A. Introducción

Existe una marcada diferencia entre los proyectos que tienen o no interfaz de usuario. En este capítulo, vamos a presentar la interfaz de una aplicación de Windows. Vamos a ver en primer lugar las herramientas que permiten dibujar esta interfaz. Luego, después de una primera aplicación, introduciremos el concepto de delegate y evento, que son utilizados generalmente para aplicar los comportamientos indirectos. Después de haber probado nuestro evento, utilizaremos el **BackgroundWorker**, nuevo componente que facilita la utilización del thread de trabajo en una aplicación para Windows.

Realizaremos una síntesis de nuestra aplicación para lograr un método de despliegue y solucionar los problemas relacionados con él. Esta síntesis nos llevará a utilizar el concepto de **Smart Client**.

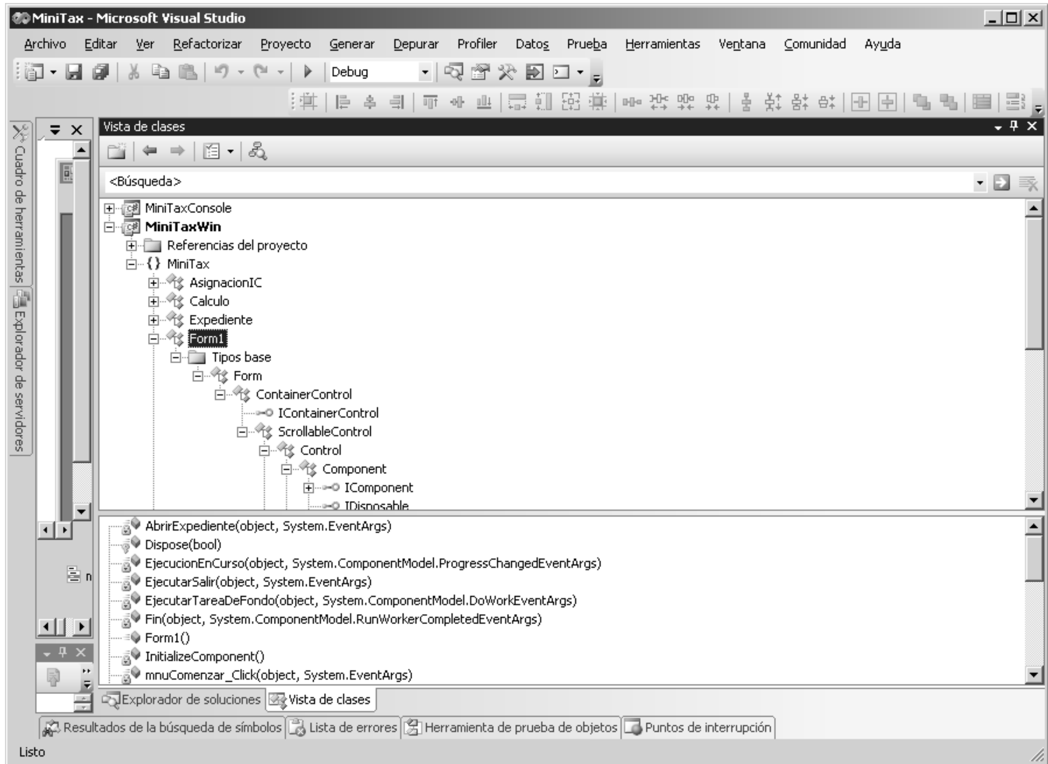
B. Los formularios

Una aplicación para Windows es una aplicación basada en una o más ventanas que generan eventos por medio del sistema operativo. Este último va interceptar los eventos materiales y a transmitirlos a la cola de mensajes del thread (hilo de ejecución del código) de la ventana activa en el momento en que el evento se produce.

Por eso, después de haber aceptado la elección de crear un proyecto de tipo aplicación para Windows, Visual Studio presenta un formulario en modo Diseño en la parte edición del entorno. La clase **Form** representa el modelo que se debe utilizar para crear una ventana Windows.

