Fundamentos de las aplicaciones Windows

En el capítulo anterior, se ha presentado la jerarquía de clases que contiene el `namespace System.Windows.Forms`. En éste se estudian algunas de esas clases. Sin embargo, antes es necesario tratar en profundidad la clase `Control` porque sus métodos, propiedades y eventos serán comunes a todos los controles que deriven de esta clase.

Posteriormente, se estudiará la clase `Form` que es muy importante ya que se utiliza en todas las aplicaciones para Windows.

La clase Control

Introducción

En general, se denomina control a un objeto o instancia de una clase del `namespace System.Windows.Forms`. Sin embargo, sólo algunas de dichas clases derivan de la clase `Control`. Por ejemplo, se dice que un objeto de la clase `ColorDialog` es un control, aunque no deriva de la clase `Control`. La clase `Control` proporciona la funcionalidad básica de todos los controles que derivan de ella. Muchas de estas clases son, a su vez, clases base de otros controles (figura 15.1).

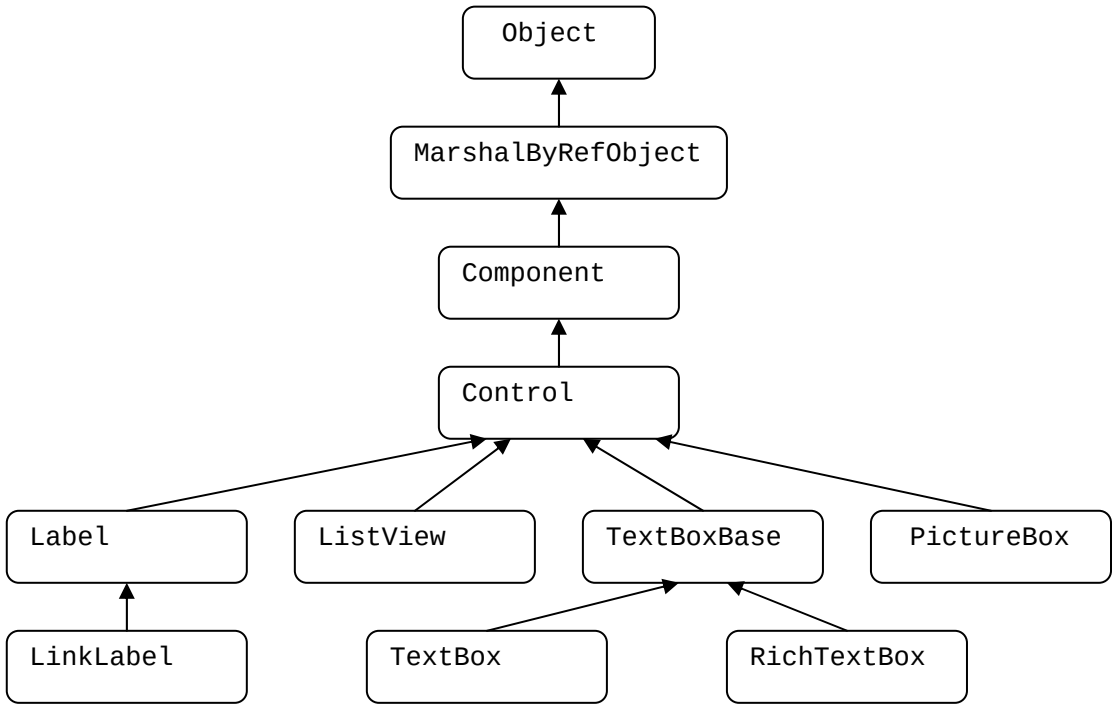

Figura 15.1: Jerarquía de la clase `Control`.

En la figura 15.2 se presenta un gráfico con los controles que proporciona, por defecto, la **caja de herramientas** de Visual Studio .NET.

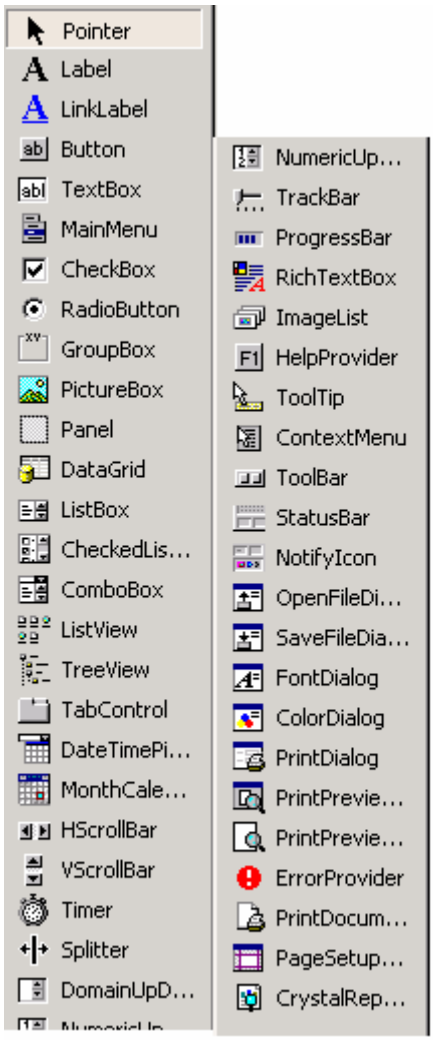

Figura 15.2: Caja de herramientas

Otro tipo distinto de controles son los que se utilizan para el diseño Web. Este tipo de controles son objetos de la clase `System.Web.UI.Control`.

Propiedades

Las propiedades que ahora se van a estudiar son comunes a la mayoría de los controles porque las heredan directamente de la clase `Control` o bien las sobrescriben.

Se explican brevemente las más utilizadas. Sin embargo, la clase `Control` tiene muchas más propiedades. Muchas de ellas se ha preferido explicarlas más adelante para no hacer en este punto una lista interminable de propiedades y eventos.

`Anchor`	Especifica el comportamiento del control cuando su "contenedor" cambia sus dimensiones.
`BackColor`	Define el color del fondo del control.
`Dock`	Permite que el control permanezca a una distancia constante –

	"anclado"- de los bordes del formulario.
`Enabled`	Especifica si un control está habilitado o no para recibir entradas del usuario. Cuando se inhabilita se dibuja en tonos grises.
`ForeColor`	Color del texto que muestra el control.
`Location (X, Y)`	Especifica las coordenadas de la esquina superior izquierda del control respecto a su contenedor.
`Name`	Nombre del control. Puede ser usado como una referencia al control.
`Parent`	Especifica el "padre" del control.
`TabIndex`	Especifica el orden o el lugar que ocupa el control al desplazarse el foco por pulsación del tabulador.
`TabStop`	Indica si un control recibirá o no el foco por la pulsación del tabulador.
`Tag`	Dato asociado al control que puede ser utilizado por el usuario.
`Visible`	Oculta o presenta el control.

Observaciones:

- Propiedades `Anchor` y `Dock`:

A veces, cuando se redimensiona una ventana o formulario dejan de verse algunos controles. Si la posición de éstos es relativa a los lados o bordes se puede provocar un auténtico caos en su distribución en el formulario. Si se diseña una aplicación multiplataforma se deseará colocar los controles del formulario en una ubicación relativa a los lados del formulario para que no haya problemas en plataformas diferentes.

.NET proporciona dos propiedades que permiten solucionar de manera sencilla este problema: `Anchor` y `Dock`**.**

`Anchor` es una propiedad que especifica cómo se comportará un control cuando el formulario que lo contiene se redimensiona. Esta propiedad determina si el control se redimensiona en proporción al cambio de dimensiones en el formulario o bien si permanece del mismo tamaño siendo constante la posición relativa respecto a algunos de los lados del formulario.

`Dock` es una propiedad vinculada a `Anchor`. Especifica si un control permanece "anclado" –a una distancia constante- de los lados del formulario. Si se redimensiona éste, el control continúa a esa misma distancia de ese lado. Más adelante, en este mismo capítulo, se hace un ejemplo que ayudará a comprender mejor estas propiedades.

Metodos

Los métodos más importantes de la clase `Control` son:

`Focus()`	Hace que el control obtenga el foco.
`Hide()`	Oculta el control.
`Show()`	Despliega el control en pantalla.

