



Programación con C# .NET

Aplicaciones de Interfaz Gráfica

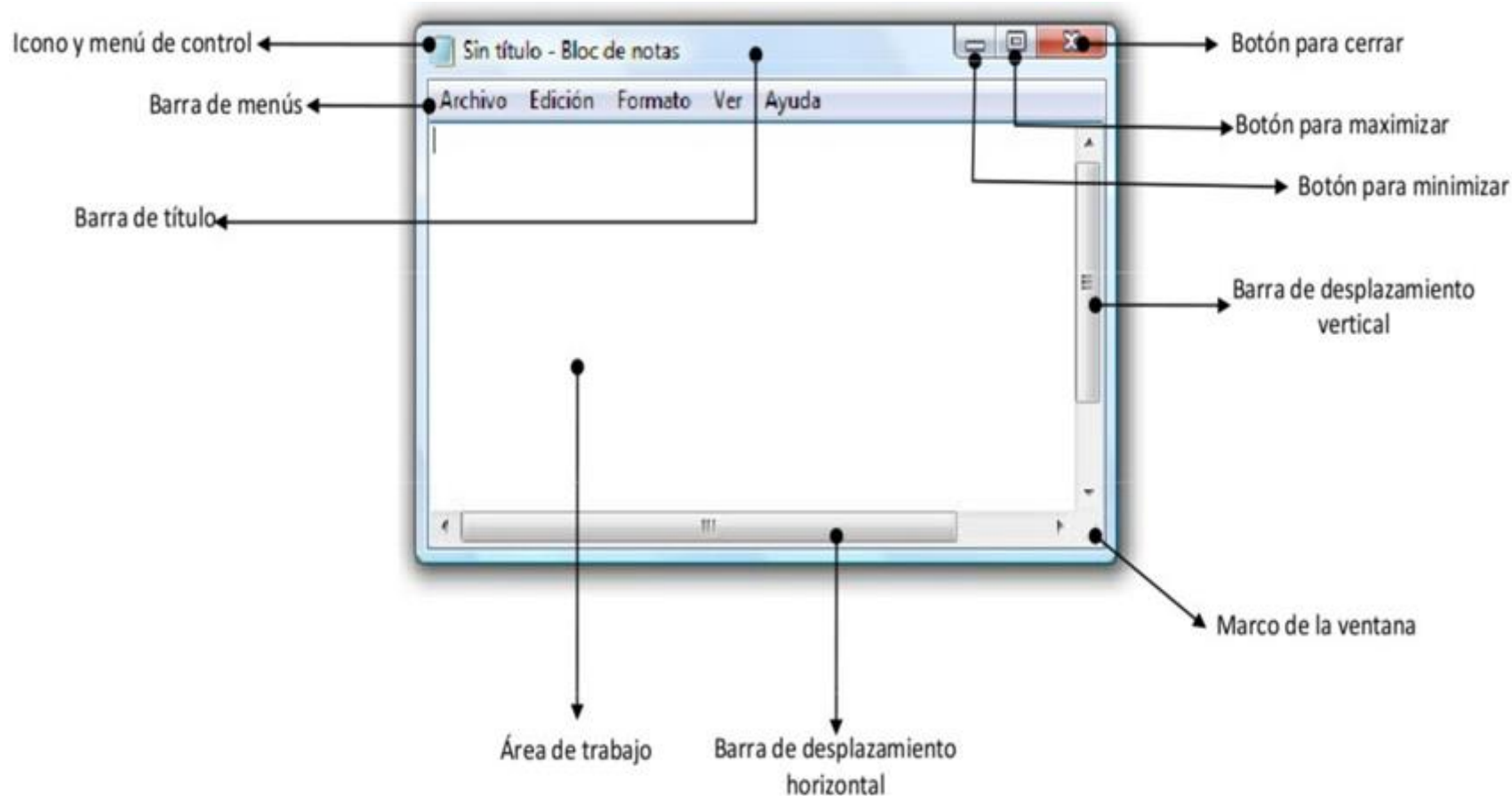


Contenido

- Introducción
- Programando en Windows
- Eventos
- Estructura de una aplicación
- Controles más comunes
- Manejo de eventos
- Eventos más comunes
- Asignar manejadores de eventos a un objeto
- Cajas de texto, etiquetas y botones
- Casilla de Verificación y botón de opción
- Listas simples y desplegables
- Botón por omisión y de cancelación
- Tecla de acceso
- Eventos asociados con el teclado
- Enfocar un objeto
- Seleccionar el texto de una caja de texto
- Métodos relacionados a la selección de texto
- Bibliografía

Introducción

- Una de las grandes ventajas de trabajar con aplicaciones gráficas es que todas las ventanas se comportan de la misma forma y todas las aplicaciones utilizan los mismos métodos básicos (menús desplegables, botones) para introducir órdenes.
- Una ventana típica tiene las siguientes partes:





Programando en Windows

- **Elementos que forman una aplicación gráfica**
 - Una interfaz gráfica tiene básicamente dos tipos de objetos:
 - Ventanas (también llamadas formularios)
 - Controles (botones, cajas de texto, menús, listas, etc.)
- **Funcionamiento de una aplicación gráfica**
 - Diseñada la interfaz gráfica, lo siguiente es escribir el código fuente relacionado con la función que tiene que realizar cada objeto de la interfaz
 - ♣ Este enfoque se dice que responde a un esquema de programación guiada por eventos y orientada a objetos



Programando en Windows

- Programar una aplicación gráfica implica escribir código separado para cada objeto; quedando la aplicación dividida en pequeños métodos conducidos por eventos

- Ejemplo:

```
btSaludo.Click += new System.EventHandler(btSaludo_Click); //manejador de evento clic
```

```
private void btSaludo_Click(object sender, EventArgs e)
{
    etSaludo.Text = "Hola Mundo!!!";
}
```

- De lo anterior podemos decir que:
 - El método *btSaludo_Click* será puesto en ejecución en respuesta al evento Click del objeto identificado por *btSaludo*; es decir, que cuando el usuario haga clic en el objeto *btSaludo* se ejecutará el método *btSaludo_Click*
 - Esto es justamente lo que está indicando el delegado *EventHandler*
 - El método *btSaludo_Click* manejará el evento *Click* del objeto *btSaludo*