© Editions ENI - All rights reserved

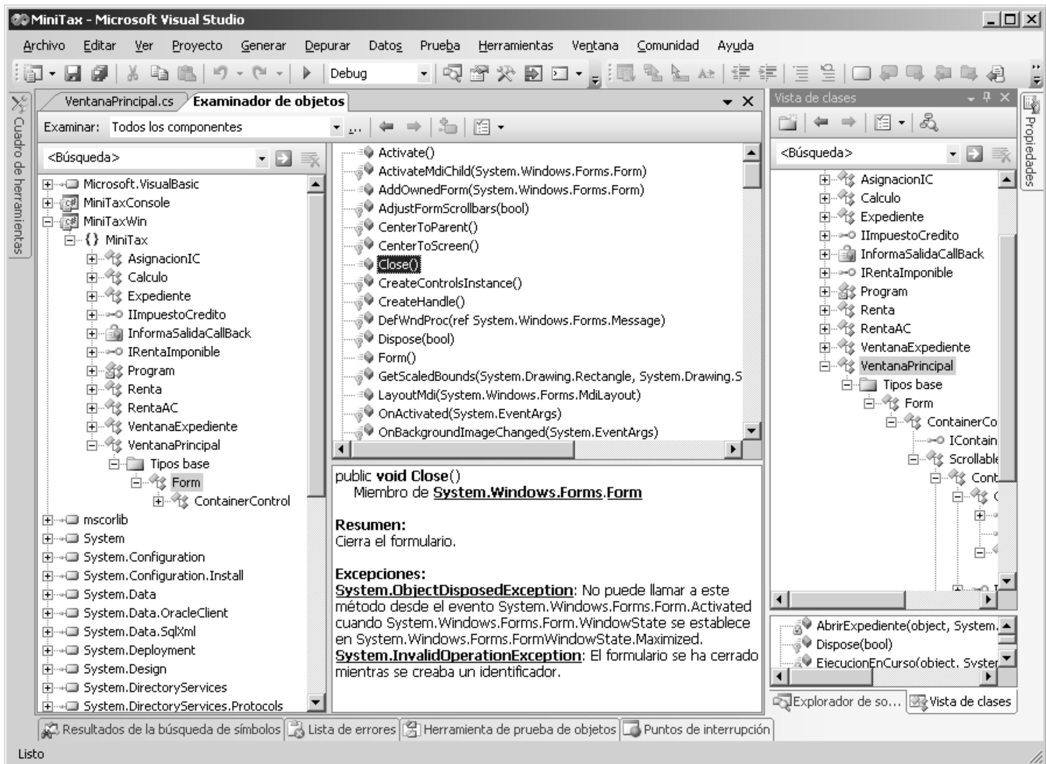
1. La clase Form



Visual Studio dispone de una **Vista de clases** en las ventanas de la derecha. Al abrir (clic en el signo +), la jerarquía de **Form1**, que es hasta el momento el nombre otorgado al único formulario de nuestro proyecto, podemos observar los nombres de tipos que sirven de base a nuestra clase. En particular, la clase **Control** que muestra la pertenencia de la clase **Form** a la familia de los componentes de Windows, y la clase **ContainerControl**, que pone de manifiesto que nuestro formulario va a poder incorporar otros componentes visuales.

Encontramos, en la lista de la parte inferior de la visualización de clases, el conjunto de los métodos y propiedades. Los iconos delante de los nombres permiten distinguir los métodos (cubo), las propiedades (página+menú), los eventos (relámpago). Una pequeña llave agregada significa que el elemento en cuestión está modificado por la palabra clave `protected`. Eso quiere decir que, para utilizarlo, es necesario ser de una clase derivada. Es el caso de nuestra clase **Form1**.

Si un método le interesa, con realizar un doble clic en él es suficiente para provocar la visualización del **Examinador de objetos** en la parte central de Visual Studio. Esta nueva visualización puede aportarle un poco más de información sobre la estructura de las clases que utiliza y de los elementos más detallados, como, por ejemplo, las opciones de una enumeración o el prototipo de una función, su resumen y las posibles excepciones que van a lanzarse.



© Editions ENI - All rights reserved

Antes de ver con detalle los usos posibles y los componentes que se pueden añadir a un formulario, vamos a observar algunas propiedades esenciales de nuestra nueva clase derivada de la clase Form.

La ventana **Propiedades** (ventana Herramienta a la derecha) permite modificar el comportamiento a partir del diseño de nuestro formulario.