Eventos

Los evento smás importantes son:

`Click`	Ocurre cuando se pulsa sobre un control con el ratón. En algunos casos, también ocurre cuando el usuario pulsa la tecla ENTER y el control tiene el foco.
`DoubleClick`	Ocurre cuando *se pulsa dos veces consecutivas* sobre un control.
`DragDrop`	Sucede cuando *se finaliza un proceso Drag-and-Drop* (arrastrar y soltar) con el ratón sobre un control, es decir cuando un objeto ha sido arrastrado hasta un control y el usuario libera el ratón.
`DragEnter`	Ocurre cuando *se arrastra* un objeto y se *introduce* en un control.
`DragLeave`	Ocurre cuando *se sale de* un control "arrastrando" un objeto.
`DragOver`	Ocurre al realizar la operación *de arrastre sobre* el control.
`KeyDown`	Ocurre cuando tiene el foco el control y *se comienza a pulsar o presionar una tecla*. Este evento ocurre siempre antes de `KeyPress` y de `KeyUp`.
`KeyPress`	Ocurre justo *al pulsar una tecla* sobre el control que en ese momento tiene el foco. Sucede justo después de `KeyDown`, pero antes de `KeyUp`. La diferencia entre este evento y los otros dos es que mientras `KeyDown` y `KeyUp` pasan el código de la tecla que ha sido pulsada, en `KeyPress` se pasa el valor correspondiente a la tecla pulsada.
`KeyUp`	Ocurre en el control que tiene el foco cuando una tecla se libera. Ocurre siempre después `KeyDown` y `KeyPress`.
`MouseDown`	Ocurre cuando se pulsa con el ratón sobre el control. No es lo mismo que el evento `Click`, porque `MouseDown` ocurre justo cuando se pulsa el botón y antes de ser liberado.
`MouseUp`	Ocurre cuando el ratón se ha pulsado previamente sobre un control y se libera.
`MouseMove`	Ocurre continuamente cuando se mueve el ratón sobre el control.
`Paint`	Ocurre cuando se "pinta" o redibuja el control.
`Validated`	Ocurre cuando un control con la propiedad `CausesValidation` a `true` está a punto de perder el foco. Ocurre después de que el evento `Validating` finaliza e indica que se ha completado la validación
`Validating`	Ocurre cuando un control con la propiedad `CausesValidation` a `true` está a punto de perder el foco. El control que va a ser validado es el control que pierde el foco y no el que lo recibe.

Observaciones

Sobre la pulsación de una tecla:

Los eventos `KeyDown`, `KeyPress` y `KeyUp` permiten mantener un control total sobre las entradas en un control. `KeyDown` y `KeyUp` reciben como parámetro el **código** de la tecla que se ha pulsado en el teclado. Esto permite conocer si se ha pulsado algunas de las teclas especiales como `Control`, `Alt`, `Mayúscula` o alguna de las teclas de función, etc...

`KeyPress` recibe el **carácter** correspondiente a la tecla pulsada. Así, con este evento, se puede distinguir entre la letra "`b`" y la letra "`B`". Esto es muy útil cuando se pretende "filtrar" las entradas del teclado –por ejemplo, cuando se desea que las entradas en una caja de texto sean numéricas-.

En el espacio dedicado al estudio del control `TexBox` se trata con detalle estos eventos y se realizar un ejemplo.

La clase Form

Introducción

Las ventanas o formularios -forms en inglés- juegan un papel fundamental en las aplicaciones para Windows.

Desde el punto de vista del usuario, una aplicación es una ventana –formulario a partir de ahora- que se conoce como formulario o ventana principal de la aplicación junto con–en ocasiones- uno o más formularios secundarios, "ventanas hijas" o cuadros de diálogo.

Desde el punto de vista del programador, un formulario –una ventana desplegada en pantalla- no es más que un objeto de una clase que deriva de la clase `Form`. Para ejecutar una aplicación es necesario crear un objeto o instancia de esa clase. Este objeto se le pasa al método estático `Run()` de la clase `Application`. Cuando desde el cuerpo del método `Main()`se ejecuta el método `Run()`, se presenta en pantalla la ventana asociada a esta clase. Los controles de un formulario son objetos o instancias de diversas clases asociadas a diversos controles, campos de esa clase.

Propiedades

La clase `Form` deriva de las clases `Control` y de `ContainerControl` y por lo tanto heredará de ellas muchas de las propiedades y eventos estudiados anteriormente.

Las propiedades de un formulario configuran su apariencia, comportamiento y estilo. Casi todas ellas pueden definirse en tiempo de diseño en la **ventana de propiedades** del formulario o modificarse en tiempo de ejecución.

Cuando se crea un nuevo proyecto para Windows, en la fase de diseño el IDE visualiza un primer formulario que será la ventana principal de la aplicación cuando ésta se ejecute. Para modificar o definir las distintas propiedades de un formulario en tiempo de diseño, trabaje desde la **ventana de propiedades**. Se ha de tratar de tener seleccionado el formulario, porque si es otro el componente o control que tiene el foco en ese momento, se estará modificando las propiedades de ese control y no las del formulario. Esto es así porque la **ventana de propiedades** está siempre asociada al control que tiene en un determinado momento el foco. También podría seleccionar la ventana por su nombre –propiedad `Name`- a través del `ComboBox` situado en la parte superior de la **ventana de propiedades**.

Este formulario contiene un rejilla de puntos que ayuda al programador en el diseño gráfico del formulario. Las propiedades `DrawGrid`, `GridSize`, `Locked`, `SnapToGrid` configuran esa rejilla. `DrawGrid` indica si se dibuja o no el grid de posicionamiento, `GridSize` define el tamaño de ese grid-, `Locked` especifica si se puede mover o redimensionar el control- y `SnapToGrid` indica si la posición del control se ajustará al grid.

La propiedad más importante es `Name`, que es el nombre del formulario y se puede utilizar como una referencia a ese formulario.

En Microsoft Windows 2000, se puede controlar el **nivel de opacidad** o **transparencia** de cualquier ventana que se diseña. Estos formularios serán totalmente opacos cuando la aplicación se ejecute en otro sistema operativo. Para controlar la transparencia de un formulario, se ha de cambiar en la **ventana de propiedades** la propiedad `Opacity` a un valor entre `0.0` (transparencia absoluta) y `1.0` (opacidad completa).

Puede también cambiarse esta propiedad en tiempo de ejecución, asignando una determinada cantidad a esta propiedad. Por ejemplo:

```
public void MiMetodo()
{
   formularioSemiTransparente.Opacity = 0.56;
}
```

Además de éstas, se dispone de otras muchas propiedades que ayudarán al programador en el diseño de su aplicación:

- Las propiedades más importantes relacionadas con la *apariencia* del formulario son:

  - `BackColor`: es el color del fondo del formulario.
    Por defecto, el color que se elija pasa, a todos los controles que se incluyan en el formulario, ya que se considera que todos los controles "heredan" las propiedades del formulario que los contienen. Si se desea que el control tenga un color distinto, basta con cambiar el color de la propiedad `BackColor` del control.
  - `BackgroundImage`: es la imagen del fondo del control.
  - `Cursor`: es el cursor que aparece cuando el ratón se sitúa o pasa sobre el control
  - `Font`: fuente del texto del control: tipo de letra, tamaño, estilo y efectos
  - `FormBorderStyle`: Controla la apariencia del borde del formulario e incluso si es ventana de diálogo.
  - `Text`: texto del título del formulario.

Propiedades relacionadas con el *comportamiento* del formulario:

- `AllowDrop`: indica si el control recibirá notificaciones drag&drop
- `ContextMenu`: menú que se muestra al pulsar con el botón derecho sobre el formulario.
- `Tag`: dato definido por el usuario asociado al control.

Propiedades relacionadas con el *diseño* del formulario:

- `AutoScale`: Indica si se ajusta la fuente de la pantalla.
- `AutoScroll`: Indica si aparecerán las barras de desplazamiento de manera automática cuando se desplaza el cursor fuera del área cliente.
- `AutoScrollMargin`: Margen alrededor del control durante el autoscroll
- `AutoScrollMinSize`: Mínimo tamaño para la región autoscroll.

- `DockPadding`: Determina el tamaño del borde de los controles acoplados.
- `MaximunSize/MinimunSize`: Tamaños máximo y mínimo del formulario que pueden ser redimensionados.
- `Size`: Tamaño del control.
- `StartPosition`: Posición inicial del formulario.
- `WindowsState`: Estado inicial del formulario.

Propiedades relacionadas con el estilo de la ventana:

- `ControlBox`: Indica si el formulario tiene el menú de control del sistema (menú de la parte superior izquierda y los tres botones, maximizar, minimizar y cerrar)
- `HelpButton`: Indica si el formulario tiene un botón de ayuda en la barra de título.
- `Icon`: Es el icono del formulario.
- `IsMdiContainer`: Indica si el formulario es MDI.
- `MaximizedBox` y `MinimizedBox`: Indica si aparecen los botones de control de maximizar y minimizar.
- `Menu`: Es el menú asociado al formulario. Debería ser del tipo MainMenu.
- `Opacity`: Indica en tanto por cien el grado de opacidad del formulario.
- `ShowInTaskBar`: Indica si el formulario aparece en la barra de tareas.
- `TopMost`: Indica si el formulario no se oculta al perder el foco.
- `TransparenceKey`: Color que será transparente cuando se dibuje en el formulario.

Además existen otras tres importantes propiedades :

- `AcceptButton`. Indica el botón al que se traslada el evento click cuando se presiona ENTER.
- `CancelButton`: Idem con el botón ESCAPE.
- `KeyPreview`: Indica si los eventos de teclado serán registrados antes en el formulario.

Observaciones:

- **Propiedad TopMost**:

Se puede dotar a una ventana de la característica de estar siempre visible. Por ejemplo, puede ser interesante para la ayuda o para una **caja de herramientas** estar siempre visible. La propiedad `TopMost` permite que un formulario sea siempre visible.

Puede hacerse esto en tiempo de ejecución:

```
miVentanaSiempreVisible.TopMost = true;
```

- **Configuración del TabOrder (Orden de tabulación)**

Cuando un formulario o un cuadro de diálogo contienen varios controles, el usuario espera que si pulsa la tecla *Tab*, el foco se vaya desplazando de un control a otro. Para

ello, todos los controles que pueden recibir el foco tienen las propiedades *TabStop* y *TabIndex*. El primero es una propiedad booleana que indica si se desea configurar el control para recibir el foco o no. La segunda establece el orden relativo según se vaya trasladando el foco de un control a otro.

