

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Tema 4 – Componentes WindowsForms

Como ya sabemos un componente es una clase ya hecha que podemos reutilizar mediante agregación o herencia. Es la base de la creación de aplicaciones gráficas en las que cogeremos la clase Form de .Net, la heredaremos y mediante agregación de componentes y gestión de eventos crearemos nuestros formularios que serán las ventanas de nuestras aplicaciones.

El formulario

Ya hemos visto brevemente al inicio del curso cómo el diseñador de Visual Studio crea y maneja los gráficos. Cada cambio que hacemos en el diseñador se refleja en cambios en el código del archivo *nombreFormulario.Designer.cs* en la región oculta.

De hecho no se deben hacer cambios en esa región ya que el Visual puede deshacerlos. Cualquier inicialización a mayores se recomienda hacerla en el evento *Load* del formulario o directamente en el constructor pero fuera de la región reservada en *InitializeComponents* (y mejor después de llamar a esta función).

Demos un repaso al código creado por el diseñador:

Si creamos un proyecto nuevo con un formulario y un par de componentes aparecen al menos tres archivos que si no hemos cambiado de nombre serán *Form1.cs*, *Form1.Designer.cs* y *Program.cs*.

La clase nueva *Form1* que hereda de *Form* está dividida entre los dos primeros archivos mediante el uso del modificador parcial:

```
public partial class Form1 : Form
```


En el constructor *Form1()* se llama a *InitializeComponents()* del segundo archivo. En este archivo, *Form1.Designer.cs*, se encuentra el resto de la definición de la clase.

La siguiente función:

```
protected override void Dispose( bool disposing )
```

sirve para limpiar la lista de componentes no visuales (por ejemplo Timers) y otros recursos. La veremos cuando haga falta.

En la zona cerrada por las directivas *#region* y *#endregion* tenemos la función *InitializeComponents* donde se inicializan los componentes visuales y no visuales que van a ser utilizados.

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Vemos un extracto:

```

this.button1 = new System.Windows.Forms.Button();
this.label1 = new System.Windows.Forms.Label();
this.textBox1 = new System.Windows.Forms.TextBox();
this.SuspendLayout(); // Oculta la presentación
//
// button1
//
this.button1.Location = new System.Drawing.Point(82, 60);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(92, 26);
this.button1.TabIndex = 0;

(....)

// Form1
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(292, 266);

//Aquí se añaden los controles a la colección del formulario.

this.Controls.Add(this.textBox1);
this.Controls.Add(this.label1);
this.Controls.Add(this.button1);
this.Name = "Form1";
this.Text = "Form1";
this.ResumeLayout(false); // activa la presentación
this.PerformLayout();


```

Es en la colección *Controls* del formulario (*this*) donde se almacenan los componentes.

Al final tenemos las declaraciones de los componentes. Estas por defecto aparecen privadas. Se pueden poner públicas cambiándolas directamente aquí o en la ventana de propiedades del componente.

Esto último será necesario cuando tengamos varias clases o formularios y queramos acceder de unos a otros.

Por supuesto esto que hacemos de forma gráfica se puede realizar por código tal cual se puede ver en el [Apéndice II](#).

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Propiedades

Muchas de las propiedades son válidas también para otros componentes. Este es un hecho que se va a repetir a lo largo de este tema, ya que todos los componentes heredan de clases comunes.

Name: Evidentemente es para cambiar el nombre del formulario. Es típico en .NET nombrar a los formularios de la forma: *frmXXXX*. También puede usarse acceder mediante hashing al componente en cuestión

Text: Título del formulario. En otros componente será también el texto visual (por ejemplo en un botón o en un TextBox).

AcceptButton y CancelButton: Se puede establecer que de los diversos botones que se coloquen en un formulario uno responda a la tecla ENTER y otro responda a la tecla ESC. Por supuesto estas respuestas serán dadas cuando el foco esté sobre un componente que no recoja la pulsación de una de estas teclas para un funcionamiento propio.

BackColor y ForeColor: Color de fondo y principal. En el formulario importa sobre todo el de fondo, pero el ForeColor será muy útil en otros componentes.

Para cambiar un color en tiempo de ejecución vamos a disponer de la clase Color que usaremos de dos formas principalmente:

1. Asignación directa de un color según el nombre: Se usa el enumerado Color

```
this.BackColor=Color.AliceBlue;
```

2. Mediante las componentes RGB

```
this.BackColor=Color.FromArgb (224,240,32);
this.BackColor=Color.FromArgb (0xE0,0xF0,0x20); // En hexadecimal
```

BackgroundImage: Coloca una imagen de fondo. Si queréis cargarla en tiempo de ejecución hay que crear un nuevo objeto de la clase *Bitmap*:

```
this.BackgroundImage = new Bitmap(@"C:\Windows\Web\Wallpaper\Theme1\img2.jpg");
```

O usando caracteres de escape \\. Es necesario tener el espacio de nombres **System.Drawing**.

Icon: Cambia el icono que se desea tener en la esquina del formulario. Se puede crear en tiempo de ejecución de forma similar a las imágenes pero con archivos tipo .ICO.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

FormBorderStyle: Indica el estilo del formulario:

Nombre de miembro	Descripción
Fixed3D	Borde tridimensional fijo.
FixedDialog	Borde de estilo de cuadro de diálogo fijo y de ancho grueso.
FixedSingle	Borde fijo de una sola línea.
FixedToolWindow	Borde de ventana de herramienta de tamaño fijo.
None	Sin borde.
Sizable	Borde de tamaño variable.
SizableToolWindow	Borde de ventana de herramienta de tamaño variable.

ControlBox, MaximizeBox, MinimizeBox: Establece si se desea o no que aparezcan los distintos iconos de minimizar, maximizar, cerrar y control en la barra de título del formulario.

Font: Establece u obtiene los distintos fonts, tamaños, atributos, etc. deseados para ese formulario o componente.

Cursor: Forma del cursor al pasar por encima de ese componente. Admite nuevos cursores a partir de archivos .CUR o a partir del enumerado *Cursors*.

```
this.Cursor = new Cursor("C:\\Windows\\Cursors\\aero_link.cur");
this.Cursor = Cursors.IBeam;
```

No admite usar cursores animados directamente (.ani), si se desea hacer esto hay que realizar algún truco más complejo como el de este ejemplo:

www.navioo.com/csharp/source_code/Load_animated_cursor_74.html

Location: Posición en pixels del componente. En el caso del formulario va a depender de *StartPosition*. Se puede acceder a través de *Top* y *Left*, o *Location.X* y *Location.Y*. Tipo *Point*. Está superitada a la propiedad *StartPosition*.

StartPosition: Indica dónde aparece el formulario cuando se le llama. Si se pone manual, la posición será la indicada por *Location*.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Size, MaximumSize y MinimumSize: Tamaño en un momento dado y el mínimo y máximo que puede tener. Las tres propiedades son estructuras con los campos *Width* y *Height*.

También se puede acceder a través de *this* a *width* y *height* directamente.

También se puede usar como una clase (clase *Size*) y tiene diversos métodos para su manejo, ver MSDN)

ClientSize: Área cliente del formulario. Es la zona del formulario donde se puede “trabajar”. La parte interna al marco.

WindowState: Indica o establece el estado de la ventana en un momento dado: Maximizada, minimizada o normal.

ShowInTaskBar: Indica si se desea que aparezca o no en la barra de tareas.

Métodos

Existen muchos más de los que aquí veremos.

Close: Cierra el formulario (para volver a abrirlo hay que hacer otro New)

Hide: Oculta el formulario

Show: Muestra el formulario. Más adelante veremos *ShowDialog* que es una variante de *Show*.

Eventos

Los eventos son comunes a muchos controles, y para no hacerlo tedioso iremos repartiendo los distintos eventos en distintos controles.

Como ya se comentó, cualquier procedimiento puede ser ejecutado al ocurrir un evento, simplemente tiene que cumplir dos cosas:

- Enlazarlo con el evento correspondiente mediante

```
Control.Evento += new System.EventHandler(funcion);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Se usa el operador += porque a un evento se le pueden asociar varios métodos a ejecutar. Además el evento no admite tener sólo el operador =.

Un evento es un concepto similar al de delegado que se ve en el tema de multihilo. Sirve para poder disponer desde una variable una referencia (puntero) a una función.

Por ejemplo:

```
this.button1.Click += new System.EventHandler(this.superclick);
```

Esto aparece en *InitializeComponents* y veremos más adelante cómo manejarlo más a fondo. Por ahora enlazaremos eventos en modo Visual seleccionando en la ventana de propiedades el evento y el método asociado.

- Que exista correspondencia con los parámetros que hay que pasar a dicho evento

Normalmente todos los eventos disponen de dos parámetros:

- *Sender*: indica quién ha generado el evento.
- *Argumentos (e)*: estructura con información del evento. Por ejemplo en un *KeyPressed* informa sobre la tecla pulsada, o en un *MouseMove* hay indicaciones de las coordenadas donde está el ratón o si se pulsa algún botón. Será de la clase *EventArgs* o una derivada (como *MouseEventArgs*)


Para buscar un evento y codificarlo, en la ventana de propiedades se escoge el botón de eventos y se hace doble click sobre el evento deseado.

Veamos algunos eventos:

Load: Al cargar el formulario. Se ejecuta después del constructor. Es un evento equivalente al constructor que se mantiene por **razones históricas**. Se recomienda en .Net inicializar en el constructor salvo cosas dependientes del aspecto visual del formulario (tamaño, posición, etc.). Inicialmente usaremos indistintamente el Load o el constructor (El constructor siempre después de *InitializeComponents*).

FormClosing: Evento que se genera en cuanto el usuario quiere cerrarlo. Este evento es útil porque permite la cancelación del cierre del formulario y se va a usar para confirmaciones.

En los argumentos del parámetro *e*, existe el campo *Cancel*. Si dicho campo se pone a *TRUE*, se cancela el cierre del Formulario. Útil para preguntar si se desea salir o no de una aplicación sin guardar cambios o similar aprovechando la respuesta de una *MessageBox*.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Veamos un ejemplo:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (MessageBox.Show("¿Seguro que desea salir?", "Mi Aplicación",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Question)
        == DialogResult.Cancel)
    {
        e.Cancel = true;
    }
}
```

FormClosed: Ocurre cuando ya se ha cerrado el formulario. Útil para liberar recursos o actualizar algún dato al cerrar un formulario.

Para cerrar formularios se recomienda usar el método *Close*, no *Environment.Exit(0)*, ya que esta última no genera eventos *Closed* ni *Closing*.

Eventos del ratón: En los argumentos suelen tener información sobre coordenadas dónde está situado el puntero y sobre si hay o no algún botón pulsado.

Los más comunes son los siguientes:

MouseDown: Se produce cuando el puntero del mouse está sobre el control y se presiona un botón del mouse.

MouseEnter: Se produce cuando el puntero del mouse entra en el control.

MouseLeave: Se produce cuando el puntero del mouse deja el control.

MouseMove: Se produce cuando el puntero del mouse se mueve sobre el control.