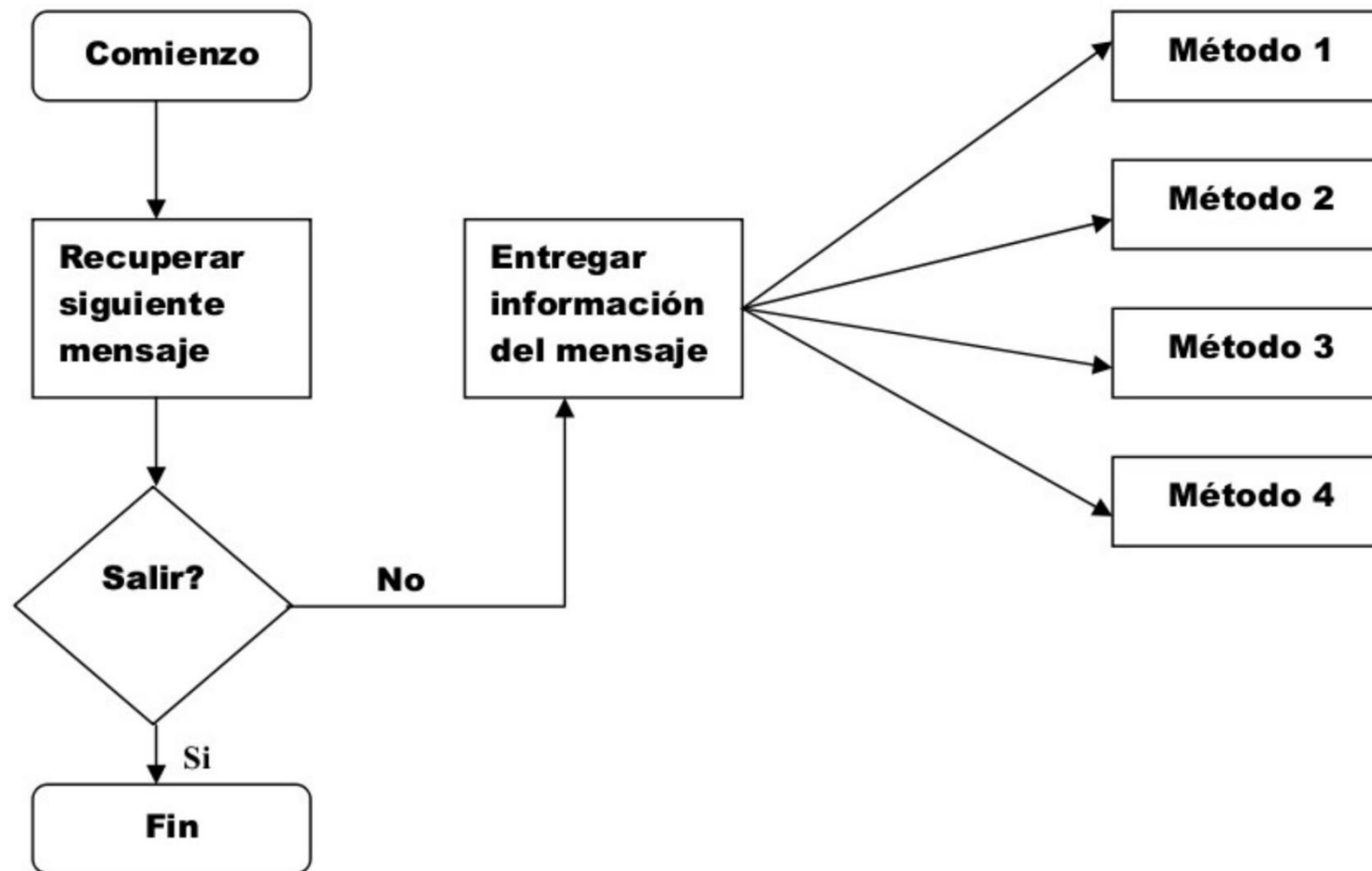


Eventos

- Son mecanismos mediante los cuales los objetos (ventanas o controles) pueden notificar de la ocurrencia de sucesos
- Pueden ser causados por:
 - Una acción del usuario. Ejemplo: Al pulsar una tecla
 - Por el sistema. Ejemplo: Al agotarse un temporizador
 - Indirectamente por el código. Ejemplo: Al cargar una ventana
- Dentro de una app. gráfica cada ventana y cada control pueden responder a un conjunto de eventos predefinidos
- Cuando ocurre uno de estos eventos, el sistema lo transforma en un mensaje que se coloca en la cola de mensajes de la aplicación implicada
- Un método **Run**, denominado bucle de mensajes, es el encargado de extraer los mensajes de la cola y despacharlos para que sean procesados
- Cada mensaje almacenará la información suficiente para identificar al objeto y ejecutar el método que tiene para responder a ese evento

Eventos

- A continuación se puede ver de forma gráfica como actúa el bucle de mensajes mientras la aplicación está en ejecución