Por ejemplo:

```
texto.TabStop = true;
texto.TabIndex = 4;
```

indica que el control `texto` ocupa el quinto lugar en la secuencia del `TabControl`.

Visual Studio .NET proporciona una sencilla herramienta para configurar esta propiedad de una manera sencilla, visual y rápida. Para desplegarla, cuando se esté trabajando en la **ventana de diseño** se selecciona **Ver/Orden de Tabulación** y aparece la misma ventana pero con un pequeño número junto a cada control que indica la secuencia o el orden del foco en los controles del formulario. Para cambiar estos valores, pulse en el orden que desee. Se puede observar que los controles que pertenecen a un `GroupBox` funcionan como un grupo y con orden diferente y aparte. Para salir de la herramienta hay que volver a elegir la misma opción de menú: **Ver/Orden de Tabulación** (figura 15.3).

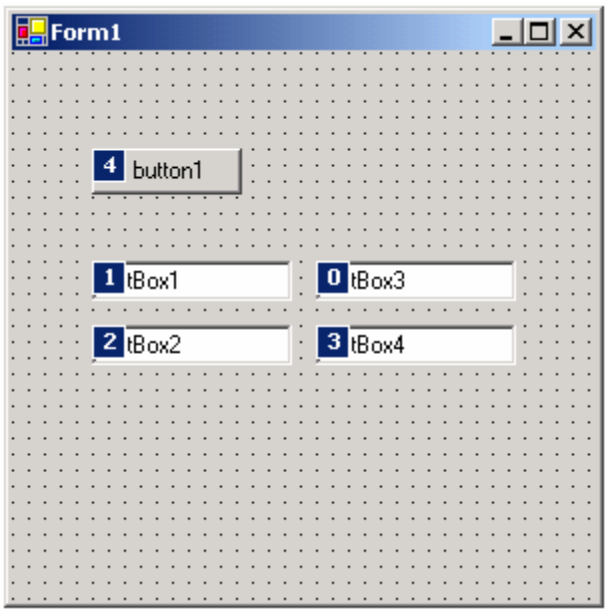

Figura 15.3 : TabOrder Wizard

Eventos

Algunos de ellos son heredados de otras clases o incluso pertenecen a otras clases. Sin embargo se señalan aquí porque son muy utilizados cuando se trabaja con formularios.

`Show()`: presenta el formulario.
`ShowDialog()`: presenrta en pantalla un formulario de dialogo modal.

`Hide()`: lo oculta.
`Close()`: lo cierra.
`Application.Exit()`: concluye la aplicación.
`Application.Run()`: visualiza el formulario que se le pasa como parámetro.

Tipos de formularios

En C# los formularios comunes –ventanas redimensionables o no- las cajas de diálogo, las cajas de herramientas, etc, son objetos o instancias de clases que derivan de la clase `Form`. Su aspecto o estilo lo define la propiedad `FormBorderStyle`. Además, esta propiedad afecta a la barra de título y a los botones de control que aparecen en ella. Los distintos valores que puede tomar esta propiedad son los siguientes:

`None`	Ningún borde especial
`Fixed3D`	Aspecto tridimensional. Puede incluir el menú de control, los botones para maximizar y minimizar en la barra de título.
`FixedDialog`	Se usa para cajas de diálogo. Los formularios con esta propiedad no son redimensionables y puede incluir el menú de control, los botones para maximizar y minimizar en la barra de título.
`FixedSingle`	Formulario no redimensionable y que puede incluir el menú de control, los botones para maximizar y minimizar en la barra de título. Sólo es redimensionable cuando se utilizan los botones de la barra de título de maximizar y minimizar. Crea un borde con una simple línea.
`FixedToolWindow`	Se usa para las ventanas de herramientas. Despliega una ventana no redimensionable con un botón `Close` y un texto en la barra de título. Este formulario no aparece en la barra de tareas de Windows.
`Sizable`	Es el valor por defecto. Es el valor más frecuente para los formularios principales. Redimensionable. Puede incluir un control de menú, los botones de maximizar y minimizar, la barra de título.
`SizableToolWindow`	Se usa para las ventanas de herramientas. Presenta una ventana redimensionable con un botón `Close` y un texto en la barra de título con una fuente reducida en tamaño. El formulario no aparece en la barra de tareas de Windows.

Todos los estilos excepto `None`, mantienen el botón `Close` en el lado derecho de la barra de título.

La elección de determinados estilos controlará la presencia de los botones `Minimize` y `Maximize` en la barra de título. Puede, no obstante, cambiarse la funcionalidad de estos botones.

Además e independientemente de lo dicho anteriormente los botones para maximizar y minimizar pueden aparecer o no en el formulario. Para ello, se deben manipular las propiedades correspondientes en la **ventana propiedades** del formulario. Así, para inhabilitar los botones de maximizar o minimizar en un determinado formulario, se ha

de escoger en la **ventana de propiedades** del formulario `MinimizeBox` o `MaximizeBox`, y asignarles `false`.

Trabajando con aplicaciones de varios formularios

A veces, en una aplicación es necesario utilizar más de un formulario –además del formulario principal-. Es muy sencillo añadir cajas de diálogo u otros formulario.

- Si se desea añadir un formulario que herede de la clase `Form`:

  En la ventana **Explorador de Soluciones**, pulse con el botón derecho del ratón en el proyecto y elija la opción **Agregar / Agregar formulario de Windows**.

- Si se quiere añadir un formulario que herede de una clase `Form` previamente creada:

  En la ventana **Explorador de Soluciones**, pulse con el botón derecho del ratón en el proyecto y elija **Agregar / Agregar formulario hererado.**

También se puede añadir un formulario desde el menú principal **Proyecto/Agregar formulario de Windows** o **Proyecto/Agregar formulario hererado.**

Cuando una aplicación se ejecuta, se presenta en pantalla el formulario que se diseña en primer lugar. Si desea que la aplicación comience con otro formulario se deben seguir los siguientes pasos:

- En la ventana **Explorador de Soluciones**, haga click con el botón derecho en el proyecto y elija **Propiedades**.
- En la ventana que se abre elija **Propiedades Comunes** / **General** y seleccione el formulario por el que se desea comenzar señalándolo en la lista que se despliega en la lista **Objeto Inicial (**ver Figura 15.4) .

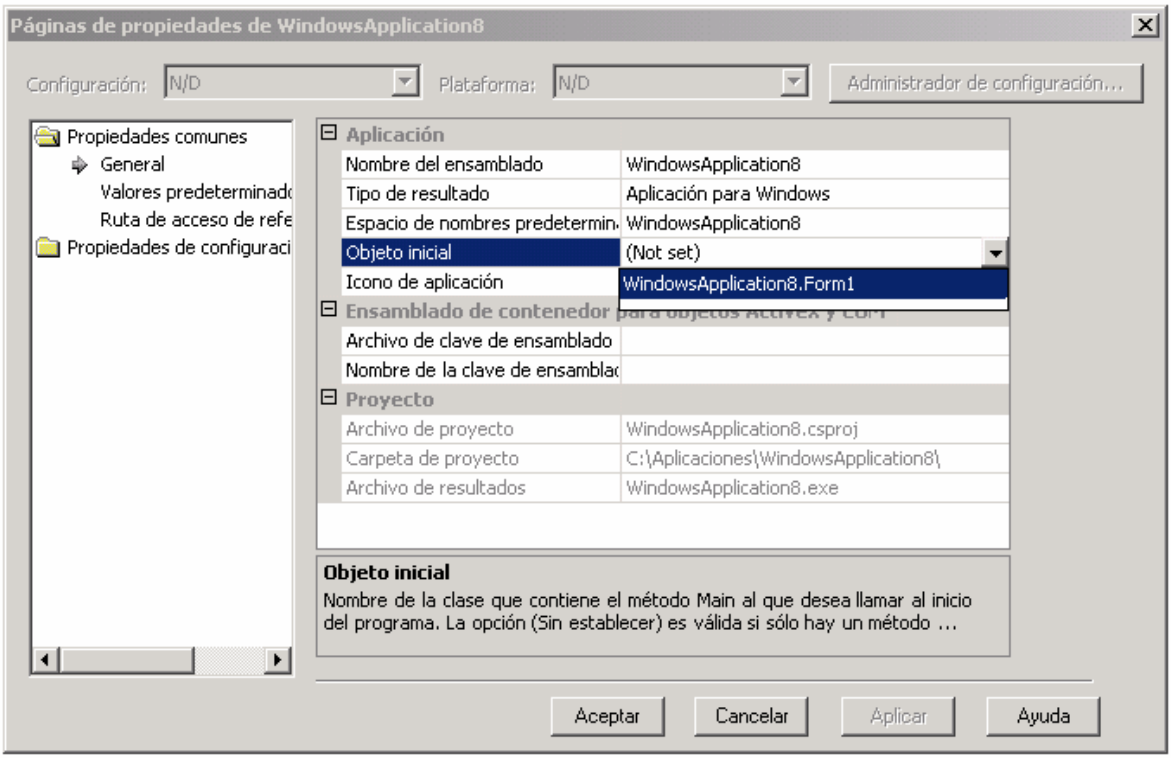

Figura 15.4: Configuración del formulario inicial de una aplicación

A veces, en el diseño de aplicaciones complejas, es difícil tener acceso a todos los formularios de un proyecto.