MouseUp: Se produce cuando el puntero del mouse está encima del control y se suelta un botón del mouse.

Eventos de teclado: Para el control de teclas:

KeyDown: Se produce únicamente cuando se presiona una tecla por primera vez.


KeyUp: Se produce cuando se suelta una tecla.

Los dos eventos anteriores se generan para todas las teclas pulsadas.

Los argumentos más importantes de los dos eventos son:

KeyCode: Código de la tecla pulsada. Se ve como enumerado (p.ej. *Keys.A*)

Alt, Control, Shift: booleanos que indican si una o varias de esas teclas están pulsadas.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Para el control de caracteres:

KeyPress: Se produce cada vez que se presiona una tecla. Si la tecla se mantiene presionada, se produce un evento *KeyPress* a la velocidad de repetición definida por el sistema operativo. Se genera sólo para teclas que contienen un carácter.


Argumento principal: *KeyChar*: Contiene el valor Unicode del carácter pulsado.

Propiedad KeyPreview del formulario

Propiedad que si se pone a TRUE provoca que la gestión del teclado sea realizada mediante los eventos del formulario y no sobre los eventos de teclado de cada uno de los componentes. Es muy útil para que independientemente del componente sobre el que se esté, la gestión mediante teclas sea única por ejemplo para la navegación. Por ejemplo:

```
private void Form1_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
    this.Text = e.KeyCode.ToString();
}
```

Coloca varios componentes en el formulario (botones, labels, al menos un textbox,...). Y prueba el anterior código sin y con keypreview.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

Label

Es un elemento informativo. Ya sea de forma estática para informar sobre algún otro componente que esté cerca (P. ej. un textbox) o en tiempo de ejecución se puede modificar para informar de alguna evolución al usuario.

Propiedades más destacadas:

AutoSize: Cuando esta propiedad está establecida en true, el alto y el ancho del control se ajustan automáticamente para mostrar todo el contenido del control. Esta propiedad se establece normalmente en true cuando se utiliza un control *Label* para mostrar texto de distinta longitud como, por ejemplo, el estado de un proceso de la aplicación. Esta propiedad puede utilizarse asimismo cuando la aplicación muestra texto en varios idiomas y el tamaño del texto puede aumentar o disminuir según la configuración de idioma de Windows.

Enabled: Si está a false, el componente aparece desactivado y no responde a eventos. Cambia su aspecto visual a gris.


Image: Imagen de fondo de la etiqueta.

Text: Contenido de texto de la etiqueta.

TextAlign: Indica dónde se desea que aparezca alineado el texto. Para que esta propiedad tenga efecto, debe estar el *AutoSize* a False.

Visible: Booleano que indica si se muestra o no el componente. Ponerlo a true o a false es equivalente a llamar a los métodos *show* y *hide* respectivamente.

El componente *Label* acepta algunos eventos y métodos ya vistos para el formulario como por ejemplo *MouseMove* y *Show*.

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD	COMPETENCIA					

Button

Muchas de las propiedades y métodos ya vistos sirven también para el botón: Text, Image, BackColor, Font... De hecho un botón puede mejorarse bastante visualmente mezclando imágenes con texto.

Propiedades

Anchor: La propiedad Anchor, existente en un gran número de controles, nos permite anclar dicho control a uno o varios bordes del formulario.

Cuando un control es anclado a un borde, la distancia entre el control y dicho borde será siempre la misma, aunque redimensionemos el formulario.

Dock: A través de la propiedad Dock de los controles, podremos acoplar un control a uno de los bordes de un formulario, consiguiendo que dicho control permanezca pegado a ese borde del formulario en todo momento.

Para seleccionar el tipo de acople, haremos clic en el botón que tiene la propiedad Dock en la ventana de propiedades, y que nos mostrará un guía de los tipos de acople disponibles.

FlatStyle: Marca el estilo con el que aparece un botón: Normal, predefinido por el sistema, plano (bidimensional) o tipo popup (se resalta al pasar el ratón por encima).

Tag: Propiedad auxiliar que sirve al programador para guardar cualquier tipo de objeto o valor que desee asociar a un componente en un momento dado.

Métodos:

PerformClick: Ejecuta un evento clic del botón. Cómodo para ejecutar el código asociado a dicho botón desde otros puntos del programa.

Eventos:

Click: Aparece cuando se pulsa sobre el botón con el ratón, cuando el botón tiene el foco y se pulsa la tecla enter o cuando este está asociado a Enter o Escape mediante AcceptButton o CancelButton del formulario.

Enter/Leave: Se disparan cuando se entra/sale del componente, es decir, cuando coge el foco o cuando se libera del mismo. (Existe, de hecho, el evento gotFocus pero es de más bajo nivel que enter y se utiliza en contadas ocasiones).

No deben confundirse con **MouseEnter** y **MouseLeave** que también existen pero sólo se accionan si el causante de acceso al button es el ratón.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Compartición de código entre controles

Hay ocasiones en que cierto número de controles de una aplicación tienen un código asociado muy similar. Un claro ejemplo es una calculadora con gran cantidad de botones en el que muchos de ellos realizan acciones similares.

En estos casos la primera idea es establecer un método por cada evento que se produzca en cada componente. Sin embargo sería probablemente más cómodo establecer un único método y asociarlo a todos los eventos que nos interesen.

Esta asociación se puede hacer tanto en el diseñador como por código. En el segundo caso se realiza mediante el objeto EventHandler (manejador de eventos) y debe especificar el procedimiento que va a usar el evento de la siguiente forma:

```
this.Componente.Evento += new System.EventHandler(NombreMetodo);
```

El método debe cumplir simplemente que no devuelve nada y que tenga dos parámetros tal y como vimos, uno que representa el receptor del evento (se suele denominar *sender* y es de tipo *object*) y otro que contiene información adicional sobre el evento (se suele denominar *e* y es de tipo *EventArgs* o algún derivado dependiendo del evento).

```
void NombreMetodo(object sender, EventArgs e)
```

Ejemplo 1: Botones cuyo campo text cambia a mayúsculas al pulsarlos. Lo habitual sería hacer

```
ButtonX.Text=ButtonX.Text.ToUpper();
```

en cada uno de los eventos de cada botón. Vamos a simplificarlo:

1. Colocar en un formulario 3 botones.
2. Hacer doble clic sobre uno de ellos y acceder al código, pero cambiamos el procedimiento de forma que su cabecera quede:

```
void BotonesClick(object sender, EventArgs e)
```

3. Escribimos un único código común para los tres:

```
((Button)sender).Text=((Button)sender).Text.ToUpper();
```

Las conversiones son necesarias porque el sender es un object.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

4. Ahora se debe realizar la línea donde se enlaza el evento con el método. En InitializeComponent, debajo de la inicialización de cada componente se escribe:

```
this.button1.Click += new System.EventHandler(this.BotonesClick);
this.button2.Click += new System.EventHandler(this.BotonesClick);
this.button3.Click += new System.EventHandler(this.BotonesClick);
```

5. Otra posibilidad sería en la hoja de diseño, en la ventana de propiedades ir a Eventos y seleccionar el método que queramos (BotonesClick) en el evento deseado.

Ejemplo 2: Podemos explotar la propiedad Tag con el objetivo de que cada botón haga una cosa distinta:

En el evento Form Load o en el constructor:

```
button1.Tag = Color.Aqua;
button2.Tag = Color.Cornsilk;
button3.Tag = Color.ForestGreen;
```

Si no tuvieras la clase Color tienes que meter el namespace Drawing:

```
using System.Drawing;
```

En el Evento de código compartido (*BotonesClick*)


```
this.BackColor = (Color)((Button)sender).Tag;
```

O también si quisiéramos hacer distintas tareas según el componente podemos hacer una sucesión de sentencias if (No está permitido hacer un Case con objetos. Habría que hacerlo, por ejemplo, pasando el objeto a ToString). Esto solo sería útil si hubiera una parte común, si no es mejor hacerlo en distintas funciones.

```
if (sender == button1)
    this.Text = "Se ha pulsado el Primer Botón";

if (sender == button2)
    this.Text = "Se ha pulsado el Segundo Botón";

if (sender == button3)
    this.Close();
```

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Orden de tabulación y acceso a componentes mediante ALT

El manejo de la aplicación mediante el teclado es muy importante de cara a la usabilidad y accesibilidad. Dos formas muy típicas de realizar esto es mediante el tabulador para ir de un componente al siguiente y mediante al tecla ALT que nos permite viendo una letra subrayada en dicho componente accionar al mismo.

Todo esto se controla mediante una serie de propiedades de los componentes.

TabIndex: Propiedad que establece el orden de tabulación de los componentes. Al moverse por un formulario con la tecla TAB, el orden será el marcado por el número aquí fijado.

Para ver un “mapeado” del orden en que están los componentes en un formulario debe estar el visual Studio en modo experto en *Herramientas → Configuración*, después se puede ir a

Ver->Orden de Tabulación (En SharpDevelop es *Formato->Mostrar Orden de Tabulación*)

Se puede cambiar el orden en dicho mapeado o directamente en el TabIndex.

En el caso de las *Label*, la tabulación no se especifica para en dicho componente sino que salta al siguiente en el orden en el que están. Es típico para uniones *Label-TextBox*

TabStop: Propiedad booleana que indica si se desea que un componente coja o no el foco mediante movimiento con la tecla TAB. Se pondrá a *False* si se desea que el usuario no pueda parar ahí.

Acceso directo


En general, para etiquetas, menús y botones suele existir la posibilidad de acceder a ello mediante la combinación de la tecla Alt+Letra, siendo la letra una que está subrayada en el texto que muestra el componente. Para hacer este efecto, llega con anteponer el símbolo & a la letra que se desea que sea de acceso directo.

Por ejemplo: Colocar en un botón la palabra &Aceptar y en otro la palabra A&rquivo.

En el diseño aparecerán Aceptar y Arquivo. Y durante la ejecución del programa se podrá acceder a ellos mediante ALT+A y ALT+R respectivamente.

En el caso de las etiquetas se accede al componente siguiente según el orden marcado por el *TabIndex*. Además hay que tener en cuenta la propiedad *UseMnemonic*.

UseMnemonic: Indica si se puede establecer un carácter de acceso directo a una *Label* mediante la tecla Alt. Si *UseMnemonic* es *true*, al colocar el símbolo & delante de un carácter del campo *Text*, este aparece subrayado en tiempo de ejecución y permite el acceso al componente contiguo a la etiqueta. Si está a *False* aparecerá directamente el símbolo &.

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD	COMPETENCIA					

Para los ejercicios de Windows Forms, aunque no lo indique el enunciado, debe tenerse **SIEMPRE** en cuenta: Orden de tabulación; Icono y título de formularios; Acceso a botones (y en el futuro a menús) con ALT; Uso de AcceptButton y CancelButton si se considera lógico.