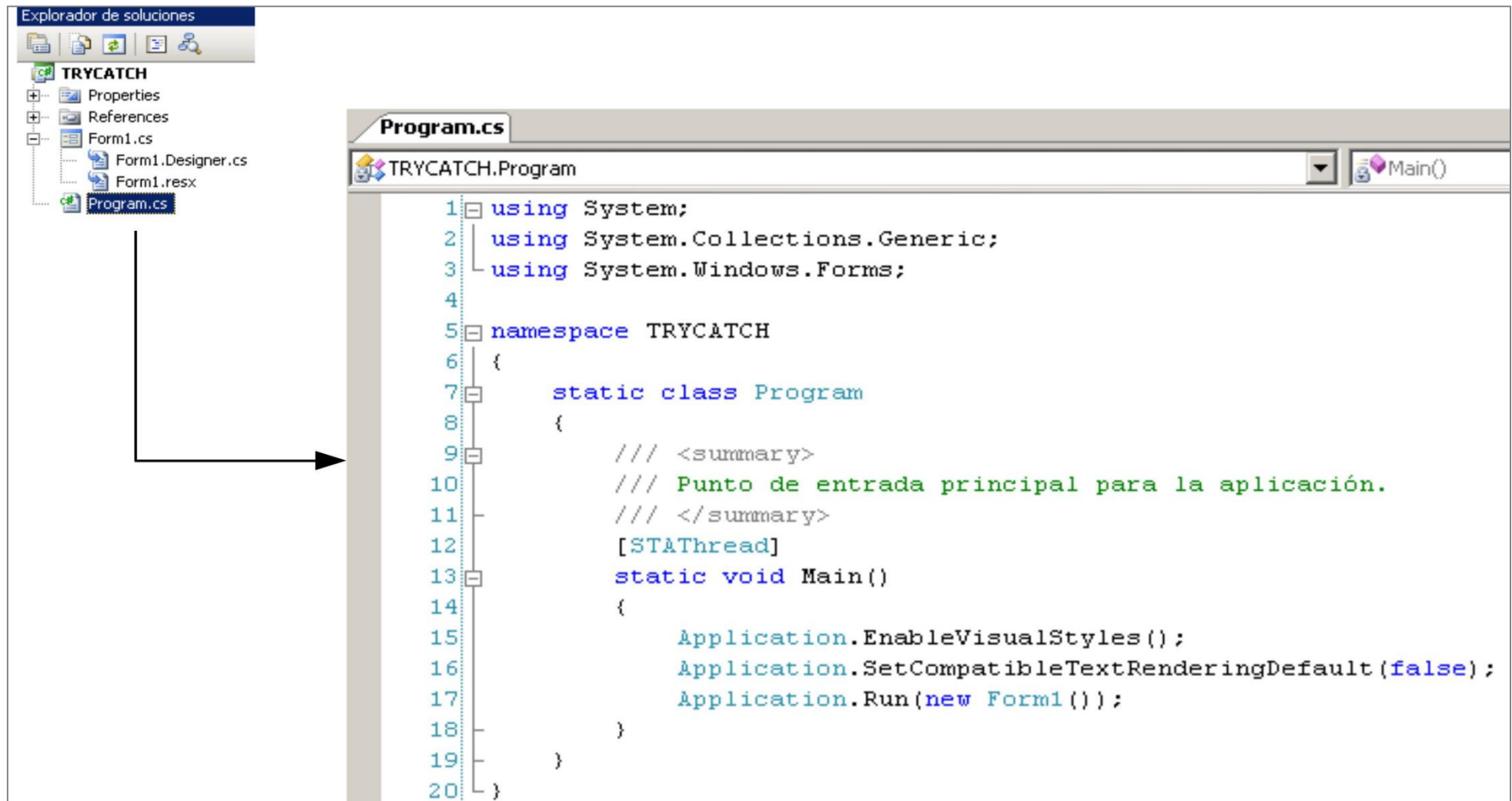




Estructura de una aplicación

- Una ventana o formulario no es más que un objeto de una clase derivada de **Form**
- Todo programa para poder ser ejecutado debe tener un método principal o método de entrada conocido como **main**
 - Este método es el punto de entrada a la aplicación
 - Al crear una aplicación C# con Visual Studio, el código generado aporta explícitamente el método **Main** en una clase llamada *Program* localizada en el fichero *Program.cs*

Estructura de una aplicación





Estructura de una aplicación

- Cuando se ejecuta el método **Main**:
 - Se invoca al constructor de la clase **Form1**
 - Esto crea la venta o formulario principal
 - El constructor de **Form1** llama al método *InitializeComponent*
 - Personaliza el tamaño, el nombre y el título del formulario principal
 - Personaliza, crea y asocia los controles al formulario principal
 - Finalizado el método *InitializeComponent*
 - La ventana principal esta construida
 - **Main** invoca al método **Run** de la clase *Program* para visualizar e iniciar el bucle de mensajes de la aplicación



Controles más comunes

- A continuación se muestra, de forma resumida, una lista de los controles más comunes:
 - *Etiquetas*. Se implementan a partir de la clase **Label**
 - *Botones*. Se implementan a partir de la clase **Button**
 - *Cajas de texto*. Se implementan a partir de la clase **TextBox** las de una sola línea de texto, las de varias líneas y las de "palabra de paso"
 - *Casillas de verificación*. Se implementan a partir de la clase **CheckBox**



Controles más comunes

- *Botones de opción.* Se implementan a partir de la clase **RadioButton** *Listas.*
- Se implementan a partir de las clases **ListBox** y **ComboBox**
- *Barras de desplazamiento.* Se implementan a partir de la clase **ScrollBar**
- Estos controles se localizan en el espacio de nombres *System.Windows.Forms* y son objetos de la subclase *Control* que a su vez se deriva de la clase *System.ComponentModel.Component*