- Para ver los formularios del proyecto: En la ventana **Explorador de soluciones**, haga doble-click en el formulario.
- Para ver el código de un formulario: En la ventana **Explorador de soluciones**, seleccione el formulario que desee y haga click con el botón derecho y elija la opción **Ver código**, o bien, haga click con el botón derecho del ratón sobre la **ventana de diseño** del formulario y elija **Ver código**.

Formularios modales y no modales

Los formularios pueden ser *modales* o *no modales*. Un formulario o caja de diálogo *modal* es aquel que tiene que ser cerrado para continuar trabajando con la aplicación

Las cajas de diálogo que despliegan mensajes deberían ser siempre modales. Por ejemplo, la caja de diálogo **Ayuda/Acerca de Microsoft VisualStudio .NET** en el menú principal es un ejemplo de caja de diálogo modal. Los `MessageBox` son también formularios modales y serán tratados en detalle más adelante.

Los formularios *no modales* permiten trabajar entre dos formularios de la misma aplicación sin cerrarlos. Este tipo de formularios son difíciles de controlar, porque no se conoce cual será el orden de acceso por parte del usuario a cada uno de ellos. Sin embargo, hay ocasiones en la que esto es muy útil. Las ventanas de herramientas son

formularios de este tipo. Por ejemplo, la caja de diálogo del menú de Visual Studio **Edición / Buscar y reemplazar/ Buscar**, es otro ejemplo de formulario no modal.

Para presentar un formulario como una caja de diálogo modal, los pasos que se deben seguir son los siguientes:

- Llamar al método `ShowDialog()`. Por ejemplo,

  ```
  Form cajaAbout = new Form();
  cajaAbout.ShowDialog();
  ```

  El método `ShowDialog()` de la clase `Form` tiene un argumento opcional, `owner`, que puede ser utilizado para especificar la relación padre-hijo de un formulario. Por ejemplo, en el código del formulario principal se puede pasar `this` como el propietario de la caja de diálogo para establecer el formulario principal como propietario o padre:

  ```
  Form f = new Form();
  f.ShowDialog( this );
  ```

  Si un formulario se presenta como modal, el código siguiente a la instrucción

  ```
  f.ShowDialog(this);
  ```

  no se ejecuta hasta que dicha ventana se cierre.

Para presentar un formulario como una caja de dialogo no modal, debe llamarse al método `Show()`:

```
Form f= new Form();
f.Show();
```

Ejemplo: trabajando con distintos eventos en un formulario

Esta aplicación permite comprobar los eventos que se van produciendo e ilustra bien algunos de los eventos de la clase `Control`.

- Cree una nueva aplicación para Windows denominada `EventosVarios`.

VisualStudio genera una aplicación, con un formulario llamado `Form1`, en un fichero llamado `Form1.cs`.

- A continuación, modifique el nombre del formulario: para ello en la ventana **Vista de clases**, pulse sobre `Form1` (Figura 15.5). En la **ventana de propiedades**, cambie el nombre del formulario `Form1` a `Formulario` (Figura 15.6).

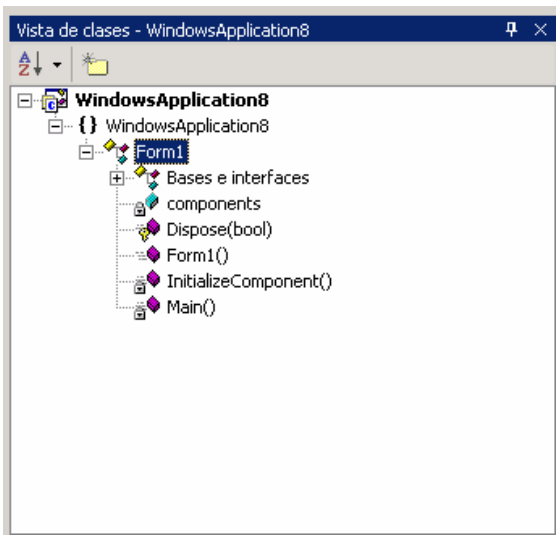

Figura 15.5. Vista de clases

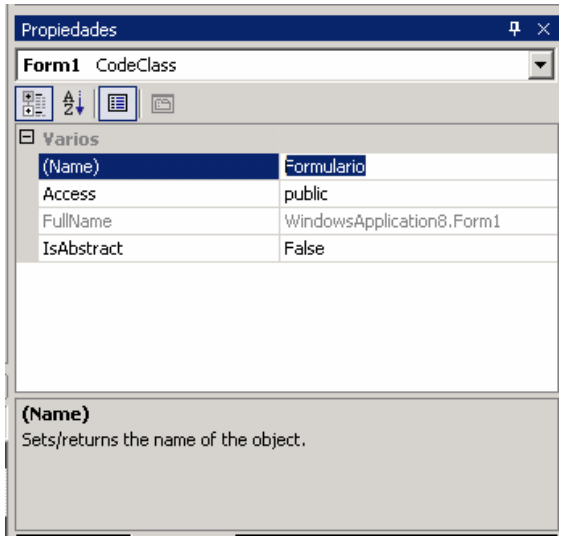

Figura 15.6. Ventana de propiedades

A continuación, cambie el nombre del fichero `Form1.cs` a `FormularioPrincipal.cs` ; pulse en el **explorador de soluciones** sobre el formulario `Formulario` y posteriormente, en la **ventana de propiedades**, cambie la propiedad `nombre de archivo` a `FormularioPrincipal.cs`. También podría haberlo hecho con la opción **Archivo/Guardar** `Form1.cs` **como...** `FormularioPrincipal.cs`

Nota: En realidad en la versión ß de VisualStudio, hay que cambiar manualmente el nombre del formulario del que la aplicación hace una instancia, es decir se cambia la línea

```
Application.Run(new Form1());
```

Por la línea:

```
Application.Run(new Formulario());
```

OJO COMPROBAR ESTO EN LA VERSION FINAL PORQUE ES POSIBLE QUE SE HAYA ARREGLADO

- Guarde la aplicación y compílela para comprobar que no dá ningún error.
- Cambie la propiedad `Text` del formulario a `Formulario Principal`.
- Coloque dos cajas de texto en el formulario, con sus propiedades `Name` a `txEventos1` y `txEventos2` respectivamente y la propiedad `Text` de ambas vacía (figura 15.7).

5. Sitúe un botón en el centro del formulario, con su propiedad `Text` a `Aceptar` y su propiedad `Name` `btnAceptar` (figura 15.7).

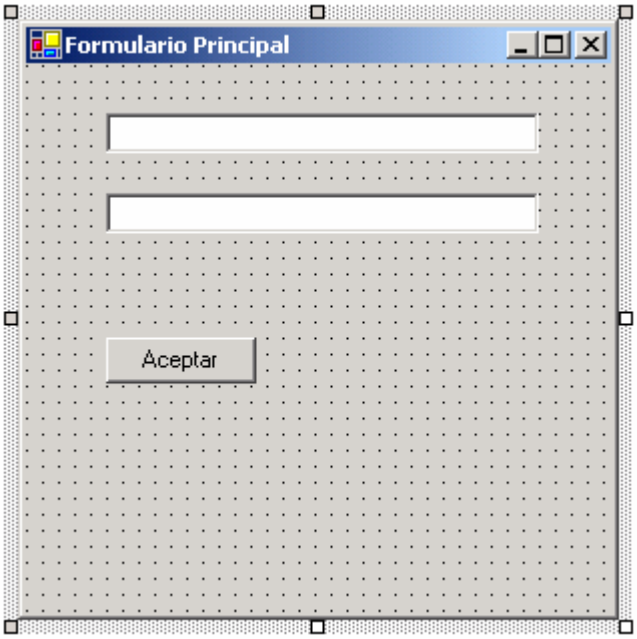

Figura 15.7. Formulario principal del proyecto `EventosVarios`

- A continuación, se van a manejar algunos eventos sobre el botón, sobre la caja de texto y sobre el formulario.

  a) Eventos del botón `btnAceptar`:
  Seleccione el botón `btnAceptar` –pulse una sola vez sobre él- y haga doble Click sobre los eventos `MouseEnter`, `MouseLeave`, `MouseUp` y `MouseDown` en la **Ventana de propiedades.** Añada el código que se indica para cada evento:

```
protected void btnAceptar_MouseUp(object sender,System.WinForms.MouseEventArgs
e)
{
      txEventos1.Text="Evento Mouse Up del boton";
}

protected void btnAceptar_MouseLeave (object sender, System.EventArgs e)
{
      txEventos1.Text="Evento Mouse Leave del boton";
}

protected void btnAceptar_MouseEnter (object sender, System.EventArgs e)
{
      txEventos1.Text="Evento Mouse enter del boton";
}

protected void btnAceptar_MouseDown (object sender, MouseEventArgs e)
{
      txEventos1.Text="Evento Mouse Down del boton";
}
```

b) Eventos sobre el formulario:
Observe cómo se puede obtener la información que llega encapsulada en el objeto `e` de la clase `MouseEventArgs`, en uno de los argumentos de la función miembro `Formulario_MouseMove`. Sitúe una nueva caja de texto en el formulario, con la

propiedad `Name txEventos3`, sin texto en la caja –propiedad `Text` vacía-. A continuación, pulse sobre el formulario, y en la **ventana de propiedades,** en **eventos**, haga doble-click sobre el evento `MouseMove`. Se abre la ventana del código correspondiente al formulario. Escriba el siguiente código:

```
protected void Formulario_MouseMove(object sender,System.WinForms.MouseEventArgs
e)
{
      txEventos3.Text="Coordenadas: ( x = " + e.X + " , y = " + e.Y + ")";
}
```