Ejercicio 1

En un formulario con título "Mouse Tester" coloca dos botones. Debe actuar de la siguiente forma:

- Cada vez que el ratón esté dentro del formulario se debe indicar la coordenada en el título (p. ej: x:100, y:212). También debe hacerlo aunque se esté encima de algún componente. Se debe hacer en una única función asociada al evento MouseMove.
- Cuando el puntero está fuera del formulario, se restablece el valor del título que había en tiempo de diseño.
- Los botones "emularán" los botones del ratón (izquierdo y derecho) cambiando de color si dicho botón del ratón está pulsado sobre el formulario imagen. En caso de un ratón con otros botones dicha pulsación coloreará los dos Button al mismo tiempo. Este efecto será solo sobre el formulario, no es necesario sobre los componentes del mismo.
- Además si se pulsa alguna tecla, dicha tecla debe aparecer como título del formulario. Si se pulsa ESC, entonces se restaura el título del formulario.
- Cambia el efecto anterior para que lo que aparezca no sea la tecla, si no el carácter pulsado. Haz que el uso de este efecto o el del punto anterior se puedan seleccionar mediante directiva de compilación (#define, #if,...)

Al salir debe preguntar al usuario si está seguro y cancelar en caso negativo.

Ejercicio 2

Crear un formulario con las siguientes características:

- Habrá un botón salir que termina la aplicación. También se puede salir con la tecla ESC
- Antes de salir se pedirá confirmación al usuario.
- Habrá 3 TextBox y un botón Color. En las 3 textbox se puede meter números RGB (0-255) y al pulsar el botón se cambia el color del fondo. También si se pulsa ENTER.
- Otro TextBox en el que se escribe el Path de una imagen, esta se cargará al pulsar otro botón asociado de fondo en una etiqueta.
- Al pasar de los textboxes de color al de image, la tecla enter debe "pulsar" el botón de carga de imagen. Por supuesto al volver a los de color debe volver a pulsar "color".
- Cambiar el icono que viene por defecto
- El formulario será normal pero su tamaño no se podrá modificar. Tampoco tendrá controles de maximización ni minimización. Aparecerá siempre centrado en la pantalla y no se verá en la barra de tareas.
- El puntero del ratón será una mano señalando.
- Cada vez que el ratón pase por encima de algún botón este cambiará de color, restaurando el color original cuando el ratón salga del mismo.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

TextBox

Este control ya lo hemos usado como ejemplos en algunos casos y se ha visto que su uso básico es muy sencillo y similar a otros como Button o Label. En este apartado vamos a profundizar un poco más en estacaja de edición.

Propiedades habituales

Text. Cadena con el texto del control.

Multiline. Permite establecer si podemos escribir una o varias líneas. Por defecto contiene False, por lo que sólo podemos escribir el texto en una línea.

En este modo existe un array de líneas (**lines**) que nos permite acceder línea a línea.

WordWrap. En controles multilínea, cuando su valor es *true*, al llegar al final del control cuando estamos escribiendo, realiza un desplazamiento automático del cursor de escritura a la siguiente línea de texto para poder escribir texto sobre él. Si está a *false*, para pasar a la línea siguiente es mediante el ENTER. Equivale al *Ajuste de línea* del Notepad.

ReadOnly. Permite indicar si el contenido del control será de sólo lectura o bien, podremos editarlo.

CharacterCasing. Esta propiedad, permite que el control convierta automáticamente el texto a mayúsculas o minúsculas según lo estamos escribiendo. Opciones: *Normal, Upper, Lower*.

MaxLength. Valor numérico que establece el número máximo de caracteres que podremos escribir en el control.

PasswordChar. Carácter de tipo máscara, que será visualizado por cada carácter que escriba el usuario en el control. De esta forma, podemos dar a un cuadro de texto el estilo de un campo de introducción de contraseña.


AutoSize. Cuando esta propiedad tenga el valor *True*, al modificar el tamaño del tipo de letra del control, dicho control se redimensionará automáticamente, ajustando su tamaño al del tipo de letra establecido.

AcceptsReturn y AcceptsTab: Si son TRUE se aceptan las teclas ENTER y TAB como parte del texto (pasará a la línea siguiente o meterá una tabulación). Si son FALSE, ENTER y TAB funcionarán como teclas habituales de formulario, en este caso se usará CTRL+ENTER/TAB para realizar esa acción en el texto.

Métodos:

Focus: Para obtener el foco.

AppendText (string): Añade texto al texto actual y coloca el cursor al final de dicho texto. La diferencia con `Text+="nuevoTexto"` es que en este segundo caso el cursor queda al principio y no hace scroll.

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Eventos:

TextChanged: Se produce si cambia el texto en la propiedad *Text*.

Herramientas de Edición

Selección de textos: Mediante las siguientes propiedades accesibles **en tiempo de ejecución**:

HideSelection: Muestra u oculta lo que se seleccione por código (no por el usuario)

SelectionStart: inicio de selección

SelectionLength: Longitud en caracteres de la selección

SelectedText: Cadena de texto seleccionada

Y los métodos:

Select(inicio, longitud): Selecciona un texto desde la posición inicio y de tamaño longitud.

SelectAll.: Selecciona la totalidad del texto.

Ejemplo: colocar en un formulario Un textBox con una frase en su interior y dos botones. Escribir el siguiente código en el evento click de los botones:

Boton1:

```
textBox1.SelectionStart=3;
textBox1.SelectionLength=5;
```


Boton 2:

```
textBox1.HideSelection =!textBox2.HideSelection;
```

Edición: Existen los métodos *Clear*, *Copy*, *Cut*, *Paste*. Ver el manejo en el MSDN. Además se puede ver también un ejemplo de cómo se maneja el *portapapeles (clipboard)*.

Modificar y Deshacer: Mediante la propiedad booleana *Modified* se puede saber si un texto ha sido modificado. Posteriormente se puede poner a *false*.

Y con CanUndo se establece si se puede recuperar el valor anterior usando el método *Undo* o eliminar los datos de recuperación usando *clearUndo*.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

Múltiples formularios

Como ya sabemos, un formulario es una clase. Como ya hemos visto si queremos que aparezcan nuevos formularios, tendremos que crear la clase como un nuevo formulario dentro de la aplicación con sus componentes y posteriormente lo instanciaremos desde el lugar deseado del formulario principal (o desde el llamante, ya que cualquier formulario puede instanciar otro).

Habíamos visto que esto se podía hacer así:

```
f2 = new Form2();
f2.Show();
```

Siendo f2 un objeto de la clase *Form2* (si no le hemos cambiado el nombre) que puede ser definido como objeto local o como propiedad dentro de la clase *Form1*.

La primera línea lo que hace es instanciar el objeto formulario a partir de la clase y la segunda lo muestra en pantalla. Ojo, no se debe modificar nada del segundo formulario si previamente no se ha instanciado.


Una vez que el formulario no interese, puede hacerse desaparecer de dos maneras:

- Mediante un *Close* si ninguna de sus propiedades va a ser necesaria posteriormente, así se libera la memoria.
- Mediante *Hide*: En este caso realmente lo que se está haciendo es ocultar el formulario, lo cual es necesario en caso de que algunas de las propiedades o datos que contiene quieran ser utilizados ya que no desaparecen de la memoria.
-

Formularios de diálogo o modales

Para mostrar un Formulario en forma modal (es decir, que bloquee la ejecución del código hasta que sea cerrado) se usa el método **ShowDialog**. Y si se desea que al cerrar dicho formulario se devuelva un valor (en el estilo del *MessageBox*) se puede usar la propiedad **DialogResult**. Esta propiedad se puede establecer en el formulario secundario o a través de los botones que lo vayan a cerrar. El ShowDialog hace que se pase por el FormLoad.

Ejemplo: Abrir una aplicación con dos formularios. En el primero (y principal) no hace falta colocar nada. En el segundo se colocara una textbox, una label que ponga Contraseña y dos Botones, uno de aceptar y otro de cancelar.

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

En el evento Load del **formulario 1** se escribirá el siguiente código:

```
Form2 f = new Form2();
DialogResult res;
res = f.ShowDialog(); //Aquí se para la ejecución del programa

switch (res)
{
    case DialogResult.OK:
        if (f.textBox1.Text.ToUpper() == "AAA")
            MessageBox.Show("Contraseña Aceptada", "Mi Aplicación",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        else
        {
            MessageBox.Show("Contraseña Invalida", "Mi Aplicación",
                MessageBoxButtons.OK, MessageBoxIcon.Stop);
            this.Close();
        }
        break;
    case DialogResult.Cancel:
        MessageBox.Show("Operación Cancelada", "Mi Aplicación",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        this.Close();
        break;
}
```


Ahora en el **segundo formulario** accede a las propiedades de los botones y en particular a DialogResult. Ahí pone en el botón Aceptar : DialogResult.OK;

Y en el botón Cancelar: DialogResult.Cancel;

En el momento que se pulsa un botón teniendo valor esta propiedad el formulario se oculta (**¡no se cierra!**). Lo mismo pasa si el usuario cierra el formulario 2 mediante la X de la esquina superior derecha, lo que es equivalente a generar un código de cancelación (*DialogResult.Cancel*). También en este caso el diálogo se oculta, no se cierra. Esto es así por si se desea leer algún dato de dicho cuadro de diálogo que pueda ser necesario para continuar el programa.

Si quisiéramos liberar la memoria, tendríamos que hacerlo nosotros directamente (por ejemplo asignando *null* a *f*)

También se puede hacer por código asignando en los eventos Clic de los botones a la propiedad DialogResult **del formulario** un valor.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

CheckBox

Este componente es una caja de chequeo que puede estar activada o desactivada en un momento dado. La gestión de **la activación y desactivación es automática** según el usuario pulse sobre ella, aunque se puede desactivar esta automatización mediante la propiedad *Autocheck*:

También dispone de un estado de indeterminación si se activa el Tercer Estado (no lo veremos), el cual aparece como un chequeo en un tono gris. Aparece al pulsar sucesivamente sobre el control.

Propiedades

Autocheck: Indica si se activa o desactiva de forma automática la casilla.

Checked.: Valor lógico que devuelve True cuando la casilla está marcada, y False cuando está desmarcada.

Appearance: Normal o de Botón. Indica si la apariencia es la de una casilla de verificación o la de un botón que puede quedar presionado (*checked*) o sin presionar (*unchecked*).

Eventos

CheckedChanged: Indica si cambian las propiedades *Checked* respectivamente.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Radiobutton y GroupBox

El *radiobutton* es un componente también de chequeo pero de uno entre varios. Es decir, el *radiobutton* nunca va solo sino que al menos va en agrupaciones de dos o más *radiobuttons*. Se usan para realizar selecciones excluyentes entre sí (sí/no, Mujer/hombre, etc.).

El *GroupBox* es un panel de agrupación de Controles para independizar distintos grupos. Es lo que denomina un componente contenedor. El formulario es otro componente contenedor, pues puede contener controles.

Ejemplo: Mostraremos la selección del tipo de letra para una etiqueta: Times New Roman, Arial o Comic Sans MS.

Colocad en un formulario 3 radiobutton con el nombre de cada tipo de letra en cada uno.

Colocad también una etiqueta con cierto texto.