Manejo de Eventos

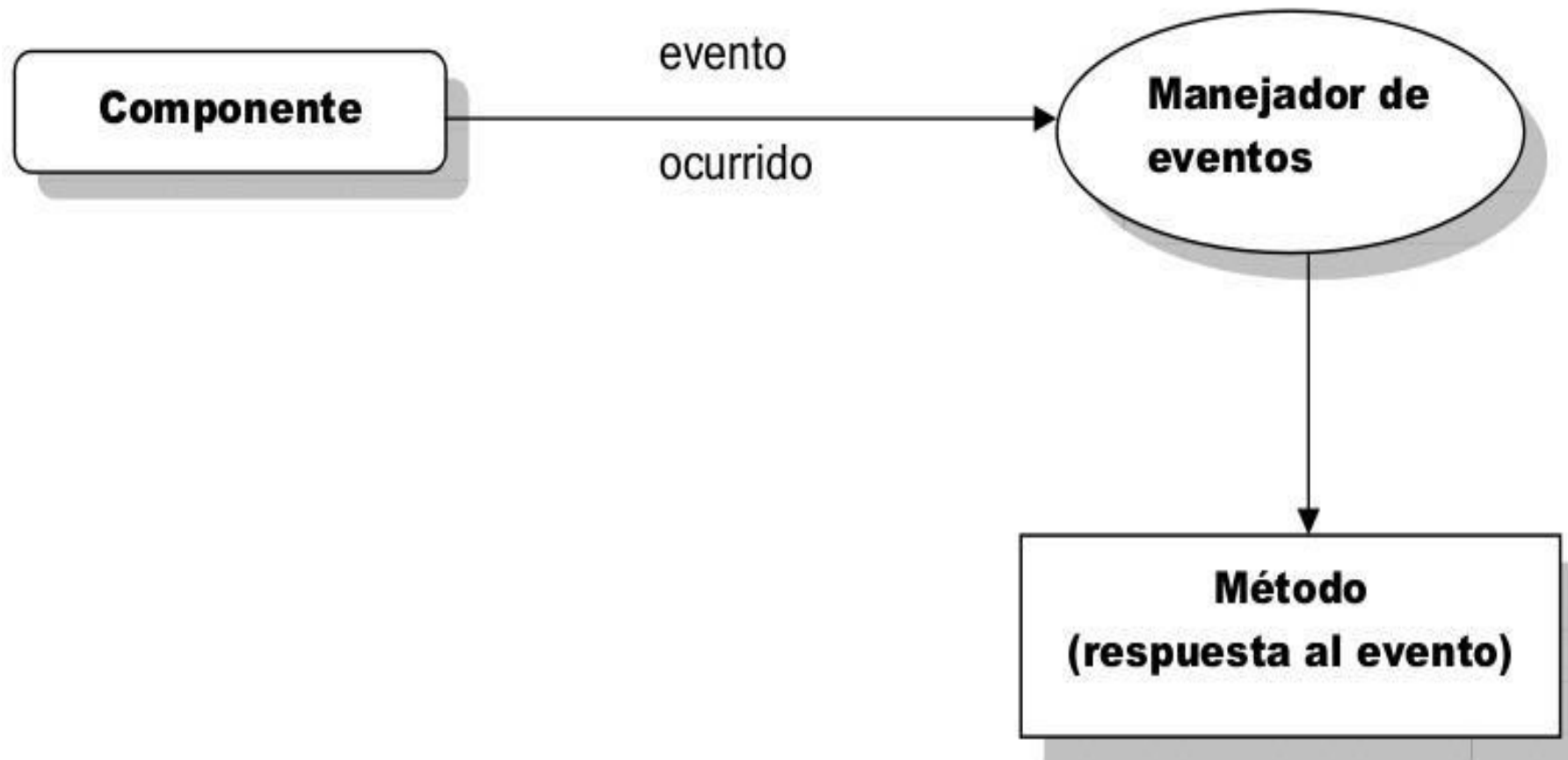
- Cuando una acción sobre un componente genera un evento, se espera que suceda algo, entendiendo por evento, un mensaje que un objeto envía a algún otro objeto
 - Esto quiere decir que hay un remitente del evento, por ejemplo, un botón, y hay un receptor del evento, por ejemplo, la ventana que contiene ese botón
- Lógicamente ese algo hay que programarlo y para ello hay que saber cómo manejar ese evento
 - Los eventos que se producen sobre un componente se manipulan a través de los manejadores de esos eventos
 - Un manejador de eventos es un objeto en el que un componente delega la tarea de manipular un tipo particular de eventos



Manejo de Eventos

- Cuando un componente genera un evento, un manejador de eventos vinculado con el componente se encarga de analizar qué evento ocurrió
 - Luego responde al evento ejecutando el método adecuado
- El manejador de eventos, recibe también el nombre de delegado y es un objeto de la clase **EventHandler**
- El delegado guarda una referencia al método que responderá al evento
 - Este método recibe el nombre de controlador de eventos

Manejo de Eventos





Eventos mas comunes

<i>Evento</i>	<i>Se produce cuando</i>
AutoSizeChanged	La propiedad AutoSize de un objeto cambia
BackColorChanged	El color de fondo de un objeto cambia
Click	Se hace clic sobre un objeto
ContextMenuStripChanged	El valor de la propiedad ContextMenuStrip de un objeto cambia
ControlAdded	Se añade un nuevo control a la colección ControlCollection
ControlRemoved	Se elimina un control de la colección ControlCollection
CursorChanged	El valor de la propiedad Cursor de un objeto cambia
DoubleClick	Se hace doble clic sobre un objeto
EnabledChanged	El valor de la propiedad Enabled de un objeto cambia
FontChanged	El valor de la propiedad Font de un objeto cambia
ForeColorChanged	El color del primer plano de un objeto cambia
Load	Se inicia la carga de un formulario por primera vez
Paint	El control se tiene que repintar
Resize	El objeto es redimensionado
SizeChanged	El valor de la propiedad Size cambia
TextChanged	El valor de la propiedad Text de un objeto cambia

Eventos mas comunes

<i>Del foco</i>	<i>(se exponen en el orden en que se producen)</i>
Enter	Se entra en el control
GotFocus	El control recibe el foco
Leave	Se sale del control
Validating	El control se está validando
Validated	El control está validado
LostFocus	El control pierde el foco
<i>Del teclado</i>	<i>(se exponen en el orden en que se producen)</i>
KeyDown	Se pulsa una tecla mientras el control tiene el foco
KeyPress	Una tecla está pulsada mientras el control tiene el foco
KeyUp	Una tecla es soltada mientras el control tiene el foco
<i>Del ratón</i>	<i>(se exponen en el orden en que se producen)</i>
MouseEnter	El puntero del ratón entra en un objeto
MouseMove	El puntero del ratón se mueve sobre un objeto
MouseHover	El puntero del ratón se sitúa encima del objeto
MouseDown	Se presiona un botón del ratón sobre el objeto
MouseWheel	La rueda del botón se mueve mientras el objeto tiene el foco
MouseUp	El puntero del ratón está encima del control y se suelta un botón del ratón
MouseLeave	El puntero del ratón deja el control