Para entender el código anterior y aunque se estudiará con más detalle más adelante, es necesario adelantar ahora que un objeto `e` de la clase `MouseEventArgs` encapsula información sobre el origen del evento: el botón presionado, coordenadas del puntero, etc. Tiene las siguientes propiedades:

```
MouseButton Button;     //indica qué botón se ha presionado
int Clicks;             //nº de clicks de ratón
int X;                  //Coordenadas x e y del puntero del ratón
int Y;
```

- Por último, se puede ver cuándo se produce el evento `Resize` del formulario. Para ello, pulse sobre el formulario, y en la **ventana de propiedades** en **eventos**, haga doble click sobre el evento `Resize` y escriba una frase significativa de este evento en el código del método manejador del evento. Por ejemplo:

```
protected void Formulario_Resize (object sender, System.EventArgs
e)
{
   txEventos2.Text="Evento resize del formulario";
}
```

Compile y ejecute la aplicación.

Observe los distintos eventos que se han manejado. Modifique también las dimensiones del formulario.

La clase Button

En C# los controles `RadioButton`, `CheckButton` y `Button` son botones. Esto es así porque cada uno de ellos deriva directamente de la clase `ButtonBase`, como indica la figura 15.8:

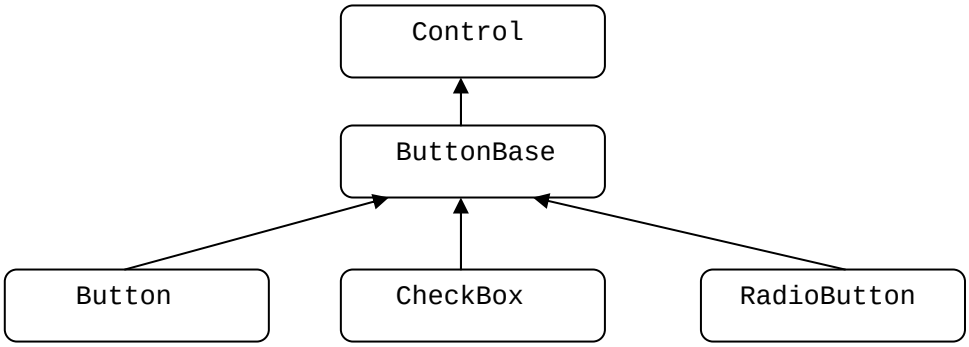

Figura 15.8

En este capítulo se estudian sólo las propiedades y eventos de la clase `Button`, dejando las otras dos para un capítulo posterior.

Propiedades

Como en los anteriores casos, no pretende dar una lista exhaustiva de propiedades de la clase sino de las más importantes y usuales. No se citan aquí las que se heredan de la clase `Control`, que son también propiedades de la clase `Button`.

`FlatSize`	Define la representación o el estilo gráfico del botón.
`Image`	Permite especificar una imagen para el botón.
`ImageAlign`	Especifica la localización relativa de la imagen en la superficie del botón.
`Text`	Texto del botón

Eventos

La clase `Button` hereda los eventos de la clase `Control`. El más utilizado de todos ellos es `Click`. Ocurre al presionar el botón izquierdo del ratón y levantarlo o liberarlo mientras el puntero del ratón está sobre el mismo control donde se inició la pulsación. Esto quiere decir que si se presiona sobre un control con el botón izquierdo del ratón y a continuación –con el botón pulsado, es decir, "arrastrando"- se desplaza fuera del control y posteriormente se libera el ratón, no se producirá el evento click. Por otro lado, si un botón tiene el foco y el usuario presiona la tecla ENTER se produce un evento click sobre ese control. Esto sucede así siempre que ningún control tenga la propiedad `AcceptButton` a `True`.

Ejemplo: trabajando con botones

a) Cree un nuevo proyecto denominado `ProyectoBotones`.
b) Cambie el nombre de la clase del formulario de `Form1` a `FormularioBotones`. pulsando sobre el formulario `Form1` y en la **ventana de propiedades** cambie la prodiedad `Name` a `FormularioBotones`. Desde la ventana **Explorador de soluciones** cambie el nombre del fichero fuente `Form1.cs` a `FormularioBotones.cs`.

   Cambie la línea de código en el fichero `FormularioBotones.cs`

   ```
   Application.Run(new Form1());
   ```

   a

   ```
   Application.Run(new FormularioBotones());  OJO. Solo beta????
   ```

   Cambie la propiedad `Text` del formulario y déjela en blanco.

c) Coloque tres botones en el formulario con las siguientes propiedades:

```
Name: btnAceptar
Text: Aceptar

Name: btnCancelar
Text: Cancelar

Name: btnSalir
Text: Salir
Image:C:\Archivos de programa\
Microsoft          Visual          Studio.NET\Common7\Graphics\icons\
Misc\SECUR08.ICO
ImageAlign: MiddleRight
```

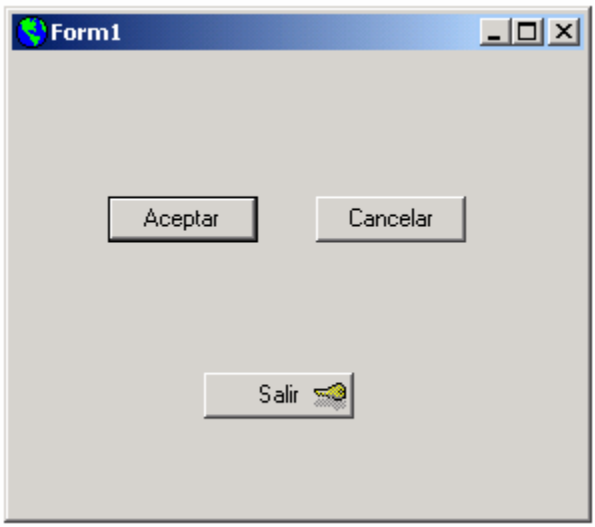

Figura 15.9

En las propiedades del formulario cambie las siguientes propiedades:

```
AcceptButton:  btnAceptar
CancelButton:  btnCancelar
```

```
Icon: C:\Archivos de programa\Microsoft Visual
                Studio.NET\Common7\Graphics\icons\Elements\Earth.ICO
```

Con la asignación de las propiedades `AcceptButton` y `CancelButton` a los botones `btnAceptar` y `btnCancelar`, se logra que cuando se pulsa la tecla `ENTER` ocurra el evento `Click` en el `btnAceptar` y que si se pulsa `ESCAPE`, se produzca el evento `Click` en el botón `btnCancelar`. En resumen, el efecto de pulsar sobre cada uno de los botones es el mismo que el de pulsar las teclas `ENTER` o `ESCAPE`.

La propiedad `Icon`, asigna un icono al formulario.

d) A continuación se escribe el código que manejará el evento del botón `btnCancelar`, pero cambiando el nombre del método que el entorno propone por defecto. Para ello, en la **ventana de propiedades** del `btnCancelar`, pulse sobre el icono correspondiente a eventos. Escriba el nombre que desee para el evento –por ejemplo, `metodoClickCancelar-` en la parte derecha del evento `Click` y realice un doble click sobre el nombre del evento. Se puede observar que se abre la ventana de código, que ha registrado el evento en el auditor y que además dispone las cosas para que se pueda escribir el código directamente. En cualquier momento puede modificar el nombre del método, bien desde la ventana de código o bien desde la **ventana de propiedades**. Escriba el siguiente código en el método.

```
private void metodoClickCancelar(object sender, System.EventArgs e)
{
   this.BackColor=Color.Blue;
   this.Text="Ha pulsado el boton cancelar";
   this.btnAceptar.BackColor=Color.Red;
}
```

Se puede observar que alguna propiedades se heredan de padres a hijos. En este caso, al modificar el color del fondo del formulario se cambian también las de todos los controles que sean “hijos” de ese control. Sólo si se cambia la propiedad en el control –bien en tiempo de diseño o bien, como se ha hecho en este ejemplo, en tiempo de ejecución- deja de ser la misma de la del formulario. Esto pasa con otras propiedades como el tipo de letra, color de la letra, color de fondo...

