
   get { return (VentanaHija)this.ActiveMdiChild; }
}
```

Esta propiedad nos permite acceder rápidamente a la ventana hija activa, y además realiza la conversión *cast* a la clase `VentanaHija` de forma automática. Añada ahora el siguiente manejador a la orden *Archivo* > *Cerrar*, que muestra cómo se utiliza la propiedad que acabamos de añadir:

```
private void ArchivoCerrar_Click(object sender, EventArgs e)
{
   this.HijaActiva.Close();
}
```

Sepa que el evento **MdiChildActivate** del formulario principal nos avisa cuando el usuario cambia la ventana hija activa.

Compilar y ejecutar para ver el resultado.

9.2 Cargar una imagen desde un recurso

Añada ahora un recurso a la aplicación (*Proyecto* > *Propiedades…*). A continuación, utilizando el propio editor de Visual Studio añada una nueva imagen (*Agregar recurso*), o bien una imagen existente, que se llamará, por ejemplo, `Imagen1` (esto crea una propiedad estática `Imagen1` en la clase `Properties.Resources`). Un buen estilo de programación sugiere que añada al proyecto una nueva carpeta *Imágenes* para guardar todas las imágenes.

A continuación, haga que al crear una nueva ventana se cargue automáticamente la imagen del recurso:

```
this.HijaActiva.PictureBox.Image = Properties.Resources.Imagen1;
```

La línea de código anterior asigna al control **PictureBox** de la ventana activa una imagen obtenida a partir de los recursos de la aplicación.

Compilar y ejecutar para ver el resultado.

9.3 Ajustar la imagen a la ventana hija

En este apartado añadiremos a la aplicación la posibilidad de mostrar la imagen redimensionada para adaptarse al tamaño de la ventana. Esta opción será accesible a través de un menú "Imagen" que estará asociado a cada ventana hija, pero que se mostrará fusionado con el menú principal.

Desde la vista de diseño del formulario `VentanaHija`, añada un control `MenuStrip` con la propiedad **Visible** desactivada. Añada a esta barra de menús un menú "Imagen" con una orden llamada "Ajustar a ventana".

Compilar y ejecutar para ver el resultado.

Al ejecutar la aplicación puede ver que la barra de menús de la ventana hija se integra con la barra de menús del formulario principal, y que desaparece cuando no existe ninguna ventana hija. Sin embargo, existe un pequeño problema ya que el menú "Imagen" aparece a la derecha de los menús "Ventana" y "Ayuda", lo cual no es el comportamiento habitual de las aplicaciones de Windows. Para solucionar esto, seleccione el menú "Imagen" y cambie su propiedad **MergeAction** por **Insert**, y su propiedad **MergeIndex** por 2.

Compilar y ejecutar para ver el resultado.

Asocie el siguiente manejador con el evento **Click** de la orden "Ajustar a ventana":

```
private void ImagenAjustar_Click(object sender, EventArgs e)
{
   // Si no estamos en modo ajustar, activamos este modo
   if (this.PictureBox.SizeMode ==
                   PictureBoxSizeMode.AutoSize)
   {
      this.AutoScroll = false;
```

```
            this.PictureBox.Size = this.ClientSize;
            this.PictureBox.SizeMode = PictureBoxSizeMode.Zoom;
        }
        // Si estamos en modo ajustar, desactivamos este modo
        else if (this.PictureBox.SizeMode ==
                        PictureBoxSizeMode.Zoom)
        {
            this.AutoScroll = true;
            this.PictureBox.SizeMode = PictureBoxSizeMode.AutoSize;
        }
    }
```

¿Qué diferencia hay entre utilizar `Zoom` o `StretchImage`? Pruébelo.

Compilar y ejecutar para ver el resultado.

¿Qué pasa cuando selecciona la opción "Ajustar a ventana" y cambia el tamaño de la ventana hija? Solucione este problema añadiendo un manejador al evento `Resize` de la clase `VentanaHija`. Modifique este manejador como se indica:

```
private void VentanaHija_Resize(object sender, EventArgs e)
{
  if (this.PictureBox.SizeMode == PictureBoxSizeMode.Zoom)
     this.PictureBox.Size = this.ClientSize;
}
```

Compilar y ejecutar para ver el resultado.

Es importante dar información al usuario sobre si el modo "Ajustar" está activado en un determinado momento. Para ello, añada la siguiente línea al final del manejador `ImagenAjustar_Click` (se supone que ha cambiado la propiedad **Name** de la orden "Ajustar a ventana" por "ImagenAjustar"):

```
ImagenAjustar.Checked =
     this.PictureBox.SizeMode == PictureBoxSizeMode.Zoom;
```

9.4 Rotar la imagen 90º

Añada una orden al menú "Imagen" que permita rotar la imagen 90º cada vez que sea ejecutada, independientemente de si ésta está ajustada o no. Use para ello el método `RotateFlip` de la clase `Image`. No olvide llamar al método `Refresh` del **PictureBox** cuando haya finalizado la rotación, para repintarlo correctamente.

Compilar y ejecutar para ver el resultado.

9.5 Añadir texto a la imagen

Añada una nueva orden llamada "Añadir texto" al menú "Imagen". Asóciele el siguiente manejador:

```
private void ImagenAñadirTexto_Click(object sender, EventArgs e)
{
```

```
    using(Graphics gfx = Graphics.FromImage(this.PictureBox.Image))
    {
       // Utilizar el método DrawString de gfx para pintar el texto
       // "Programación Visual" en la posición 0,0 del área de
       // cliente, utilizando una fuente Arial de 20 ptos
       // y una brocha del color deseado.
    }

    // Refrescar el picture box
}
```

El método estático `FromImage` de la clase `Graphics` nos proporciona la superficie de dibujo para dibujar sobre la imagen especificada.

Compilar y ejecutar para ver el resultado.

9.6 Convertir la imagen a escala de grises

Como hemos visto en el apartado anterior, es fácil realizar transformaciones sencillas sobre una imagen empleando un objeto `Graphics` creado a partir de ella. Otra transformación interesante es convertir la imagen a un espacio de color diferente. Para ello hay que crear una matriz de transformación de 5x5 y redibujar la imagen usando el método `DrawImage` y la matriz de transformación. Para poner esto en práctica, añada una nueva orden "Escala de grises" al menú "Imagen" y asóciele el siguiente manejador:

```
private void ImagenEscalaDeGrises_Click(object sender, EventArgs e)
{
   PictureBox pictureBox = this.PictureBox;
   Image imagen = pictureBox.Image;

   using(Graphics gfx = Graphics.FromImage(imagen))
   {
      // Matriz para realizar una transformación al gris
      // manteniendo los valores de luminancia correctos
      ColorMatrix cm = new ColorMatrix(new float[][]{
                    new float[]{0.3f,0.3f,0.3f,0,0},
                    new float[]{0.59f,0.59f,0.59f,0,0},
                    new float[]{0.11f,0.11f,0.11f,0,0},
                    new float[]{0,0,0,1,0},
                    new float[]{0,0,0,0,1}});
      // Información acerca de la manipulación de los colores del
      // mapa de bits
      ImageAttributes atrImg = new ImageAttributes();
      atrImg.SetColorMatrix(cm);

      // Dibujar la imagen:
      // DrawImage(Imagen, RectImgDestino,
      //   XImgFuente, YImgFuente, anchoImgFuente, anchoImgFuente,
      //   UnidadesGráficas, AtributosImagen)
   }
   // Refrescar pictureBox
}
```

Para que este código funcione tendrá que hacer visible el espacio de nombres **System.Drawing.Imaging**, al que pertenecen la clase `ColorMatrix` e `ImageAttributes`.

Compilar y ejecutar para ver el resultado.

9.7 Abrir y guardar imágenes

Añada a la aplicación la funcionalidad necesaria para que desde la orden "Abrir" del menú "Archivo" se pueda cargar cualquier fichero de imagen (incluya al menos soporte para los formatos `bmp`, `jpg` y `gif`). Esto podría hacerse mediante un manejador con la siguiente estructura:

```
private void ArchivoAbrir_Click(object sender, EventArgs e)
{
   // Mostrar diálogo OpenFileDialog

   // Mediante la propiedad Filter indicamos el tipo de archivos
   // que se permiten abrir y mediante Title especificamos
   // un título explicativo para la ventana.