Asignar manejadores de eventos a un objeto

- Un objeto, generalmente un componente, puede tener asociados tantos manejadores de eventos como tipos de eventos tenga que manejar
- Ejemplo:

```
btSaludo.Click += new System.EventHandler(btSaludo_Click); private void  
btSaludo_Click(object sender, EventArgs e)  
{  
    etSaludo.Text = "Hola Mundo!!!";  
}
```
- Para indicar que el método *btSaludo_Click* es el manejador del evento *Click* se crea un delegado de tipo **EventHandler**, pasándole la dirección del método y añadiendo este delegado a la lista de métodos que serían llamados cuando el evento *Click* sea generado
 - Para añadir un delegado se utiliza el operador += y para quitarlo se utiliza el operador -=
- Un delegado es un objeto de una clase que puede contener una referencia a un método



Añadir una etiqueta y editar sus propiedades

- Primero hay que añadir a la clase del formulario una variable de tipo **Label**
 - `private Label etSaludo;`
- Luego, crear un objeto **Label** referenciado por *etSaludo*
 - `etSaludo = new Label();`
 - Esto generalmente esta dentro del método *InitializeComponent*
- Establecer las propiedades de la etiqueta
 - `etSaludo.Name = "etSaludo";`
 - `etSaludo.Text = "etiqueta";`
 - `etSaludo.Font = new Font("Microsoft Sans Serif", 14, FontStyle.Regular);`
 - `etSaludo.Location = new Point(53, 48);`
 - `etSaludo.Size = new Size(187, 35);`
- Asociar la etiqueta a la colección de controles del formulario
 - `Controls.Add(etSaludo);`



Añadir un botón de pulsación y editar sus propiedades

- Hay que añadir a la clase del formulario una variable de tipo **Button**
 - `private Button btSaludo;`
- Luego, crear un objeto **Button** referenciado por *btSaludo*
 - `btSaludo = new Button();`
 - Esto generalmente esta dentro del método *InitializeComponent*
- Establecer las propiedades del botón
 - `btSaludo.Name = "btSaludo";`
 - `btSaludo.Text = "Haga &clic aquí";`
 - `btSaludo.Location = new Point(53, 90);`
 - `btSaludo.Size = new Size(187, 23);`
- Asociar el botón a la colección de controles del formulario
 - `Controls.add(btSaludo);`



Añadir una caja de texto y editar sus propiedades

- Hay que añadir a la clase del formulario una variable de tipo **TextBox**
 - `private TextBox ctSaludo;`
- Luego, crear un objeto **TextBox** referenciado por *ctSaludo*
 - `ctSaludo = new TextBox();`
 - Esto generalmente esta dentro del método *InitializeComponent*
- Establecer las propiedades del botón
 - `ctSaludo.Name = "ctSaludo";`
 - `ctSaludo.Text = "";`
 - `ctSaludo.Location = new Point(53, 90);`
 - `ctSaludo.Size = new Size(187, 23);`
- Asociar el botón a la colección de controles del formulario
 - `Controls.add(ctSaludo);`



Añadir una casilla de verificación y editar sus propiedades

- Una casilla de verificación es un control que indica si una opción en particular está activada o desactivada. Cada casilla es independiente de las demás, ya que cada una de ellas tiene su propio identificador.
 - El número de opciones representadas de esta forma puede ser cualquiera, y de ellas el usuario puede seleccionar todas las que desee cada vez.
- La funcionalidad para manipular este tipo de controles es proporcionada por la clase **CheckBox**.
- Un clic sobre un objeto CheckBox genera un evento Click y cuando su estado cambia, un evento CheckedChanged.