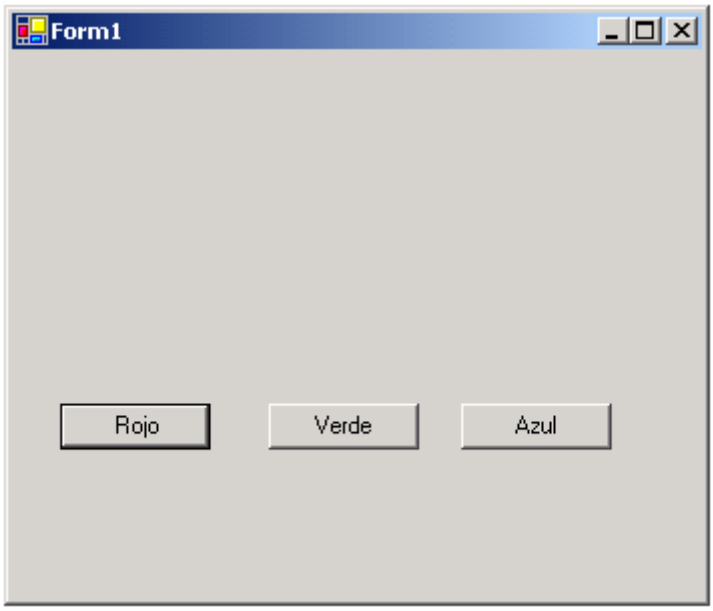

Figura 15.10

Añadiendo manejadores o auditores de eventos:

Para tratar el evento `Click` de `btnAceptar` se pueden hacer las cosas, paso a paso –como en el apartado anterior- o bien se puede hacer doble click en tiempo de diseño sobre el botón. Visual Studio traslada el foco y el cursor a la **ventana de código**. Esto sucede así con todos los controles y se registra el evento por defecto de ese control, que es el evento más común o utilizado de dicho control.

Además, se escriben de manera automática varias líneas de código.

La primera de ellas en el método `InitializeComponent()` que registra el evento `Click`. Por ejemplo, si se hace esta operación en el botón `btnAceptar`, la línea que se añade es:

```
this.btnAceptar.Click += new
System.EventHandler(this.btnAceptar_Click);
```

Si no la encuentra esta línea, pulse sobre el signo + situado a la izquierda de la línea:

```
private void InitializeComponent()…
```

y se descomprimirá el código correspondiente a ese método. En él se encontrará la línea anterior.

`btnAceptar.Click` hace referencia al evento `Click` del botón `btnAceptar`. Por otro lado, se crea un objeto de la clase `System.EventHandler` para que manipule este evento a través del método que se le pasa como parámetro a su constructor, que es `btnAceptar_Click` . `this` es la referencia al formulario actual. El nombre que propone Visual Studio para el método manejador está compuesto por el nombre del control que

es fuente del evento, seguido del nombre el evento. Como anteriormente se ha hecho, se puede cambiar el nombre en la **ventana de propiedades**.

`+=` indica que se está registrando el evento situado en el lado izquierdo en el auditor que manipulará dicho evento y que se crea en la parte derecha de la expresión.

Además de esta línea se añade en la parte final del código las siguientes líneas:

```
private void btnAceptar_Click(object sender, System.EventArgs e)
{

}
```

El código que el programador escriba en este método es el que se ejecutará cuando se produzca el evento `Click` del botón. El programador sólo tiene que escribir dicho código en el cuerpo del método.

Observe que se pasan dos parámetros al método manejador del evento, que son un objeto que encapsula el objeto fuente del evento y otro que encapsula el propio evento:

a) `sender`: es un objeto de la clase `object`. Es un referencia al control donde se ha producido el evento. En este caso, `sender` es el botón `btnAceptar`, pero en otros casos este método puede ser compartido por varios controles y se puede utilizare este parámetro para chequear, en tiempo de ejecución, el control que ha producido el evento. Más adelante se utiliza esta técnica en varios ejemplos.
b) `e`: este objeto encapsula la información sobre el evento que se ha producido. En este caso, no se utiliza.

Escriba el siguiente código para tratar el evento `Click`, en el interior del cuerpo del método anterior.

```
this.Text="Se ha pulsado el botón Aceptar";
```

Con esa línea se asigna a la propiedad `Text` del formulario `-this-` la cadena de la parte derecha de la expresión.

Siga los mismos pasos para añadir un método que manipule el evento `Click` del botón `btnSalir` y escriba la siguiente línea en el cuerpo del método para salir de la aplicación:

```
Application.Exit();
```

`Exit()` es un método estático de la clase `Application` que hace que la aplicación termine o se cierre.

Las clases Label y LinkLabel

El propósito de estos dos controles es presentar una etiqueta con objeto de exponer una información en el formulario. Visual Studio .NET presenta dos tipos de etiquetas:

a) `Label`: es la etiqueta estándar.

b) `LinkLabel`: deriva de la clase `Label` y tiene algunas características añadidas. Tiene aspecto de hiperlink.

En general, una etiqueta no necesita añadir código. Sin embargo, si se desea que una etiqueta de tipo `LinkLabel` abra una página web de una determinada dirección es necesario añadir código extra. Muchas de las propiedades de `Label` derivan de la clase `Control`. Aquí se señalan las más importantes. Algunas de ellas corresponden sólo a la clase `LinkLabel`.

`Border Style`	Permite especificar el estilo de los bordes de la etiqueta.
`DisableLinkColor`	(Sólo `LinkLabel`) Define el color de la etiqueta después de haber sido pulsada.
`LinkArea`	(Sólo `LinkLabel`) Es el rango de texto que será presentado como un Link.
`LinkVisited`	(Sólo `LinkLabel`) Propiedad booleana. Especifica si una dirección ha sido visitada o no.
`LinkColor`	(Sólo `LinkLabel`) Color del Link

La clase TextBox

Este control se utiliza para que el usuario introduzca texto en él . Se puede “filtrar” el tipo de caracteres que se introducen en este control.

Esta clase -al igual que la clase `RichTextBox`- deriva de la clase `TextBoxBase`, que a su vez deriva de la clase `Control`. La clase `RichTextBox` no se estudiará detenidamente en este libro pero se realiza un extenso ejemplo que la utiliza más adelante.

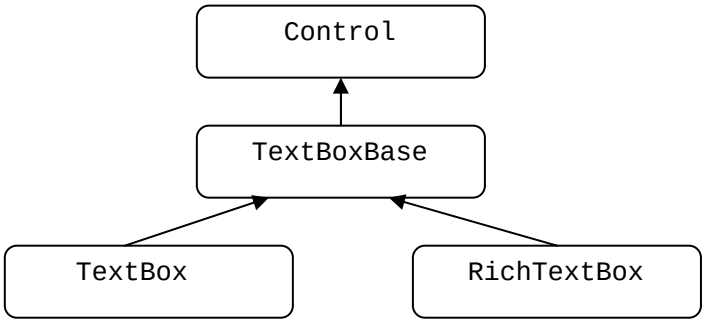

Figura 15.11

La clase `TextBoxBase` proporciona la funcionalidad básica para manipular cajas de texto, como por ejemplo, cortar, copiar o pegar texto.

Propiedades

Las propiedades más importantes son:

`CausesValidation`	Indica si el control genera eventos de validación.
`CharacterCasing`	Indica si el texto que se ha introducido se convierte a mayúsculas –`Upper`-, minúsculas –`Lower`- o se escribe como se introduce –`Normal`-
`MaxLength`	Especifica el número máximo de caracteres que se pueden escribir en una caja de texto. `0` indica que se pueden introducir tantos caracteres como se desee –siempre que la memoria lo permita-.
`MultiLine`	Indica si el control puede tener varias líneas o sólo una.
`PasswordChar`	Especifica el carácter que se escribirá para sustituir a los caracteres que se van introduciendo. Esta propiedad sólo tiene efecto si `MultiLine` es `false` –una sóla línea-
`ReadOnly`	Especifica si el contenido de la caja de texto es de sólo lectura.
`ScrollBars`	Especifica el tipo de barras de desplazamiento.
`SelectedText`	Especifica el texto seleccionado en la caja de texto.
`SelectionLength`	Define el nº de caracteres seleccionados en la caja de texto.
`SelectionStart`	Especifica el lugar donde comenzará la selección.
`WordWrap`	Especifica si en una caja de texto multilínea se realiza o no el salto de línea de manera automática al extenderse la longitud del texto.

Los eventos más importantes de una caja de texto son los mismos que los que se han estudiado para la clase `Control`. Es importante señalar que, aunque los eventos de teclado y de ratón pertenecen a la clase Control sin embargo, se explican a continuación porque son eventos muy utilizados en estos controles. Además, este control tiene un evento específico que se denomina `Change`, que ocurre siempre que cambia el texto de la caja.

Antes de realizar un ejemplo que ayude a comprender cómo puede controlarse los caracteres que se introducen por el teclado, se estudian con cierto detenimiento, los eventos de ratón y teclado.

Eventos del ratón

Clase MouseEventArgs

Cuando ocurre uno de los eventos `MouseUp`, `MouseDown` o `MouseMove`, se pasa al método manejador un objeto de la clase `MouseEventArgs`, que encapsula y proporciona información sobre los eventos `MouseUp`, `MouseDown` o `MouseMove` que se han producido.

Esta clase deriva de la clase `EventArgs` que a su vez deriva directamente de `Object`:

El evento `MouseDown` ocurre cuando el usuario presiona el botón mientras el puntero está sobre el control. `MouseUp` se produce cuando el usuario libera el botón del ratón mientras el puntero permanece sobre el control. `MouseMove` se dá cuando el usuario mueve el puntero del ratón sobre el control. Estos eventos pasan siempre a los manejadores de estos eventos un objeto de la clase `MouseEventArgs`, que especifica y encapsula información sobre qué botón se ha presionado, cuántas veces, las coordenadas del puntero del ratón, etc.