   // Si el resultado del diálogo es distinto de OK, terminar.

   byte [] contenidoArchivo = File.ReadAllBytes(rutaArchivo);

   // Creamos un flujo de tipo MemoryStream pasándole el contenido
   // del archivo.
   // Usamos el método FromStream de la clase Image para crear una
   // "imagen" a partir del flujo anterior.
   // Creamos una nueva ventana hija con el método NuevaHija,
   // pasándole como título el nombre del archivo abierto.
   // (dlg.FileName)
   // Obtenemos una referencia a la nueva ventana mediante la
   // propiedad HijaActiva.
   // Asignamos "imagen" al picture box de la nueva ventana
}
```

Compilar y ejecutar para ver el resultado.

Añada un menú "Archivo" a la barra de menús del formulario `VentanaHija` (mismo identificador y título que en la ventana padre). Cambie su propiedad **MergeAction** al valor **MatchOnly**. Esto hará que se combine con el menú "Archivo" del formulario principal. Agregue al nuevo menú dos órdenes: "Guardar" y "Guardar como…", que se encargarán de guardar en disco la imagen mostrada en la ventana activa, y un separador. Cambie la propiedad **MergeAction** de estos tres elementos por **Insert**, y su propiedad **MergeIndex** por 3, 4 y 5, respectivamente, para que se inserten en el lugar apropiado del menú "Archivo" del formulario principal.

Utilice el siguiente esquema para el manejador de "Guardar como…":

```
private void ArchivoGuardarComo_Click(object sender, EventArgs e)
{
   // Mostrar el diálogo SaveFileDialog y configurarlo de forma
   // análoga al OpenFileDialog de la opción "Abrir".
```

```
    // Si el resultado del diálogo es distinto de OK, terminar.

    if (dlgGuardar.FileName.ToUpper().EndsWith(".JPG"))
      // Usar el método Save para guardar la imagen  en formato jpg.

    //Análogamente con el resto de formatos permitidos.

    // Actualizar el título de esta ventana.
}
```

Respecto al manejador de "Guardar", debe tener en cuenta que si el título de la ventana es *DocN* entonces la imagen no ha sido guardada con anterioridad, por lo que el proceso es análogo al de "Guardar como…". En cambio, si el título de la ventana ya es el de una ruta de un archivo, simplemente hay que guardar la imagen en esa ruta:

```
private void ArchivoGuardar_Click(object sender, EventArgs e)
{
    if (título de la ventana hija activa comienza con "Doc")
      // Mismo código que "Guardar como..."
    else
      // La guardamos en el mismo archivo de donde la leímos.
}
```

Para evitar tener código duplicado, utilice la herramienta de refactorización para convertir el código del manejador de "Guardar como…" en un nuevo método `GuardarComo` al que llamen tanto "Guardar como…" como "Guardar".

Compilar y ejecutar para ver el resultado.

Ejecute su aplicación y seleccione, sin haber abierto ninguna ventana, la orden "Archivo > Cerrar". ¿Qué pasa? El manejador de dicha orden espera que exista una ventana activa sobre la que trabajar, pero en este caso no la hay (la propiedad `ActiveMdiChild` vale *null*). Como resultado, el entorno de ejecución lanza una excepción de tipo `NullReferenceException`. Para evitar esto existen dos enfoques posibles. Uno de ellos pasa por comprobar, en los métodos de la clase `VisorImágenes` en los que sea oportuno, si existe una ventana activa. Así, por ejemplo, podríamos añadir al comienzo del manejador de la orden "Archivo > Cerrar" el siguiente código:

```
private void ArchivoCerrar_Click(object sender, EventArgs e)
{
    if(this.HijaActiva == null)
    {
       MessageBox.Show("No hay ninguna ventana activa","Error",
                       MessageBoxButtons.OK,MessageBoxIcon.Error);
       return;
    }
    // Lo demás igual que antes
}
```

Otro enfoque consiste en no mostrar al usuario las órdenes que trabajan sobre la ventana activa cuando dicha ventana no existe, o mostrarlas inhabilitadas. Un buen sitio para hacer esto es el manejador del evento **MdiChildActivate,** que es invocado cada vez que se cambia la ventana hija activa. Por ejemplo, para conseguir que la orden "Archivo >

Cerrar" aparezca inhabilitada cuando no exista ninguna ventana sobre la que trabajar, añada el siguiente código al final de dicho manejador:

```
menúArchivoCerrar.Enabled = this.HijaActiva != null;
```

Deberá hacer lo propio con el resto de opciones de menú que no tengan sentido si no existe una ventana activa (por ejemplo, "Ventana > Mosaico"). Use la herramienta de refactorización para extraer todo este código en un nuevo método llamado `ActualizarMenús`, y llame a este método también desde el constructor para que inicialmente los menús aparezcan en el estado correcto.

Compilar y ejecutar para ver el resultado.

9.8 Añadir soporte para "arrastrar y soltar"

En una interfaz gráfica de usuario, llamamos "arrastrar y soltar" (*drag-and-drop*) al proceso por el cual el usuario hace clic sobre un objeto y lo arrastra encima de otro. Un ejemplo muy común es arrastrar el icono de un archivo sobre una aplicación para abrirlo con ella. Añadiremos esta funcionalidad de forma que el usuario pueda seleccionar una imagen en el explorador de Windows y arrastrarla sobre nuestra aplicación para visualizarla.

Para que la aplicación acepte el arrastre de archivos, active la propiedad **AllowDrop** del formulario principal. A continuación, añada el siguiente manejador para el evento **DragEnter** de dicho formulario:

```
private void VisorImágenes_DragEnter(object sender, DragEventArgs e)
{
   // Nos aseguramos de que lo que se está arrastrando son archivos
   if(!e.Data.GetDataPresent(DataFormats.FileDrop))
   {
      e.Effect = DragDropEffects.None;
      return;
   }

   string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

   foreach (string file in files)
   {
      if (!file.ToUpper().EndsWith(".JPG") &&
          !file.ToUpper().EndsWith(".BMP") &&
          !file.ToUpper().EndsWith(".GIF"))
      {
         e.Effect = DragDropEffects.None; // Alguno de los archivos
                                          // no es una imagen
         return;
      }
   }
   e.Effect = DragDropEffects.Copy; // Correcto, son todo imágenes
}
```

Este manejador nos permite seleccionar qué tipos de objetos se pueden arrastrar sobre nuestra aplicación (en este caso, archivos de extensión `jpg`, `bmp` o `gif`). Si el usuario

arrastra un objeto que no corresponde a esta descripción, el cursor adopta la forma de un signo de prohibición y la operación no se puede finalizar.

Compilar y ejecutar para ver el resultado.

Añada a continuación el siguiente manejador del evento **DragDrop**, que se encarga de abrir los archivos arrastrados una vez que el usuario ha soltado el botón del ratón:

```
private void VisorImágenes_DragDrop(object sender,
                                    DragEventArgs e)
{
   string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
   foreach (string file in files)
      AbrirArchivo(file);
}
```

Deberá añadir el método `AbrirArchivo`, que tendrá una implementación similar a la de las últimas líneas del manejador de la orden "Archivo > Abrir". Para evitar duplicar código, genere este método usando la herramienta de refactorización *Extraer método*.

Compilar y ejecutar para ver el resultado.

9.9 Barras de herramientas y de estado

Cree una barra de herramientas en el formulario principal y agréguele botones asociados a las funciones más importantes del menú principal ("Nuevo", "Abrir", "Acerca de"). Para estos botones, especifique los mensajes que deben aparecer al poner el ratón sobre ellos (propiedad **Text**). Nota: una forma sencilla de asociar una imagen a un botón de la barra de herramientas es crearla (o copiarla) con un programa de dibujo (por ejemplo, MS Paint) y a continuación importarla; para ello, seleccione la propiedad **Image** del botón en la ventana de propiedades y seleccione un nuevo valor: *Recurso local* > *Importar* > seleccione la imagen de la carpeta *Imágenes*.

Con el fin de evitar la duplicación de código, asocie a cada botón de la barra de herramientas el manejador del evento **Click** que ya creó para la correspondiente orden de menú.

Compilar y ejecutar para ver el resultado.

Como vimos en un apartado anterior, el menú "Imagen" y la parte del menú "Archivo" correspondiente a las órdenes de guardar son especiales, ya que pertenecen a la ventana hija activa, pero aparecen integrados con el menú principal. Sus correspondientes botones de barra de herramientas se comportarán de forma análoga. Para ello, añada una barra de herramientas al formulario `VentanaHija`. Desactive la propiedad **Visible** de esta barra de herramientas, y añada botones para las órdenes "Guardar", "Ajustar a ventana", "Añadir texto", "Rotar" y "Escala de grises". Inserte un separador entre las dos primeras y asocie a cada botón el manejador de la opción de menú correspondiente.

Para que los botones de la nueva barra de herramientas se inserten en el lugar apropiado de la barra principal tendrá que cambiar su propiedad **MergeAction** por **Insert**. Cambie también su propiedad **MergeIndex** dándole a cada botón un valor que permita que se

inserte donde le corresponde (ver figura siguiente). A diferencia de las barras de menús, la fusión de las barras de herramientas no es automática, sino que hay que llamar explícitamente al método `Merge` de `ToolStripManager`. El sitio más apropiado para realizar esta operación es el manejador del evento **MdiChildActivate** del formulario MDI:

```
private void VisorImágenes_MdiChildActivate(
object sender, EventArgs e)
{
  // ...
  ToolStripManager.RevertMerge(this.barraHerramientas);
  VentanaHija ventanaHijaActiva =
                              this.ActiveMdiChild as VentanaHija;
  if (ventanaHijaActiva != null)
    // invocar a Merge
}
```

Compilar y ejecutar para ver el resultado.

Pruebe ahora a quitar la llamada al método **RevertMerge** y compare los resultados.

No olvide que el botón correspondiente a la orden "Ajustar a ventana" debe aparecer marcado (**Checked**=*true*) cuando esta opción se encuentre activada para la ventana hija activa.

Compilar y ejecutar para ver el resultado.

A continuación, añadirá una barra de estado. Para ello, añada un control **StatusStrip** al formulario `VisorImágenes` desde la caja de herramientas. A continuación, coloque en ella una etiqueta (control **StatusLabel**) llamada "etiquetaEstado". Active la propiedad **Spring** de esta etiqueta, y cambie su propiedad **TextAlign** a **MiddleLeft**.

Haciendo uso de los eventos **MouseEnter** y **MouseLeave** de los menús, botones, etc., muestre la información que desee en la barra de estado (por ejemplo, para la orden de menú "Archivo > Nuevo" y para su correspondiente botón de la barra de herramientas puede mostrar el texto "Crea una nueva ventana con una imagen predefinida".

Compilar y ejecutar para ver el resultado.

Para finalizar deberá agregar al menú "Ver" órdenes que permitan mostrar u ocultar la barra de estado y la barra de herramientas. Estas órdenes deberán aparecer marcadas (**Checked**=*true*) cuando la barra correspondiente esté visible (pista: use el evento **DropDownOpened** del menú "Ver").

Compilar y ejecutar para ver el resultado.