Añadir una casilla de verificación y editar sus propiedades

```
partial class DlgCasillaVerificacion
{
    //...
    internal CheckBox cvConverMayus;

    cvConverMayus = new CheckBox();

    cvConverMayus.Name = "cvConverMayus";
    cvConverMayus.Text = "&Convertir a mayúsculas";
    cvConverMayus.AutoSize = true;
    cvConverMayus.Location = new System.Drawing.Point(29, 101);
    cvConverMayus.Size = new System.Drawing.Size(131, 17);
    cvConverMayus.TabIndex = 1;

    Controls.Add(cvConverMayus);

    //...
}
```



Añadir una casilla de verificación y editar sus propiedades

```
private void cvConverMayus_CheckedChanged (object sender, EventArgs e)
{
    string texto = ctTexto.Text;
    if (cvConverMayus.Checked)
        ctTexto.Text = texto.ToUpper();
    else
        ctTexto.Text = texto.ToLower();
    ctTexto.Focus();
    ctTexto.SelectionStart = ctTexto.Text.Length;
}
```




Añadir un botón de opción y editar sus propiedades

- Es un control que indica si una opción en particular está activada o desactivada. Cada botón de opción es independiente de los demás, ya que cada una de ellas tiene su propio identificador.
 - El número de opciones representadas de esta forma puede ser cualquiera, y de ellas el usuario solo puede seleccionar una cada vez.
- La funcionalidad para manipular este tipo de controles es proporcionada por la clase **RadioButton**.
- Un clic sobre un objeto RadioButton genera un evento Click y cuando su estado cambia, un evento CheckedChanged.



Añadir un botón de opción y editar sus propiedades

- Cuando el usuario selecciona un botón de opción en un grupo, los demás se desactivan automáticamente.
 - Para crear varios grupos en un mismo formulario, coloque cada grupo en su propio contenedor, por ejemplo en un control GroupBox o en un Panel.



Añadir un botón de opción y editar sus propiedades

```
partial class DlgBasesNum
{
    //...
    internal GroupBox gbGrupoBotones1;
    internal RadioButton;

    btopDecimal = new RadioButton();
    gbGrupoBotones1 = new GroupBox();

    btopDecimal.Name = "btopDecimal";
    btopDecimal.Text = "&Decimal":
    btopDecimal.AutoSize = true;
    btopDecimal.Checked = true;

    btopDecimal.Location = new System.Drawing.Point(6, 19);
    btopDecimal.Size = new System.Drawing.Size(59, 17);
    btopDecimal.TabIndex = 0;
```



Añadir un botón de opción y editar sus propiedades

```
private void btopDecOcHex_CheckedChanged(object sender, EventArgs e)
{
    if (btopDecimal.Checked)
        ctDato.Text = Convert.ToString(numeroActual, 10);
    else if (btopOctal.Checked)
        ctDato.Text = Convert.ToString(numeroActual, 8);
    else if (btopHex.Checked)
        ctDato.Text = Convert.ToString(numeroActual, 16).ToUpper();
    ctDato.Focus();
}
```



Añadir una lista simple y editar sus propiedades

- Una lista es un control de la clase **ListBox** que pone a disposición del usuario un conjunto de elementos, que podrá elegir haciendo clic sobre ellos.
- De forma predeterminada, los elementos de una lista son visualizados verticalmente en una sola columna, aunque se puede establecer múltiples columnas estableciendo la propiedad `MultiColumn` a valor `True`.



Añadir una lista simple y editar sus propiedades

- Las propiedades **Items**, **SelectedItems** y **SelectedIndices** proporcionan acceso a las tres colecciones que utiliza un control **ListBox**. La primera contiene todos los elementos de la lista, la segunda contiene los elementos seleccionados y la tercera los índices de los elementos seleccionados.
 - Para añadir elementos a la colección **Items** se puede usar el método **AddRange** o **Add**.
- La propiedad **SelectedIndex** permite acceder al índice del elemento seleccionado en la lista y la propiedad **SelectedItem** es una referencia a este elemento.



Añadir una lista simple y editar sus propiedades

- Cuando seleccionamos un elemento de la lista, se modifica su propiedad **SelectedIndex** para almacenar el índice del nuevo elemento seleccionado y esto hace que se produzca el evento **SelectedIndexChanged**.
- Cuando no hay ningún elemento seleccionado, la propiedad **SelectedIndex** vale **-1**, y cuando hay varios elementos seleccionados almacena el índice del primer elemento de la selección.



Añadir una lista simple y editar sus propiedades

```
partial class DlgListaSimple
{
    //...
    internal ListBox lsListal;

    lsListal = new ListBox();

    lsListal.Name = "lsListal";
    lsListal.Location = new System.Drawing.Point(12, 12);
    lsListal.Size = new System.Drawing.Size(180, 147);
    lsListal.TabIndex = 0;

    Controls.Add(lsListal);
    //...
}
```




Añadir una lista simple y editar sus propiedades

- Iniciar la lista:

```
private void DlgListaSimple_Load(object sender, EventArgs e)
{
    string [] elemento = new string [] {
        "uno", "dos", "tres", "cuatro", "cinco", "seis", "siete",
        "ocho", "nueve", "diez", "once", "doce", "trece", "catorce" };

    this.lsLista1.Items.AddRange(elemento);
}
```