Escribid el siguiente código en el evento checked **de cada** RadioButton

```
this.label1.Font = new System.Drawing.Font(radioButtonTimes.Text, 16);
```

```
this.label1.Font = new System.Drawing.Font(radioButtonArial.Text, 12);
```

```
this.label1.Font = new System.Drawing.Font(radioButtonComic.Text, 10);
```

Ejecutar el programa y ver como cambia.

Supongamos que también queremos hacer selección de colores. Para ello ponemos 3 nuevos radiobutton pero para seleccionar color. Además marcamos el Backgroundcolor para hacer más fácil (y visual) la selección.

Escribir el código en los eventos checked:


```
this.label1.ForeColor = radioButtonXXXX.BackColor;
```

Donde XXXX representa a cada Radiobutton.

Al ejecutar el programa aparece el problema, y es que tanto los de font como los de colores son excluyentes entre sí. Para evitar esto, se puede agrupar uno de ellos (o ambos) en un GoupBox, por ejemplo el de los colores. De esta forma se independizan.

Así el *GroupBox* sirve para reagrupar y diferenciar grupos de componentes en un formulario. Tanto a nivel visual como de programación.

Las propiedades interesantes de los *RadioButton* son *Checked* y *Appearance*, y su evento *CheckedChange*.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

Panel

Este es un control contenedor de otros componentes y nos va a ayudar en la agrupación visual y funcional de nuestra aplicación. Sin embargo tiene un par de matices que lo va a hacer útil para ciertas funciones:

- Es un control que puede no tener ningún tipo de borde, lo cual puede interesar a nivel visual. También tiene la posibilidad de tener borde efecto 3D. Esto se manejará mediante la propiedad **BorderStyle**.
- Es un control que permite Scroll. Es decir, si lo que contiene no cabe en el área visible, permite hacer Scroll siempre que esté la propiedad *AutoScroll* a TRUE (Existen otras propiedades para el manejo del Scroll. Verlas en el MSDN).

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Menús

La mayoría de los programas disponen de un menú que contempla todas las acciones que puede realizar cierta aplicación. Para crear en un programa un menú se necesitan como mínimo dos componentes: El contenedor del menú y los ítems del menú.

Como contenedor de menú desde el Framework 2.0 vamos a disponer de dos contenedores distintos, el llamado *MenuStrip* y el *ContextMenuStrip*.

Nota: Si trabajamos con Framework 1.1 tendremos los equivalentes MainMenu y ContextMenu que son menos versátiles que los aquí explicados y su uso es incluso más sencillo.

MenuStrip

Es el menú que aparece en la parte superior de un formulario. Hay una serie de opciones o ítems normalmente desplegables y que lleva la otras opciones o ítems.

Para crear uno escogemos en la barra de herramientas el componente *MenuStrip* y lo arrastramos al formulario.

En principio aparece como un componente no visual en la parte inferior. Se puede comprobar que apenas tiene propiedades. Con todo aparece la posibilidad de ir escribiendo los diversos ítems directamente sobre el formulario. El que ahí se escribe es el campo *Text* de cada objeto de clase *ToolStripMenuItem* ya que el *MenuStrip* contiene una colección de *ToolStripMenuItem*.

La colección donde guarda los elementos del menú es **Items**, y si accedemos a ella desde propiedades permite un uso más avanzado de estos elemento.


Otras **propiedades** que nos interesarán de **ToolStripMenuItem** son:

Text. Contiene una cadena con el literal o texto descriptivo de la opción de menú. Se puede usar el ampersand (&) para acceso mediante ALT+TECLA como se hacía con las *Label* y los *Button*.

Si en este campo se coloca un guión simple (-) el ítem será una línea de separación y no responderá a eventos.

Checked/CheckOnClick. La primera marca/desmarca la opción. Cuando una opción está marcada, muestra junto a su nombre un pequeño símbolo de verificación (tick). En el caso de tener una imagen, esta aparece con un marco. Esta propiedad no es automática como ocurría en la *Checkbox*, si no que hay que activarla o desactivarla segundo se desee a menos que se active la propiedad **CheckOnClick**.

ShortcutKeys. Se trata de un atajo de teclado, o combinación de teclas que nos van a permitir ejecutar la opción de menú sin tener que desplegarlo. Al elegir esta propiedad aparecerá un formulario para seleccionar el atajo.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

ShowShortcutKeys. Permite mostrar u ocultar la combinación de teclas del atajo de teclado que haya asignado una opción de menú.


Image: Establece la imagen que se desea que aparezca en el margen del *ToolStripMenuItem*.

DropDownItems: Colección de ítems del submenú. Lo normal es crearlo con el diseñador de menús.

En el formulario podemos encontrar la propiedad **MainMenuStrip**: Propiedad que hace referencia el menú que puede ser controlado desde el teclado. Esto es porque se pueden definir varios *MenuStrip* y todos están asociados al formulario. Para hacer que se vean unos u otros se usa la propiedad *Visible*.

El evento principal asociado a un menú es el **click** sobre cada uno de los ítems. En ocasiones puede ser interesante hacer compartición de código entre varias opciones o entre botones del formulario y opciones del menú.

Además de los *ToolStripMenuItem* en cada apartado del menú se pueden colocar *comboboxes* (*ToolStripComboBox*) o *TextBoxes* (*ToolStripTextBox*) con propiedades similares a los componentes ya vistos. Se hace seleccionando el componente deseado en la combobox del diseñador.

Puedes crear un menú totalmente estándar de forma automática usando el **Smart Icon**  seleccionando Insertar Elementos Estándar. Pruébalo.

Menú contextual (ContextMenuStrip):

Es el menú que aparece al pulsar el botón secundario del ratón. La construcción es similar al *MenuStrip* pero sin cabeceras.

Una vez construido se debe enlazar con algún componente mediante la propiedad *ContextMenuStrip* del componente.

Si un menú contextual es idéntico a un submenú del principal, lo mejor es diseñar el contextual y luego asociarlo al ítem del principal mediante la propiedad *DropDown*.

Hay que tener cuidado con algunos componentes (como *textbox*) que ya tienen menú contextual y si añadimos sobre escribimos el existente.

En el [Apéndice VII](#) puedes ver el uso del *ToolStrip* que es un complemento habitual al Menú.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD	COMPETENCIA						

PictureBox

Componente para la visualización y manejo de imágenes de diversos tipos: JPEG, BMP, PNG GIF, y algunas de metaarchivo (WMF)

Image: Establece la imagen que se ha de visualizar en el PictureBox. Esta a su vez contiene propiedades sobre la imagen que contiene: Alto y ancho, resolución, modo de color, ... Y también dispone de métodos para guardarla en el disco duro o rotarla.

```
pictureBox1.Image = System.Drawing.Image.FromFile(nombreadarchivo);
```

Como la clase *Bitmap* hereda de la clase *Image*, también es posible hacer:

```
pictureBox1.Image = new Bitmap(nombreadarchivo);
```

```
// o también
```

```
Bitmap a=new Bitmap(nombreadarchivo);
/* ..... */
pictureBox1.Image = a;
```


SizeMode: Establece el modo en el que se muestra la imagen:

- *PictureBoxSizeMode.Normal:* el objeto Image se coloca en la esquina superior izquierda de PictureBox y se recorta la parte de la imagen que es demasiado grande para PictureBox.
- *PictureBoxSizeMode.StretchImage:* hace que la imagen se estire o encoja para ajustarse a PictureBox. Sufre deformación.
- *PictureBoxSizeMode.AutoSize:* hace que el control cambie de tamaño para ajustarse siempre a la imagen.
- *PictureBoxSizeMode.CenterImage:* hace que la imagen se centre en el área de cliente.
- *PictureBoxSizeMode.Zoom:* La imagen se ajusta al PictureBox pero sin deformarse. Quedan bandas horizontales o verticales si es necesario.

BorderStyle: Estilo del borde, si se desea que tenga efecto 3D o no.

Ejemplo de uso de PictureBox con Panel:

1. Colocar en un formulario un Panel y dentro de este panel un PictureBox ajustado a la esquina superior izquierda.
2. La propiedad del PictureBox SizeMode colocadla a AutoSize y cargar una imagen grande en el picture box (que no quepa en el Panel)
3. La propiedad AutoScroll del Panel colocadla a TRUE.
4. Ejecutadlo y ver como automáticamente se maneja el campo de visión mediante las barras de scroll.

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

ListBox

Componente usado habitualmente para disponer de una serie de elementos textuales (aunque podría contener otros) de cara al usuario de forma que este pueda escoger uno o varios elementos de esa lista, o añadir y quitar elementos de la misma.

Propiedades:

Items: Contiene la lista de valores que visualiza el control. Se trata de una colección de objetos (tipo *ListBox.ObjectCollection*), de manera que el contenido de la lista puede ser tanto tipos carácter, como numéricos y objetos de distintas clases. Al seleccionar esta propiedad en la ventana de propiedades del control, y pulsar el botón que contiene, podemos introducir en una ventana cadenas de texto para el control.

Durante el código, al ser una colección, para añadir, eliminar, buscar, etc... elementos podemos usar los métodos vistos en el capítulo de Colecciones (*add, remove, clear, contains,...*).

Se deben controlar las excepciones que se puedan producir en colecciones.

IntegralHeight: Los valores de la lista son mostrados al completo cuando esta propiedad contiene *True*. Sin embargo, al asignar el valor *False*, según el tamaño del control, puede que el último valor de la lista se visualiza sólo en parte.

MultiColumn: Visualiza el contenido de la lista en una o varias columnas en función de si asignamos *False* o *True* respectivamente a esta propiedad.

ColumnWidth: Ancho de la columna.

SelectionMode: Establece el modo en el que vamos a poder seleccionar los elementos de la lista. Si esta propiedad contiene:

None, no se realizará selección;

One, permite seleccionar los valores uno a uno;


MultiSimple permite seleccionar múltiples valores de la lista pero debemos seleccionarlos independientemente;

MultiExtended nos posibilita la selección múltiple, con la ventaja de que podemos hacer clic en un valor, y arrastrar, seleccionando en la misma operación varios elementos de la lista.

SelectedItem: Devuelve el elemento de la lista actualmente seleccionado.

SelectedItems: Devuelve una colección *ListBox.SelectedObjectCollection*, que contiene los elementos de la lista que han sido seleccionados.

SelectedIndex: Informa del elemento de la lista seleccionado, a través del índice de la colección que contiene los elementos del *ListBox*.

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD		COMPETENCIA				

SelectedIndices: Colección de los índices seleccionados.

Métodos

FindString(cadena) o (cadena, indice): Busca un elemento del ListBox que empiece por la cadena especificada. Se sobrecarga para poder buscar cierta cadena a partir de cierto índice.

Es una función y devuelve el índice de la búsqueda. Si no encontrara nada, devuelve el valor *ListBox.NoMatches* definido en el propio componente.

No distingue entre mayúsculas y minúsculas.

FindStringExact: En este caso busca la cadena exacta.

GetSelected(indice): Devuelve true o false si cierto *índice* está seleccionado. Ojo porque produce excepción de fuera de rango.