Las propiedades de `MouseEventArgs` son:

`Button`	Indica qué botón fue presionado. La propiedad puede valer uno de los valores de la enumeración `MouseButtons`.
`Clicks`	Indica el número de veces que se presionó el botón y fue liberado. Es un entero.
`X`	Coordenada `X` del ratón, en pixeles, relativa al area cliente. Es un entero.
`Y`	Coordenada `Y` del ratón, en pixeles, relativa al area cliente. Es un entero.

Enumeración MouseButtons

Es una constante que especifica qué boton del ratón se ha pulsado en el evento.

Esta enumeración se usa en muchas clases como `AxHost`, `Control`, `DataGrid`, `Form`, `RadioButton`, `Splitter`, `StatusBar`, y `UpDownBase`.

`MouseButtons` puede tomar los siguientes valores:

`Left`	Se presionó el botón izquierdo.
`Middle`	Se presionó el botón del medio.
`None`	No se pulsó ningún botón.
`Right`	Se presionó el botón derecho

Ejemplo: trabajando con los eventos de ratón

Cree una nueva aplicación. Llámela `EventosDeRaton`. Borre la propiedad `Text` del formulario. Añada un `Label` en el centro del formulario y ponga su propiedad `Name` a `texto` y borre su propiedad `Text`.

Esta sencilla aplicación permite controlar las coordenadas del lugar donde se pulsa el ratón que irán escritas en la cabecera del formulario y la posición del ratón mientras se mueva –se presentará en la etiqueta-.

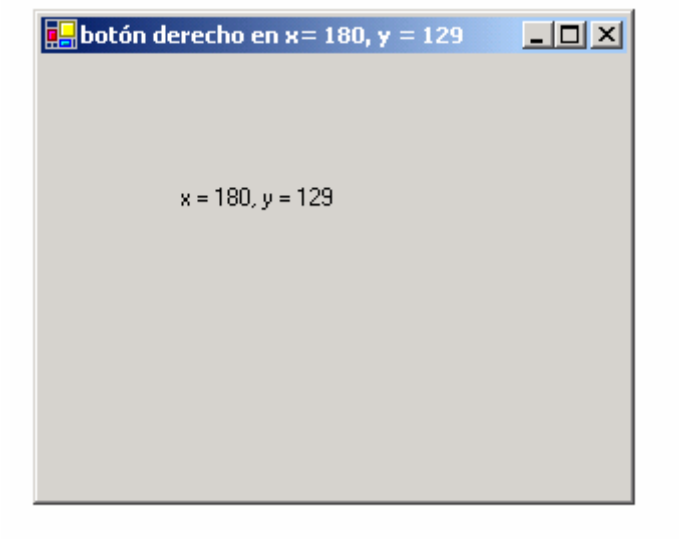

Figura 15.12

Para ello, se va a trabajar con los eventos del formulario, `MouseDown` y `MouseMove`.

En **la ventana de propiedades** del formulario, pulse el icono correspondiente a eventos y haga un doble click sobre los eventos `MouseDown` y `MouseMove`.

Escriba el siguiente código en cada uno de ellos:

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
      texto.Text = " x = " + e.X;
      texto.Text+= ", y = " + e.Y;
}

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
      if(e.Button == MouseButtons.Left)
            this.Text = "botón izquierdo";
      else if(e.Button == MouseButtons.Right)
            this.Text = "botón derecho";
      else if(e.Button == MouseButtons.Middle)
            this.Text = "botón central";
      this.Text += "en x= " + e.X + ", y = " + e.Y;
}
```

Compile y ejecute la aplicación.

Eventos de teclado

El evento `KeyPress` permite obtener información sobre el carácter correspondiente a la tecla pulsada y los eventos `KeyDown` y `KeyUp` sobre las teclas modificadoras que se han pulsado de manera simultánea a otra tecla.

KeyPress

El evento `KeyPress` ocurre cuando el usuario presiona una tecla. En realidad, se producen tres eventos. `KeyDown`, `KeyPress` y `KeyUp` –por este orden- . El evento `KeyDown` precede al evento `KeyPress` y al evento `KeyUp` que ocurre cuando el usuario *libera* la tecla. Cuando el usuario *permanece con la tecla pulsada*, se repiten cada cierto tiempo los eventos `KeyDown` y `KeyPress` pero sólo se produce una vez el evento `KeyUp`.

Cuando se produce un evento `KeyPress` se envían al método manejador del evento dos parámetros: un objeto de la clase `object` llamado `sender` que contiene información correspondiente al control donde se ha originado el evento y un objeto de la clase `KeyPressEventArgs` que encapsula información sobre el carácter de la tecla que se ha pulsado.

Con cada evento `KeyDown` o `KeyUp`, además de `sender` se pasa otro parámetro: un objeto de la clase `KeyEventArgs` que encapsula información sobre las teclas modificadoras (`CTRL`, `SHIFT`, o `ALT`) que han sido presionadas de manera simultánea a otra tecla.

La clase KeyPressEventArgs

La jerarquía de esta clase es la siguiente:

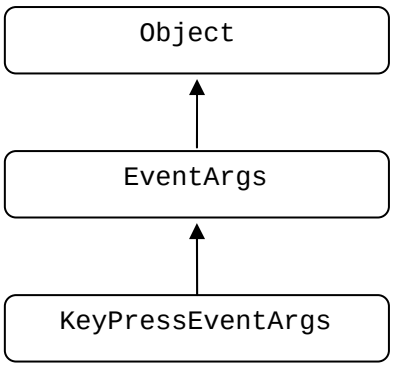

Figura 15.13

Un objeto de la clase `KeyPressEventArgs` especifica el carácter que se produce al pulsar una tecla. Por ejemplo, cuando el usuario presiona `SHIFT + K`, la propiedad `KeyChar` devuelve una `K` (mayúscula).

Las propiedades más importantes de la clase `KeyPressEventArgs` son:

`Handled`	Valor booleano que indica si el evento `KeyPress` ha sido manipulado. Si es `true`, no se hace nada con el carácter pulsado. Si `false`, se sigue el curso normal. Un buen ejemplo lo constituye la aplicación que se hace a continuación.
`KeyChar`	Obtiene el carácter –como `char`- correspondiente a la tecla pulsada.

Ejemplo: Trabajando con los eventos del teclado.

Esta aplicación, obtiene el carácter de la tecla pulsada y su código ASCII. Se estudia el evento `KeyPress`, y, en concreto, la propiedad `KeyChar` de la clase `KeyPressEventArgs` que es un `char`. Para ello:

- Cree un nuevo proyecto denominado `ProyectoEventosKeyPress`.
- Sitúe cuatro etiquetas en el formulario con los textos que se indican en la figura 15.14, y coloque tres cajas de texto, con la propiedad `Text` vacía y las siguientes propiedades `Name`: `texto`, `codigoKeyChar` y `codigoNumericoKeyChar`, respectivamente.

Figura 15.14

- En el evento `KeyPress` de la primera caja de texto, `texto`, escriba el siguiente código:

```
private void texto_KeyPress(object sender, KeyPressEventArgs e)
{
   char caracterIntroducido=e.KeyChar;
   codigoKeyChar .Text=caracterIntroducido.ToString();
   codigoNumericoKeyChar.Text=((int)caracterIntroducido).ToString();
}
```

Se puede comprobar que cuando en la caja texto `texto` se pulsa una tecla, el evento `KeyPress` puede capturar el código como un `char` a través de la propiedad `KeyChar` de e, que es el parámetro que recibe este método. Por esta razón, se puede distinguir entre

las letras a y A. Observe que algunas teclas especiales también tienen un código ASCII aunque no devuelven un carácter "legible". A continuación, se adjunta una tabla con los códigos ASCII correspondientes a los valores más usuales:

Tecla pulsada	**Valor ASCII**
0, 1, 2, ... , 9	48, 49, ..., 57
a, b, c, ... , z	97, 98, 99, ... , 122
A, B, ... , Z	65, 66, ..., 90
Esc	27
Retroceso	8
ENTER	13
Control+ENTER	10

Esto es muy útil para filtrar las entradas desde el teclado y será el objeto de este segundo ejemplo:

Ejemplo: filtrando las entradas introducidas desde el teclado.

Se trata de implementar un proyecto con una caja de texto que únicamente permita entradas numéricas. Para ello, se debe controlar que las entradas estén comprendidas entre 0 y 9. Si la entrada no es un número, no se hace nada con la entrada. Sólo se tiene en cuenta una excepción: que la entrada sea la tecla RETROCESO. De esa manera será posible borrar una entrada. Para implementar esto, cambie en el anterior ejemplo el código del evento KeyPress de la caja de texto texto, por el siguiente:

```
private void texto_KeyPress(object sender, KeyPressEventArgs e)
{
      char caracterIntroducido=e.KeyChar;
      //Si no es numerico y distinto de RETROCESO
      if( (caracterIntroducido<48 || caracterIntroducido>57)
                        && caracterIntroducido != 8)
      {
            //No se escribe nada en la caja de texto
            e.Handled=true;
      }

      codigo.Text=caracterIntroducido.ToString();
      codigoNumerico.Text=((int)caracterIntroducido).ToString();
}
```

En la línea correspondiente a la sentencia if los valores ASCII correpondientes a números comprendidos entre 0 y 9, están entre 48 y 57. En esta línea se asegura que el carácter introducido no está en ese rango.

El carácter ASCII correspondiente a la tecla RETROCESO es el 8. Se permite esta entrada para poder modificar la caja de texto. En un ejemplo posterior se verá otra manera de conocer si ésta es la tecla pulsada.

La propiedad Handled de KeyPressEventArgs está a true indicando que no se hace nada con ese carácter, y por lo tanto, no se escribe nada.

Por supuesto, se podría sustituir la línea

```
if( (caracterIntroducido<48 || caracterIntroducido>57)
                        && caracterIntroducido != 8)
```

por la línea

```
if( (caracterIntroducido<'0' || caracterIntroducido>'9')
                        && caracterIntroducido != '\b')
```

que es quizá más comprensible.

Aunque no se describirá la aplicación completa, es interesante considerar la posibilidad de que un programa cuente el número de veces que se han pulsado una serie de teclas, por ejemplo las teclas: `ESCAPE`, `REROCESO`, `RETURN` y número de pulsaciones totales.

Para ello, se podrían definir cuatro variables estáticas del formulario, por ejemplo:

```
static long contadorPulsacionesTeclas = 0 ;
static long contadorRetrocesos =  0;
static long contadorEnter = 0 ;
static long contadorEsc = 0 ;
```

El código del método manipulador del evento `KeyPress` podría ser algo parecido a:

```
private void texto_KeyPress(object sender, KeyPressEventArgs e)
{
        switch(e.KeyChar)
        {
                // Contar  Retrocesos.
                case '\b':
                        contadorRetrocesos++;
                        break ;
                // Contar ENTER.
                case '\r':
                        contadorEnter++;
                        break ;
                // Contar ESCAPE.
                case (char)27:
                        contadorEnter ++ ;
                        break ;
                // Contar otras pulsaciones.
                default:
                        contadorPulsacionesTeclas++ ;
                        break;
        }
}
```

Eventos KeyDown y KeyUp.

Como anteriormente se ha explicado, un evento `KeyDown` ocurre siempre que el usuario *presiona* una tecla. El evento `KeyUp` ocurre cuando se *libera* esa tecla. Siempre que se mantiene pulsada una tecla, ocurre cada determinado tiempo un evento `KeyDown`**,** pero sólo ocurre una vez el evento `KeyUp` –puesto que sólo una vez se ha liberado esa tecla-.

Cuando se produce un evento `KeyDown` o `KeyUp` se pasa un objeto de la clase `KeyEventArgs`, al método que manipula o maneja ese evento. Ese objeto encapsula información sobre qué tecla específica ha pulsado el usuario y si de manera simultánea ha pulsado una tecla modificadora (`CTRL`, `ALT`, y `SHIFT`). La jerarquía de la clase `KeyEventArgs` es la siguiente:

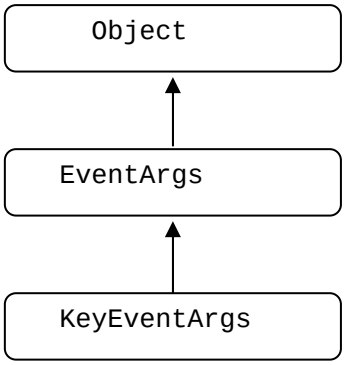

Figura 15.15

Algunas de las propiedades de la clase `KeyEventArgs` son las siguientes:

`Alt`	Obtiene un valor que indica si se ha presionado la tecla `ALT` .
`Control`	Obtiene un valor que indica si se ha presionado la tecla `CTRL`.
`Handled`	Obtiene un valor booleano para indicar si se ha tratado el evento.
`KeyCode`	Devuelve un objeto `Keys` correspondiente al código de la tecla que se ha pulsado. Es de sólo lectura.
`KeyData`	Obtiene un objeto `Keys` con la combinación de las teclas que se presionaron.
`KeyValue`	Devuelve un valor entero correspondiente al código de la tecla que se ha pulsado.
`Modifiers`	Indica qué tecla modificadora se ha pulsado (`CTRL`, `SHIFT` y/o `ALT`).
`Shift`	Obtiene un valor que indica si se ha presionado la tecla `SHIFT`.

Observaciones:

Las propiedades `KeyCode`, `KeyData` y `KeyValue` son de sólo lectura y tienen el siguiente formato:

```
public Keys KeyCode
public Keys KeyData
public int KeyValue
```

Observe que tanto `KeyCode` como `KeyData` son del tipo `Keys`.

Enumeración Keys

Especifica los códigos de las teclas y de los modificadores (`modifiers`).

Esta enumeración contiene constantes para procesar las entradas por teclado. Las teclas se identifican con valores de tecla que consisten en un código y en un juego de modificadores combinados en un simple valor entero. Los cuatro dígitos de la izquierda del valor de la tecla contienen el código de la tecla –que es el mismo que los tradicionales de Windows- y los cuatro de la derecha, contienen los bits modificadores para las teclas `SHIFT`, `CONTROL` y `ALT`.

A continuación se adjunta una tabla con los valores más usuales.

Nombre	**Descripcion**
A, B, … Z	A, B, ..., Z.
Add	+
Alt	ALT.
Back	BACKSPACE.
Cancel	CANCEL.
Capital	CAPS LOCK.
CapsLock	CAPS LOCK .
Clear	CLEAR.
Control	CTRL .
ControlKey	CTRL .
D0, D1, …, D9	0, 1, ...,9.
Decimal	decimal.
Delete	DEL o SUP.
Divide	/
Down	Flecha abajo.
End	END.
Enter	ENTER .
Escape	ESC .
F1, F2, … ,F24	F1, F2, … , F24.
Help	HELP.
Home	HOME.
KeyCode	El bit de mascara de bits para determinar el código de la tecla a partir del valor.
Left	Flecha izquierda.
Modifiers	El bit de máscara de bits para determinar los modificadores.
Next	PAGE DOWN.
None	Ninguna tecla presionada
NumLock	NUM LOCK .
NumPad0, NumPad1, NumPad9	…,Teclas teclado numerico 0, 1, 2,…,9.
PageDown	PAGE DOWN
PageUp	PAGE UP .
Pause	PAUSE .
Print	PRINT .
PrintScreen	PRINT SCREEN .
Prior	PAGE UP .
Return	The RETURN .
Right	Flecha derecha.
Shift	SHIFT.
Snapshot	PRINT SCREEN .
Space	SPACEBAR .

```
Subtract                -
Tab                     TAB.
Up                      Flecha arriba.
```

Ejemplo: trabajando con eventos de teclado II

En este ejemplo, se puede observar bien las diferencias entre `KeyCode`, `KeyData`, `KeyValue` y `Modifiers`.

Cree un nuevo proyecto, y llámele `ProyectoEventosKey`. Sitúe cuatro etiquetas en el formulario principal, en una columna, déje sus propiedades `Name` por defecto y ponga sus propiedades `Text` a `KeyCode`, `KeyData`, `KeyValue` y `Modifiers`. Sitúe en la parte de la derecha otra columna de cuatro etiquetas, con sus propiedades `Name` a `codigo`, `data`, `valor` y `modificador` y su propiedad `Text` vacía. En la **ventana de propiedades** del formulario `Form1`, pulse sobre la **ventana de eventos** y haga doble-click sobre el evento `KeyUp` y escriba el código siguiente:

```
private void Form1_KeyUp(object sender, KeyEventArgs e)
{
      codigo.Text=e.KeyCode.ToString();
      data.Text=e.KeyData.ToString();
      valor.Text=e.KeyValue.ToString();
      modificador.Text=e.Modifiers.ToString();
}
```

Guarde el programa, compílelo y ejecútelo a continuación.

Compruebe que si, por ejemplo, pulsa la letra `a` y de manera simultánea la tecla `Shift` el formulario se parecerá al de la figura 15.16:

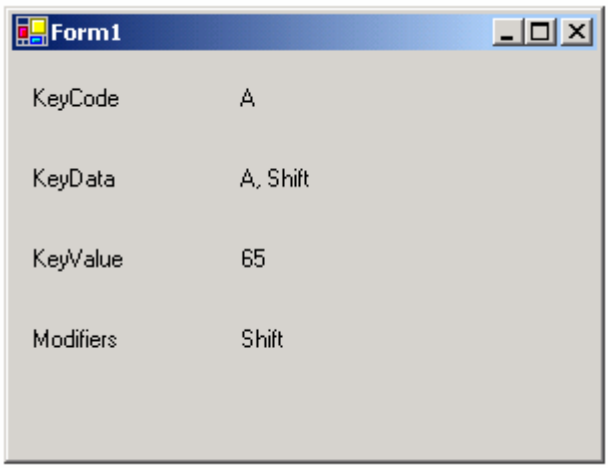

Figura 15.16

Es importante observar que con este evento no se puede distinguir entre la pulsación de la letra `a` y la `A`. Para ello, hay que trabajar con el evento `KeyPress`.