La propiedad **Name** es importante, ya que permite el cambio de nombre de la clase representada en la pantalla por el formulario en modo Diseño. En efecto, todo lo que se representará en la pantalla será creado por el código aplicado por una clase derivada de Form.

La propiedad **Text** permite cambiar el contenido indicado en la mayoría de los controles, y, en particular, la barra de título de una ventana.

Es posible crear aplicaciones SDI (*Single Document Interface*) y MDI (*Multiple Document Interface*). Es también factible tener varios contenedores MDI en una aplicación. Para aclarar este vocabulario, podríamos comparar una aplicación SDI con otra muy conocida: Notepad, y una aplicación MDI con Word o Excel. Cuando hablamos de contenedor, se trata de la ventana que va a contener una serie de ventanas hijas.

La propiedad **IsMdiContainer** va a utilizarse para precisar el estado buscado. La posibilidad de crear varios contenedores en una aplicación introduce una nueva lógica a la ya conocida en otros entornos de desarrollo. Va a ser necesario configurar la ventana hija durante la instanciación para decir en qué contenedor se quiere colocarla. Por ejemplo:

```
Form f = new VentanaExpediente() ;
f.MdiParent = this ;
f.Show() ;
```

La visualización de un formulario se hace con el método **Show**.

2. Los cuadros de diálogo y la herencia de formularios

a. La visualización de un formulario en modal

El método **Show** tiene varias formas y permite elegir el nivel de posesión de una ventana, pero no es el método indicado para un cuadro de diálogo.

Un cuadro de diálogo está caracterizado por un comportamiento y sus atributos visuales.

Los atributos visuales son: un borde grueso y un fondo gris (el color por defecto de Windows). No tienen, en general, posibilidad de modificación y es posible desplazar el cuadro haciendo un clic/arrastrando después de haber hecho clic en el título.

Un cuadro de diálogo tiene su propio bucle de mensajes de Windows, interpretando los clics del usuario. Eso se traduce en la imposibilidad para el usuario de pulsar fuera del cuadro para activar otra ventana de la aplicación en tanto no haya cerrado el cuadro de diálogo.

Es el método **ShowDialog** el que permite abrir un formulario pasando el control de los mensajes a un bucle interno. Se habla de ejecución modal.

Algunas propiedades tienen un sentido particular en este contexto, como **AcceptButton**, que permite definir un botón cuyo comportamiento se activará como si se hubiera hecho clic en él o como si el usuario hubiera pulsado la tecla [Enter]. **CancelButton** tiene también estas posibilidades de asociación para la utilización de la tecla [Esc]. **FormBorderStyle** permite precisar el formato visual que se quiere dar.

El método **ShowDialog** devuelve un valor del tipo enumerado **DialogResult**. Una utilización clásica sería:

```
if (MiCuadro.ShowDialog() == DialogResult.OK)
{
// Realizar una tarea
}
```

© Editions ENI - All rights reserved

b. Los cuadros de diálogo comunes

Los cuadros de diálogo comunes son los cuadros que todas las aplicaciones utilizan (deberían utilizar) para presentar al usuario las elecciones en cuanto a selección de archivo, fuentes, color, impresión. Las clases correspondientes se derivan de la clase **CommonDialog** y son: **FileDialog**, **FontDialog**, **ColorDialog**, **PrintDialog**, también **FolderBrowserDialog** y **PageSetupDialog**.

Como **FileDialog** es una clase abstracta, es necesario utilizar una de sus clases derivadas, **OpenFileDialog** y **SaveFileDialog**.

Sólo es necesario instanciar una de estas clases, asignar valores a sus propiedades y ejecutar ShowDialog.

c. La herencia de formularios

De la misma forma que es posible derivar una clase especializándola, es posible añadir un formulario heredado a una aplicación. La herencia en este caso es interna al ensamblado. Después de haber creado el formulario básico y haber escrito el código en relación con los eventos, debe generar el ensamblado para poder utilizar esta clase como base.

Luego elija agregar un nuevo formulario y escoger un formulario heredado. Después de haber elegido el formulario, tendrá la posibilidad de agregar controles a su formulario derivado e incluso de sobrecargar los métodos del formulario básico.

La dificultad de mantener estos formularios probablemente le limitará a los encabezados o los bordes, de modo que también se limitarán las idas y vueltas del diseño visual.