SetSelected(indice, bool): Fuerza que el elemento *índice* pase a seleccionado o no seleccionado. Produce excepción.

Eventos:

SelectedIndexChanged: Se produce al hacer/deshacer alguna selección.

Este componente tiene aún más propiedades y métodos a tener en cuenta para su manejo como el control de elementos por separado, posibilidad de seleccionar por coordenadas del ratón, dibujar,... Más información en la ayuda de Microsoft

Ejemplo: Colocar un Listbox en un formulario con un botón, un label y dos textbox. En el clic del botón programáis parte del código que viene en el MSDN:

```
// Forzamos a múltiples columnas y permitimos selección extendida
listBox1.MultiColumn = true;
listBox1.SelectionMode = SelectionMode.MultiExtended;

// Rellenamos con 50 items.
for (int x = 1; x<=50; x++)
    listBox1.Items.Add("Item " + x.ToString());

// Forzamos la selección de 3 ítems.
listBox1.SetSelected(1, true);
listBox1.SetSelected(3, true);
listBox1.SetSelected(5, true);

// Mostramos en la consola de depuración el segundo ítem seleccionado
System.Diagnostics.Debug.WriteLine(listBox1.SelectedItems[1]);
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

```
// Mostramos en la consola de depuración el índice del 1er ítem.
System.Diagnostics.Debug.WriteLine(listBox1.SelectedIndex[0]);
```

Observa el uso de la consola de depuración (si no aparece se puede ver en *Ver->Resultados*).

Escribimos también código para el evento *SelectedIndexChanged*:

```
System.Diagnostics.Debug.WriteLine(listBox1.SelectedItem.ToString() + " Índice:"
+ listBox1.SelectedIndex.ToString());
```

```
label1.Text= listBox1.SelectedIndex.ToString();
```

Finalmente hacemos un único *TextChanged* para las dos *TextBoxes* y escribís:

```
void TextBox_TextChanged(object sender, System.EventArgs e) {
    int i;

    listBox1.SelectionMode=SelectionMode.One;
    if (sender==textBox1)
        i = this.listBox1.FindString(this.textBox1.Text);
    else
        i = this.listBox1.FindStringExact(this.textBox2.Text);

    if (i!= ListBox.NoMatches)
        listBox1.SelectedIndex=i;
}
```

Existe un componente denominado *ListView* que es más avanzado que el *ListBox*, permitiendo varias vistas, pero también es más complejo de usar. Podrás usarlo de forma opcional en los ejercicios.

ComboBox (Lista desplegable)

Este componente es una mezcla del *ListBox* y del *TextBox*. Funciona de forma similar a la *ListBox* y para acceder al texto que aparece cuando no está seleccionado, está el campo *text*.

DropDownStyle: Selecciona el estilo de la Combo. Puede ser:

Simple: Aparece la zona de edición y la lista. Al escribir en la caja, busca el elemento.

DropDown: La lista no aparece hasta que se le da al botón de desplegar, pero es editable el campo de texto.

DropDownList: Como el anterior, pero el usuario no puede editar el campo de texto.

El resto de las propiedades es común a las del *ListBox* a las que se le suma la propiedad *Text* del *TextBox*.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Timer

Control no visual que provoca la generación de un evento cada cierto tiempo. Así podemos ejecutar cierto código cada cierto tiempo. Es útil para realizar tareas en segundo plano o para simular bucles que requieran cierta sincronización temporal.

Es un componente no visual, de forma que en el momento que se añade al formulario no aparece en este si no que se muestra en la parte inferior del diseñador de formularios.

Propiedades

Enabled: Si está a true el *timer* genera eventos, si está a false no.

Interval: Indica cada cuanto tiempo (en milisegundos) se genera un evento *Tick*. Por debajo de 60 ms no acelera.

Métodos

Start/Stop: Activa/desactiva el *timer* (activan o desactivan el *Enabled*)

Evento:

Tick: Evento que se genera de forma automática cada *Interval* milisegundos.

ToolTip

Representa una pequeña ventana emergente (de tono amarillento) que aparece al colocar el ratón sobre algún componente. Suele servir de indicación o ayuda al usuario.

La forma de trabajo es la siguiente: se establece un objeto *ToolTip* para indicar tiempos de aparición del mensaje y en dicho *ToolTip* se establece una colección de mensajes asociados a distintos componentes.

Es una especie de tabla hash en la que la clave es el componente y el dato es el mensaje.


Si se desea tener mensajes con distintos tiempos de respuesta se pueden colocar más de un *ToolTip* y asociarlo a distintos componentes.

Uso básico del ToolTip:

Un *ToolTip* es un componente en principio no visual ya que se trata de una colección de mensajes. Una vez arrastrado hacia un formulario aparece en la parte inferior. Para un uso simple no es necesario tocar ninguna de las propiedades.

Una vez establecido el *ToolTip*, aparecen en todos los componentes del formulario un nuevo campo en las propiedades denominado "*ToolTip en ToolTip1*". Esa propiedad realmente no pertenece al componente si no que se guarda en el propio *ToolTip*.

En cualquier caso en esa propiedad se coloca la ayuda deseada como texto y ya funcionaría.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Control de tiempos:

Para un uso más profundo se puede jugar con las distintas propiedades de tiempo del *ToolTip*: *AutomaticDelay*, *AutoPopDelay*, *InitialDelay*, *ReshowDelay*. Son propiedades que nos permiten jugar con el tiempo que tarda en surgir, en desaparecer, etc.. Se pueden ver en el MSDN.

A partir del Framework 2.0 existen también propiedades para cambiar la forma, el color y la aparición de iconos entre otros.

Cambio de texto en tiempo de ejecución:

Como ya se comentó, la propiedad *ToolTip* no pertenece a cada componente aunque así lo parezca en la ventana de propiedades. Así si queremos cambiar u obtener el texto de un componente se usarán los siguientes **métodos del componente *ToolTip***:

GetToolTip: Función a la que se le pasa un componente y devuelve el texto asociado.

```
public string GetToolTip (Control control)
```

Por ejemplo:


```
string texto=toolTip1.GetToolTip(this.button1);
```

SetToolTip: Procedimiento al cual se le pasa un control y un texto y lo asocia como *ToolTipText* para dicho componente.

```
public void SetToolTip (Control control, string caption)
```

Por ejemplo:

```
toolTip1.SetToolTip(this.button1, "Nuevo mensaje");
```

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

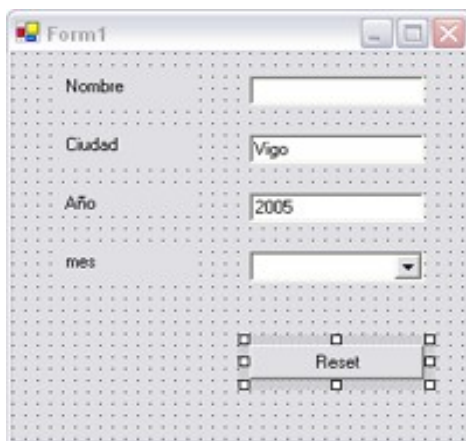
La colección de controles del formulario

Los controles que están en el formulario se encuentran dentro de la Colección *Controls* del mismo y por tanto podemos manejarla como cualquier colección que hayamos visto en temas anteriores.

A su vez, como ya sabemos, algunos controles están también compuestos de colecciones como pueden ser los menús o los *ListBox*.

Veamos como podríamos cambiar de una forma cómoda una serie de controles que hay en un formulario aprovechando las facilidades que nos dan las colecciones.


Ejemplo: Inicializar los Tag con los valores iniciales.



En el evento *Click* del botón

```
foreach (Control c in this.Controls) {
    // Si el control es un TextBox restauramos su valor original
    if (c.GetType() == typeof(TextBox)) {
        if (c.Tag == null)
            c.Text = "";
        else
            c.Text = c.Tag.ToString();
    }

    // Y si se trata del combobox, lo colocamos en el índice inicial
    if (c == this.comboBox1)
        ((ComboBox)c).SelectedIndex = 0;
}
```

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

Crear componentes en tiempo de ejecución

Un componente es un objeto, por tanto para crearlo en tiempo de ejecución basta con llamar a su constructor, inicializarlo con los valores deseados y añadirlo a la colección que le corresponda, ya sea directamente la del formulario u otra como puede ser la de un menú.

Lo primero es declarar el componente como una propiedad más de la clase formulario:

```
private Button btn;
```

Luego en el evento donde se desea que provoque la aparición del botón nuevo se mete el código de inicialización :

```
btn = new Button();
btn.Text = "Púlsame!";
btn.Location = new Point(10, 10);
btn.Size = new Size(100, 20);
btn.Enabled = true;

/* Le añadimos los eventos a los que debe responder
 * Por ejemplo en este caso solo el Click */
this.btn.Click += new System.EventHandler(this.btnClick);

// Lo añadimos a la colección de componentes
this.Controls.Add(btn);

// Creamos el método asociado al evento Click del nuevo botón (Fuera de la
función anterior, por supuesto
void btnClick(object sender, System.EventArgs e) {
    btn.Text="Pulsado";
}
```

Cuando se asignan eventos, hay que tener en cuenta el evento en concreto ya que si por ejemplo en lugar de Click tenemos un **MouseMove**, el manejador de eventos es **System.Windows.Forms.MouseEventHandler** y los argumentos del método son **MouseEventArgs** en lugar de **EventArgs** (más específico). Hay que conocer el método que se desea manejar.

Lo anterior es muy práctico si por ejemplo tenemos que crear una gran cantidad de componentes muy similares: lo realizamos con un bucle de forma que dentro de cada vuelta creamos el componente con las propiedades que nos interesen y luego lo añadimos a la colección Controls.


COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Clave hash de los componentes

La colección de componentes puede ser accedida mediante claves hash, que son los nombres especificados en la propiedad Name de cada componente. Por ejemplo si queremos cambiarle el Text a una etiqueta que se llama "label1" (Ojo, propiedad Name, no tiene porque tener ninguna variable asociada) podemos hacer:

```
this.Controls["label1"].Text = "Nuevo Texto";
```

Esto nos puede valer también para marcar componentes. Por ejemplo podemos darle un nombre que empiece por determinada palabra a una serie de componentes que queramos realizar cierta tarea con ellos con lo que no es necesario crear un array a mayores.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Ejercicio 3

Se abre un formulario en el que mediante un botón se saca un OpenFileDialog (Ver [apéndice III](#)) en el que se verán diversos archivos de imagen (al menos jpg y png) además de todos los archivos. Debe haber también checkbox con la palabra Modal como texto. Una vez pulsado el usuario seleccione una imagen y le de Aceptar en el OpenFileDialog se deben hacer las siguientes tareas:

- Se saca un segundo formulario del tamaño de la imagen y se muestra la imagen en dicho formulario. Si cambias el tamaño del formulario debe cambiar también el de la imagen de fondo deformándose (Usa un pictureBox). El formulario será modal si el checkbox está marcado y no modal en caso contrario.
- El título del formulario secundario será el nombre de la imagen (sin el path) y el del principal "Visor de imágenes"
- Se debe pedir confirmación antes de salir del programa.

Ejercicio 4 (Usa delegados)


Realiza un programa con al menos 2 textbox, un botón y 4 RadioButton que permitan seleccionar entre suma, resta, multiplicación y división. Actuará de la siguiente forma:

- Al pulsar en un RadioButton se selecciona la función a ejecutar mediante una tabla hash con clave el campo Text del RadioButton y valor el delegado.
- Habrá también una etiqueta entre los dos TextBox donde aparece el símbolo de operación que cambiará al cambiar los RadioButton (Guarda el símbolo en el Tag).
- Al pulsar el botón ejecuta dicha función con los operandos que haya en los textbox. Las funciones de operación realízalas como funciones lambda.
- En la barra de título se verá el tiempo de uso en minutos y segundos (formato siempre con dos cifras mm:ss).

Ejercicio 5

Coloca 2 ListBox (la primera permite multiselección, la segunda no), 4 botones, 1 TextBox, 2 label

- Cada botón es una acción (añadir, quitar, traspasar (2 botones)).
- Se añade el texto que hay en la TextBox a la lista 1. Al darle tanto al botón Añadir como a la tecla Enter.
- Se elimina los elementos seleccionados (comprobar se hay alguno) de la lista 1.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					


- Se traspasan los elementos seleccionados de una lista la otra con los botones. Deben quedar en el mismo orden (uso de Insert en posición 0 en vez de add).
- En una label indica cuantos elementos hay en la lista principal y en la otra los índices de los seleccionados.
- El título del formulario debe ser animado, apareciendo una letra del título empezando por el final (efecto scroll) cada 200 ms y cuando se complete, volviendo a empezar. Además debe cambiar también el icono del formulario. Se debe hacer **sólo con un timer**. Los dos iconos que uses para el intercambio deben estar como recursos (ver apéndice VIII).
- Meter ToolTip para los distintos componentes. En el caso de la lista secundaria, el ToolTip debe mostrar la cantidad de elementos que hay.

Opcional: De fondo y como adorno que haya una pelota rebotando en las paredes del formulario. Utiliza un PictureBox y muévelo mediante un timer (Se puede usar el anterior o uno nuevo).

Ejercicio 6

Realizar un formulario que simule el teclado de un móvil antiguo mediante un TextBox y 12 botones (dígitos, * y #). Especificación:

- Nada más arrancar se pedirá mediante un formulario modal un pin de 4 dígitos que no deben verse. Si se mete mal 3 veces, la aplicación se cierra, si no da paso al programa principal.
- Los botones deben ser creados e inicializados en tiempo de ejecución en el evento Load o en el Constructor del Formulario.
- Las pulsaciones de los botones escriben su contenido en el TextBox. Además cuando el ratón pase por encima de cada uno, este cambiará de color resaltándolo y volviendo al color original al salir. Si se aprieta, cambiará a un tercer color que **ya no se restaura**.
- Debe existir también un botón de Reset (este creado en tiempo de diseño) que borra el TextBox y deja todos los botones del color original.
- Tendrá un pequeño menú con dos encabezados: Archivo y Acerca de... En Archivo habrá las opciones Grabar número, Reset, separador y Salir. Acerca de dará información de la app y el autor en un MessageBox con icono de información y botón único.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD		COMPETENCIA					

- Grabar número añade a un archivo de texto seleccionado mediante SaveDialog (txt o todos los archivos) el número que haya en el TextBox (la única comprobación es que haya algo).
- Reset hará lo mismo que el botón y salir saldrá del programa sin pedir confirmación.

Nota: si no se ha visto aún el tema de archivos, el menú Grabar deberá Sacar el SavFileDialog pero no hará nada, quedará como practica para cuando se vea dicho tema.

Opcional: Aplica internacionalización con dos idiomas. Mira el Apéndice VIII para ver cómo realizarlo.

Ejercicio 7

Se desea realizar un editor de texto con las opciones distribuidas en menús. Estás serán:

Archivo

- Nuevo documento
- Guardar texto mediante SaveFileDialog.
- Abrir archivo (debe dejar abrir archivos txt, ini, java, cs, py, html, css, xml y todos los archivos) mediante OpenFileDialog.
- Lista de los archivos recientes (máximo de 5)
- Salir.

Edición


- Deshacer, Copiar, cortar, pegar.
- Seleccionar todo.
- Información de selección (Se describe más abajo).

Herramientas:

- Ajuste de línea (El menú dispondrá de un check).
- Selección de escritura: todo en mayúsculas, minúsculas o normal (Se verán Checks).
- Selección de color de texto y de fondo mediante ColorDialog y cambio de fuente mediante FontDialog.
- Cuadro de Acerca de... modal. Usa una función lambda para lanzar este cuadro.

Además:

- Introduce un ToolStrip (Ver [apéndice VII](#)) para la parte de Archivos, nuevo documento y edición.
- El Tooltip del textbox mostrará los Estadísticos: Muestra Cantidad de frases, cantidad de palabras y cantidad de caracteres.

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

- El editor (TextBox) ocupará siempre todo el formulario.
- Antes de cerrar el programa se pedirá confirmación si el archivo no está grabado tras la última modificación (o si no se grabó nunca).
- Existirá un fichero binario de configuración que se lee al principio del programa (transparente al usuario) y se guarda al final del mismo. La información que debe mantener es:
 - Si estaba activo el ajuste de línea
 - Si estaba trabajando en mayúsculas, minúsculas o normal
 - Color de texto y de fondo
 - Fuente
 - Último directorio de trabajo
 - Lista de archivos recientes
- Mediante la opción del menú Información de selección se podrá sacar un segundo formulario **no modal** en el que habrá en todo momento (si lo desea abrir el usuario) información sobre la selección: Punto de inicio, longitud de la selección. Tendrá también un botón aplicar para poder forzar manualmente una selección.

NOTA: Eventos para detectar cambio de selección:

- Movimiento del Mouse o en MouseDown: Se comprueba si al mismo tiempo que se mueve está el botón izquierdo apretado.

```
if (e.Button==MouseButtons.Left) ...
```


- Tecla Liberada (KeyUp): Se comprueba si al mismo tiempo se pulsa la tecla shift y el cursor.

```
if (e.Shift)
    if (e.KeyCode==Keys.Right) ...
```

Nota: si no se ha visto aún el tema de archivos, los menús que leen y guardan archivos se dejarán exclusivamente con los cuadros de diálogo correspondientes pero no harán nada, quedará como práctica para cuando se vea dicho tema.

Opcional:

- Añade una opción de Imprimir usando PrintDialog.
- La última parte del fichero de configuración realízala mediante un archivo estándar XML o JSON. Documentate sobre las clases existentes en .Net para el manejo de este tipo de archivos y úsalas.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Ejercicio 8 (Solo cuando se haya visto el tema de archivos)

Realizar un nuevo visor de imágenes de la siguiente manera: En el formulario principal habrá un botón “Abrir” mediante el cual se abrirá un OpenFileDialog para seleccionar una imagen en un directorio determinado. Una vez seleccionada dicha imagen se deben hacer las siguientes tareas:

- Se saca un segundo formulario sin bordes y la imagen seleccionada se muestra en el mismo. Debes buscar la manera para que el tamaño del formulario sea el mismo de la imagen.
- Se colocan dos botones (Avance y retroceso) en el primer formulario y se muestra a la imagen siguiente o la posterior del directorio según se pulse un botón u otro.
- También se debe poder hacer esta navegación mediante las flechas del cursor (u otras teclas que elijas), independientemente del control que tenga el foco.
- El título del formulario principal será “Visor de imagenes -<nombre de la imagen>”
- Habrá una etiqueta que informa del directorio actual de trabajo y otra de los datos del archivo imagen: Nombre, tamaño en KB o MB (dependiendo del mismo) y resolución de la imagen.
- Si se pulsa sobre la imagen con el botón derecho saldrá un menú contextual con las opciones Siguiente, Anterior y Cerrar imagen.
- ToolTips en los distintos botones.
- Se debe pedir confirmación antes de salir del programa

Ejercicios opcionales


9. Retoma la clase Aula del tema 3 y diseña una interfaz de usuario para gestionarla.

10. Realizar el juego Jawbreaker o similar (Marcar varias bolas de colores para ir eliminándolas). Ver ejemplos en:

[https://en.wikipedia.org/wiki/Jawbreaker_\(Windows_Mobile_game\)](https://en.wikipedia.org/wiki/Jawbreaker_(Windows_Mobile_game))

<http://www.minijuegosgratis.com/juegos/jawbreaker/jawbreaker.htm>

11. Juego Campo de Minas. El jugador parte de una esquina del formulario y tiene que llegar a la esquina opuesta (en diagonal). Habrá una serie de minas aleatorias invisibles distribuidas por el campo de juego. Mediante un radar (puede ser una label) se le informa cuantas minas hay cerca (por ejemplo en un radio de 50píxels o puedes hacerlo según una cuadrícula con movimiento de cuadrados enteros, en ese caso el radar diría las minas en los cuadrados de alrededor). Puede haber otros elementos como escudos o vidas ocultos o visibles en el camino.

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD COMPETENCIA						

12. Se desea crear un programa de control de bases de datos de personas. Para ello se mantiene un archivo (Usa BinaryWriter y BinaryReader) que contenga todos los datos personales necesarios (que se exponen más adelante). Paralelamente, se dispone de un formulario en el cual aparece una lista con los nombres y apellidos (en formato Apellidos, Nombre) de las personas y se permite hacer las siguientes opciones (mediante botones y menú contextual de la lista):


- Añadir persona (se ve más abajo que hay que hacer)
- Eliminar persona: se eliminará una persona de la lista y de la colección. Previamente se mostrarán los datos y se pide confirmación
- Mostrar datos de la persona seleccionada.
- Salir

Cada botón tendrá su ToolTipText

En el caso de seleccionar “añadir persona” aparecerá un formulario modal que pida al usuario los siguientes datos (que son los que se han de guardar en la colección de estructuras):

- Nombre
- Apellidos
- DNI: validarlo de alguna forma como nº de 8 cifras, además se calculará la letra de forma automática (no se tienen en cuenta DNIs extranjeros)
- Población
- Código postal: validarlo como nº de 5 cifras. Que muestre la provincia a partir de las dos primeras cifras: 36 Pontevedra, 15 La Coruña, 28 Madrid, 08 Barcelona, ...busca los códigos en Internet.
- Fecha de nacimiento: Usa algún componente tipo calendario
- Correo electrónico: Validar que tenga una @ y un punto o usar un MaskedTextBox.
- Si es hombre o mujer
- Realizar las validaciones mediante el evento Validating y el ErrorProvider. También se recomienda el uso de MaskedTextBox donde sea posible y string.Format.


Una vez metidos los datos, el usuario tendrá la opción de pulsar un botón de cancelar (que cierra el formulario previa confirmación) y uno de Aceptar el cual hace que se presente un informe y si está correcto lo guarda en la colección y cierra el formulario, y si no vuelve al formulario de petición de datos.

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD COMPETENCIA						

13. Juego de las letras. Van cayendo etiquetas o botones con una letra o número aleatorio. El usuario tiene que pulsar dicha tecla en el teclado. Si llega a la parte inferior se pierde “una vida”. Cada vez más rápido. Busca información sobre el Timer. Mejóralo a tu gusto.

14. Realiza un procesador de Textos tipo Wordpad (con RichTextBox) y múltiples formularios hijo (MDI: ver apéndice)

Si prefieres plantea tú un proyecto de tu interés en el que se manejen los distintos elementos vistos hasta ahora de .Net

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Apéndice I: La clase control

Es una clase que no hace falta instanciar y nos permite conocer en cualquier momento el estado del ratón en cuanto a posición y pulsación de los botones y también la pulsación de las teclas de modificación (*Alt*, *Ctrl*. y *Shift*) en cualquier instante del programa sin tener que estar asociado a un evento de teclado o ratón de los vistos.

Propiedades de interés (Son de sólo lectura):

ModifierKeys: Detecta si en un momento dado están pulsadas las teclas shift, control y/o alt.

Se pueden comprobar combinaciones mediante OR.


```
if (Control.ModifierKeys == (Keys.Shift | Keys.Control))
....
```

Esto detectaría la pulsación al mismo tiempo de Shift y de Control (Es una detección por bits, ya que $\text{Keys.Shift}=2^{16}$ y $\text{Keys.Control}=2^{17}$).

Esto se puede meter, por ejemplo en un evento Click de un botón para comprobar si a la vez se está pulsando una tecla o una combinación de las mismas.

MouseButtons: Detecta la pulsación de botones del ratón. También funciona como el anterior, pues es una combinación bit a bit.

MousePosition: Clase Point. Devuelve la posición del ratón en pixels.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

Apéndice II: Creación de un formulario por código

Al igual que hacíamos el año pasado, todo esto que hace el diseñador podemos hacerlo nosotros mediante código. Veamos un ejemplo: Elimina el formulario y dentro de la clase *Program.cs* borra todo y escribe el siguiente código:


```
using System;
using System.Windows.Forms;

public class Miformulario : System.Windows.Forms.Form
{
    public Miformulario()
    {
        this.Name = "frmManual";
        this.Text = "formulario creado desde código";
        this.StartPosition = FormStartPosition.CenterScreen;
        this.ClientSize = new System.Drawing.Size(300, 50);
    }
}

public class Principal
{
    static void Main()
    {
        Miformulario f = new Miformulario();
        Application.Run(f);
    }
}
```

No debería ser complicado entenderlo teniendo en cuenta que *Application.Run* llama al gestor de eventos y le pasa el formulario como receptor de esos eventos.

Se podría hacer desde un proyecto vacío incluyendo las mismas referencias que los *using* y estableciendo las propiedades de proyecto como Aplicación de Windows.

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD	COMPETENCIA					

Apéndice III: Componentes de Dialogo predefinidos

SaveFileDialog

Es un formulario modal predefinido mediante el cual podemos seleccionar un directorio y un nombre para un archivo y luego grabar.

El formulario es simplemente un interfaz con el usuario: **NO GRABA**. Eso lo tendremos que hacer por código. Simplemente facilita la petición de datos por parte del usuario.

Ejemplo de uso

```
this.saveFileDialog1.Title = "Selección de directorio para almacenar datos";
this.saveFileDialog1.InitialDirectory = "C:\\";
this.saveFileDialog1.Filter = "texto (*.txt)|*.txt|pascal (*.pas) |*.pas|visual
basic (*.vb)|*.vb|Todos los archivos|*.*";
this.saveFileDialog1.ValidateNames = true;
this.saveFileDialog1.ShowDialog();

// Aquí habría que comprobar que se devuelve OK antes de grabar
StreamWriter s;
s = new StreamWriter(this.saveFileDialog1.FileName);
s.Write(this.textBox1.Text);
s.Close();
```

Más información:

[http://msdn.microsoft.com/es-es/library/system.windows.forms.savefiledialog\(v=vs.110\).aspx](http://msdn.microsoft.com/es-es/library/system.windows.forms.savefiledialog(v=vs.110).aspx)

OpenFileDialog

Cuadro de diálogo con una utilidad similar al anterior pero esta vez es para obtener información para la lectura de un archivo. Tiene propiedades similares al anterior y alguna añadida.

[http://msdn.microsoft.com/es-es/library/system.windows.forms.openfiledialog\(v=vs.110\).aspx](http://msdn.microsoft.com/es-es/library/system.windows.forms.openfiledialog(v=vs.110).aspx)

FontDialog


Para selección de tipos de letra y estilo.

[http://msdn.microsoft.com/es-es/library/system.windows.forms.fontdialog\(v=vs.110\).aspx](http://msdn.microsoft.com/es-es/library/system.windows.forms.fontdialog(v=vs.110).aspx)

ColorDialog

Selección de color.

[http://msdn.microsoft.com/es-es/library/system.windows.forms.colordialog\(v=vs.110\).aspx](http://msdn.microsoft.com/es-es/library/system.windows.forms.colordialog(v=vs.110).aspx)

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Desarrollo de Interfaces				CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	UNIDAD	COMPETENCIA					

Apéndice IV: Formulario MDI

Una aplicación MDI es aquella cuyo formulario principal hace de contenedor de otros formularios. Para hacer una aplicación de este estilo, se requiere definir el formulario principal como MDI y los hijos crearlos en tiempo de ejecución indicando su pertenencia a ese formulario principal.

En el principal se debe colocar (en las propiedades o por código):

```
IsMDIContainer = true;
```

Luego se crea el diseño del o de los hijos. Para indicarle por código que tienen un “padre MDI” se hará a través de la propiedad *MDIParent*.

Ejemplo: Se dispone de un formulario principal MDI y de dos formularios denominados frmHijo y frmFecha. En el hijo se tiene una TextBox que ocupa todo el formulario. En el de fecha hay una etiqueta que muestra la fecha y hora actuales (mediante un Timer y System.DateTime.Now) En el MDI en un menú una opción es sacar el formulario de fecha y otro el de documento (frmHijo).

Primero hay que crear las clases de los hijos frmFecha y frmHijo. Lo hacemos en las dos opciones del menu.

Para el de fecha:

```
frmFecha f=new frmFecha();
f.MdiParent = this;
f.Show();
```

Para el de documento:


```
frmHijo f=new frmHijo();
f.MdiParent = this;
f.Show();
```

La realización de formularios MDI nos permite un par de nuevas opciones en el menú. Por un lado, podemos tener la opción de menú de ventanas:

Creamos un nuevo *MenuItem* y le colocamos la propiedad *MDIList=true*

Por otro lado podemos reordenar las ventanas contenidas en Mosaico, Cascada, etc...

Usaremos la función `this.LayoutMdi(MDILayout.TileHorizontal)` para mosaico horizontal. Ver lo otros enumerados en el MSDN

	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD COMPETENCIA				

En el ejemplo previo, va a interesar tener vario *forms* de documentos, pero solo uno de información de fecha. Para que que el usuario no pueda sacar más, desactivaremos la opción del menú activándola cuando se cierre.

Desactivación: En el clic de ese *MenuItem*

```
menuItem3.Enabled = false;
```

Y en el *Close* de la clase *frmFecha* accedemos al padre MDI mediante la propiedad *MDIParent*:

```
((Form1) this.MdiParent).menuItem3.enabled = true;
```

Finalmente, lo que nos puede interesar de los formularios hijo es poder recorrerlos. El Formulario principal tiene la propiedad *MDIChildren* que es un *Array* que contiene todos los formularios.

IsMDIChild, IsMDIContainer: Propiedades que toman valores true o false dependiendo si el formulario es MDI hijo o Contenedor.

MDIChildren: Matriz de formularios hijo MDI que tiene un formulario dado.


ActiveMDIChild: Propiedad del formulario principal que hace referencia al formualrio MDI hijo activo en un momento dado. Si la aplicación no es MDI, existe la propiedad *ActiveForm*.

Ejemplo

En el ejemplo siguiente se obtiene una referencia al formulario MDI secundario activo y se pasa el texto del documento a color rojo tras la pulsación de una opción en un menú.

```
((Form2) this.ActiveMdiChild).textBox1.ForeColor = Color.Red;
```

MDIChildActivated: Evento que ocurre cuando un formulario hijo toma el foco.

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD	COMPETENCIA						

Apéndice V: Validación de datos

Eventos Validating/Validated

En ocasiones nos interesa que cuando un usuario escriba un dato, dicho dato sea de ciertas características. Puede darse el caso, por ejemplo, de que interese que un usuario introduzca un número en cierta *TextBox*. Vimos una manera de comprobar esto que era mediante el control de excepciones, pero veamos otras formas de validar datos.

Todos los componentes tienen la posibilidad de generar los eventos *validating* y *validate* en el momento que cogen el foco. Veamos como se gestiona esto:

Propiedad **CausesValidation**: Si está a *TRUE*, cuando el componente coge el foco genera el evento *Validating* sobre el **componente que tenía el foco previamente**.

Evento **Validating**: Se genera para la comprobación de alguna condición. Si no se cumple dicha condición se puede cancelar la validación y no permitir al usuario que abandone el componente. Esto se consigue poniendo el parámetro *e.Cancel=TRUE* de forma similar a como se hacía a la hora de cerrar o no un formulario en el evento *Closing*.

Evento **Validated**: Se produce si la validación ha tenido éxito.

El funcionamiento es el siguiente. Un usuario escribe cierto dato en un componente (por ejemplo una caja de texto que llamaremos *txtA*) y luego accede a otro componente mediante el ratón o la pulsación de TAB (por ejemplo a cierto botón que llamaremos *btnA*). Si dicho componente *btnA* tiene la propiedad *CausesValidation* a *TRUE*, entonces se ejecuta el código asociado al evento *Validating* de *txtA*.

Si en ese evento se pasa *e.Cancel* a *TRUE*, no se permitirá al usuario salir del componente *txtA*.

Ejemplo: Vamos a hacer un formulario en el que se validen los textbox si se desea aceptar pero no si se desea cancelar.

- Colocad en un form dos TextBox (con etiquetas de fecha y número >1900 y <2100 y dos botones (Aceptar y Cancelar).*
- Los CausesValidation de los textbox y del botón aceptar a TRUE. El del botón Cancelar a FALSE. El del formulario también debe estar a FALSE (para que se pueda cerrar con la X)*
- Escribir los siguientes fragmentos de código*

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

```
// Evento Validating textBox1
try {
    DateTime.Parse (textBox1.Text);
}
catch (Exception){
    e.Cancel = true;
    MessageBox.Show("El dato debe ser una fecha", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}

// Evento Validating textBox2
int a;
try {
    a=Convert.ToInt32(textBox2.Text);
    if (a > 2100 || a < 1900)
        throw new Exception();
}
catch (Exception) {
    e.Cancel = true;
    MessageBox.Show("El dato debe ser un número entre 1900 y 2100",
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}


// click de Cancelar
textBox1.CausesValidation=false;
textBox2.CausesValidation=false;
button1.CausesValidation=false;
this.Close();

/* Si se escribe
* Application.Exit();
* no son necesarios los
* CauseValidation=false; */
```

Hay que tener cuidado con los métodos ejecutados dentro de un componente al que se pueda llegar sin validación (por ejemplo un botón de ayuda o de cerrar) porque hay ciertos métodos (por ejemplo *Close*) que comprueban los componentes que requieren validación y comprueban dicha validación (Habría que colocar todos los *CausesValidation* a FALSE antes de hacer el *Close*).

ErrorProvider

Junto a los eventos de validación se puede utilizar un sistema de información muy similar al

	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD	COMPETENCIA						

que existe en páginas web que es la aparición de una marca junto a los campos que no son correctos. Esto evidentemente se puede hacer mediante algún icono y cierta cantidad de código.

Sin embargo existe el control no visual *ErrorProvider* que se maneja de forma similar al *ToolTip*. Una vez que se selecciona aparece en la parte inferior de la pantalla y en los *TextBox* aparece una nueva propiedad denominada *Error* in *ErrorProvider1*. Esta inicialmente está vacía. Si se pone algún texto, significa que en dicho campo ha habido error y por tanto se hará el icono visible.

En el programa anterior cambiar el evento de validación del primer *TextBox* tal y como sigue:

```
try {
    DateTime.Parse (textBox1.Text);
    this.errorProvider1.SetError(textBox1, "");
}
catch (Exception) {
    e.Cancel = true;
    this.errorProvider1.SetError(textBox1, "Introduzca fecha");
}
```

Esto activa y desactiva el icono según sea o no un dato incorrecto. Se usa habitualmente asociado a bases de datos y está más enfocado al uso en Web (ASP.Net) que al uso en aplicaciones.

MaskedTextBox

Como complemento a la validación existe el componente *MaskedTextBox* en el cual no vamos a profundizar pero debe conocerse por si interesa usarlo. Es similar al *TextBox* pero se le puede dar cierto formato.

En SharpDevelop ya viene integrada, en VS2003 se debe “personalizar barra de herramientas” y agregando el componente *Microsoft Masked Edit Control*. Aparecerá como *MaskedTextBox*.

Por ejemplo si en su propiedad *Mask* se coloca *##/##/##* la caja de texto se verá como

Para que el usuario sólo tenga que rellenar los valores de una fecha sin tener que preocuparse de los separadores. Esto es así porque # representa un número o espacio. Si por ejemplo queremos forzar a que sea sólo número, usamos 0 en lugar de #. Se pueden ver otros caracteres de máscara en el MSDN.

Si lo que se desea es cambiar el formato de salida para mostrar un dato en una etiqueta, se puede usar el método **Format** de la clase *string*. Ver su funcionamiento en el MSDN.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Apéndice VI: Drag and Drop básico

Veamos el funcionamiento básico de este evento que permite arrastrar y soltar elementos.

1. Coloca un ListBox con los días de la semana
2. Coloca un textbox vacío.
3. Pon el AllowDrop del TextBox a true para permitir la acción DragAndDrop. En algunos componentes (como el PictureBox) no aparece esta propiedad en el IDE, pero se puede poner a true en el Load del formulario).
4. En el listBox en el evento MouseDown ejecuta el siguiente método

```
DoDragDrop(this.listBox1.SelectedItem.ToString(), DragDropEffects.Copy);
```

Este método guarda en el MouseDown el dato que se quiere arrastrar a otro componente. El primer parámetro indica el dato que va a ser arrastrado y el segundo el efecto de arrastre (copiar, mover, etc...)

Si dependiendo del destino puede haber varios efectos, se puede usar un OR lógico. Por ejemplo:
DragDropEffects.Copy | DragDropEffects.Move

5. En el textbox en el evento DragEnter escribe la siguiente sentencia

```
e.Effect = DragDropEffects.Copy;
```

que indica el icono que se va a presentar si se arrastra algo a dicho componente (debe coincidir con el de la fuente, si no no funciona).

6. En el TextBox en el Evento DragDrop (fin del proceso) escribe el siguiente código:

```
this.textBox1.Text = e.Data.GetData(DataFormats.Text).ToString();
```

Mediante el cual obtenemos el dato de arrastre. Si se quiere comprobar que el tipo de dato que se arrastras es correcto se puede hacer una comprobación como la siguiente:

```
if (e.Data.GetDataPresent(DataFormats.Text))
    this.textBox1.Text = e.Data.GetData(DataFormats.Text).ToString();
```

Se pueden recoger elementos del Explorador de Windows mediante el DataFormat FileDrop, el cual devuelve un array de strings con los nombres de los archivos arrastrados.

En el caso de trabajar con DragDrop y ListBoxes hay que tener cuidado porque el método DoDragDrop anula el funcionamiento del evento selectedIndexChange en casos de selección múltiple debido a que no está del todo depurado el drag and drop en Windows Forms. Además debe usarse el MultiSimple, no el MultiExtended ya que da algunos problemas con los índices seleccionados.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

La solución pasa por buscar otro u otros eventos donde se pueda ejecutar lo que se desea hacer (por ejemplo el propio MouseDown).

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Desarrollo de Interfaces			CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:
	UNIDAD	COMPETENCIA			

Apéndice VII: ToolStrip


Este es un componente muy similar al MenuStrip, de hecho el menú hereda del ToolStrip. El ToolStrip es una banda de iconos que nos permite de forma rápida acceder en una aplicación a ciertos comandos.

Al añadir un toolstrip se permite meter una serie de componentes. Haz la prueba.

Luego esos componentes se pueden tratar como componentes sueltos o recorrer como si de una colección se tratase con un bucle del estilo de:

```
foreach (ToolStripItem c in toolStrip1.Items)
    ...
```

Puedes probar la siguiente secuencia como ejemplo de diseño:

1. Añade un *ToolStrip*
2. Añade elementos de trabajo estándar en el icono (Smart Icon )
3. En el smart icon pulsa en Editar Elementos
4. Añade 3 botones más (también lo puedes hacer directamente en el *ToolStrip*)
5. En las propiedades en imagen (o con el botón derecho en el propio botón) cámbiale la imagen a los dos primeros
6. En la propiedad *DisplayStyle* del segundo pon *ImageAndText* y *Text* en el tercero.
7. Cámbiale los campos *Text*.
8. Cambia también el *ToolTipText* al menos del primer botón.

Deben compartir método asociado al evento click con sus respectivas órdenes del *MenuStrip*.

Existen distintos componentes que se pueden añadir al *ToolStrip*.

Puedes ver un ejemplo de uso: <https://www.youtube.com/watch?v=BZdoXggYnSY>

En ocasiones puede ser interesante insertar los *ToolStrip* y *MenuStrip* en un *ToolStripContainer*, el cual gestiona estos elementos en cuatro paneles distribuidos en las cuatro paredes del formulario.

Existe la posibilidad de usar un *ToolStripContainer* que permite distribuir en zonas (arriba, abajo, izquierda y derecha), *ToolStrips*, Menús y *StatusBar*. Los *toolstrip* los pone en horizontal o vertical según su situación.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Desarrollo de Interfaces					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	UNIDAD	COMPETENCIA						

Apéndice VIII: Gestión de recursos y localización

Lo habitual en un proyecto es que dispongamos de una serie de recursos que se usan a lo largo de la aplicación. Estos puede ser cadenas, iconos, imágenes, audio, etc.

Esto es útil y no solo por tener todo gestionado de forma cómoda en un único sitio, si no que además es práctico para poder aplicar localización de forma que dependiendo del país o región dónde se ejecute, o por decisión del usuario, se muestren mensajes en distintos idiomas o se disponga de distinta iconografía.

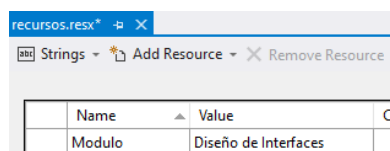
Vemos a continuación primero como gestionar recursos. Para ello en VS se accede a un archivo *resx* dentro de la carpeta Properties en el explorador de soluciones. Haz lo siguiente:

En el explorador de soluciones, botón derecho del ratón → Properties → Pestaña recursos y crea ahí el archivo. Si ya existiera simplemente te lleva ahí (Puede existir o no dependiendo de la plantilla: consola, WindowsForms, etc.).

También se puede hacer sobre el “directorio” Properties del explorador de soluciones y ahí añadir nuevo archivo de recursos.

Podemos cambiarle el nombre al archivo, por ejemplo **recursos.resx**.

En el editor de recursos se me permite por defecto añadir cadenas como recursos. Por ejemplo podemos crear una cadena denominada Modulo (sería la clave) que contenga el valor Diseño de Interfaces (no son necesarias las comillas).



Puedes añadir más cadenas u otros recursos en el desplegable Strings.


En el Programa luego se puede acceder al recurso mediante el espacio de nombres **Properties**. Por ejemplo:
`Console.WriteLine(Properties.recursos.Modulo);`

Lo mismo sería con imágenes, iconos o cualquier otro recurso. Incluso con imágenes tiene un editor a nivel de pixel.

Localización

En el caso de que se desee aplicar localización, debe haber un archivo por cada idioma. El estándar, no tiene nada añadido, el resto se pone antes de la extensión una clave que identifica la región.

Por ejemplo, crea otro archivo de recursos denominado *recursos.en.resx* y otro *recursos.gl.resx*.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Desarrollo de Interfaces				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	UNIDAD	COMPETENCIA				

En el primero mete la cadena Modulo como Interfaces Design, en el segundo Diseño de Interfaces.

Para que aparezca en inglés pondrías (Añade los using necesarios):

```
Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo("en");
Console.WriteLine(Properties.recursos.Modulo);
```

En gallego:

```
Thread.CurrentThread.CurrentUICulture = CultureInfo.GetCultureInfo("gl");
Console.WriteLine(Properties.recursos.Modulo);
```

De esta forma a partir de claves culturales se pueden gestionar distintos idiomas. Para saber la lista de claves de cada región puedes hacer lo siguiente:

```
CultureInfo[] cultures = CultureInfo.GetCultures(CultureTypes.AllCultures);
foreach (CultureInfo culture in cultures)
{
    Console.WriteLine($"{culture,15} - {culture.EnglishName}");
}
```

Si tienes muchas entradas en un archivo para no tener que escribirlas de nuevo puedes copiar el archivo en con la clave de otro idioma (por ejemplo recursos.fr.resx) y luego añades Item existente a Properties.

Para profundizar en el tema:

<https://docs.microsoft.com/es-es/dotnet/framework/resources/creating-resource-files-for-desktop-apps>

<https://docs.microsoft.com/es-es/dotnet/framework/resources/working-with-resx-files-programmatically>

Fuentes:

<https://stackoverflow.com/questions/90697/how-to-create-and-use-resources-in-net>

<https://stackoverflow.com/questions/1142802/how-to-use-localization-in-c-sharp>

<https://www.codeproject.com/Questions/407031/How-to-get-the-list-of-regions-using-system-global>