Añadir una lista simple y editar sus propiedades

- Acceder a los elementos seleccionados:

```
private void btAceptar_Click (object sender, EventArgs e)
{
    if (lsLista1.SelectedIndex < 0) return;
    object elementoSeleccionado;
    elementoSeleccionado = lsLista1.SelectedItem;
    MessageBox.Show(elementoSeleccionado.ToString());
}
```



Añadir una lista simple y editar sus propiedades

- Añadir y borrar elementos de la lista:

```
private void btAñadir_Click(object sender, EventArgs e)
{
    // Añadir el elemento introducido en la
    // caja de texto ctAñadir
    if (ctAñadir.Text.Length != 0)
        lsListal.Items.Add(ctAñadir.Text);
}
```

```
private void btBorrar_Click(object sender, EventArgs e)
{
    if (lsListal.SelectedIndex < 0)
        lsListal.Items.RemoveAt(lsListal.SelectedIndex);
}
```



Añadir una lista desplegable y editar sus propiedades

- Una lista desplegable es un objeto de la clase **ComboBox**. La diferencia entre una lista simple y una lista desplegable es que esta última es una combinación de una lista simple y una caja de texto, y que permite la selección de un solo elemento cada vez.
- Hay tres estilos diferentes de listas desplegables: ***editable, no editable y estática***, que pueden ser fijados mediante su propiedad `DropDownStyle` (`DropDown`, `DropDownList`, `Simple`).



Añadir una lista desplegable y editar sus propiedades

```
partial class DlgListaDesplegable
{
    //...
    internal ComboBox ldListal;

    ldListal = new ComboBox();

    ldListal.Name = "ldListal";
    ldListal.Location = new System.Drawing.Point(13, 13);
    ldListal.Size = new System.Drawing.Size(169, 21);
    ldListal.TabIndex = 0;

    Controls.Add(ldListal);
    //...
}
```



Añadir una lista desplegable y editar sus propiedades

- Acceder a los elementos seleccionados:

```
private void btAceptar_Click (object sender, EventArgs e)
{
    if (ldLista1.SelectedIndex < 0) return;
    object elementoSeleccionado;
    elementoSeleccionado = lsLista1.SelectedItem;
    MessageBox.Show(elementoSeleccionado.ToString());
}
```



Añadir una lista desplegable y editar sus propiedades

- Añadir y borrar elementos de la lista:

```
private void btAñadir_Click(object sender, EventArgs e)
{
    // Añadir el elemento introducido en la
    // caja de texto ctAñadir
    if (ctAñadir.Text.Length != 0)
        ldLista1.Items.Add(ctAñadir.Text);
}

private void btBorrar_Click(object sender, EventArgs e)
{
    if (ldLista1.SelectedIndex < 0)
        ldLista1.Items.RemoveAt(ldLista1.SelectedIndex);
}
```



Botón por omisión y de cancelación

- Si una ventana tiene uno o más botones, uno y sólo uno de ellos puede ser el botón por omisión, lo que implica que la tecla *Enter* realice la misma función
- Para hacer que un botón sea el botón predeterminado de un formulario, hay que asignar a la propiedad *AcceptButton* del formulario el nombre de ese botón
 - El botón por omisión se distingue de los demás porque aparece rodeado con un borde más oscuro
- Para hacer que un botón sea el botón de cancelación predeterminado de un formulario, hay que asignar a la propiedad *CancelButton* del formulario el nombre de ese botón
 - El botón de cancelación predeterminado se distingue de los demás porque al pulsar la tecla *Esc* ejecuta su código

Botón por omisión y de cancelación

The screenshot shows a Visual Studio IDE with a Windows Form named 'Form1'. The form contains a text box with the placeholder text 'Ingrese un número:', a 'Calcular' button, and a 'Salir' button. The Properties window on the right shows the 'Form1' properties. The 'AcceptButton' property is set to 'btnCalcular' and the 'CancelButton' property is set to 'btnSalir'. Arrows indicate the association between the buttons and their respective properties.

Asociado al pulsar Enter sobre el formulario

Asociado al pulsar Esc sobre el formulario

Form1 System.Windows.Forms.Form	
(ApplicationSettings)	
(DataBindings)	
(Name)	Form1
AcceptButton	btnCalcular
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AutoScaleMode	Font
AutoScroll	False
AutoScrollMargin	0; 0
AutoScrollMinSize	0; 0
AutoSize	False
AutoSizeMode	GrowOnly
AutoValidate	EnablePreventFocusChange
BackColor	Control
BackgroundImage	(ninguno)
BackgroundImageLayout	Tile
CancelButton	btnSalir
CausesValidation	True
ContextMenuStrip	(ninguno)
ControlBox	True
Cursor	Default
DoubleBuffered	False
Enabled	True
Font	Microsoft Sans Serif; 8,25pt
ForeColor	ControlText
FormBorderStyle	Sizable
HelpButton	False

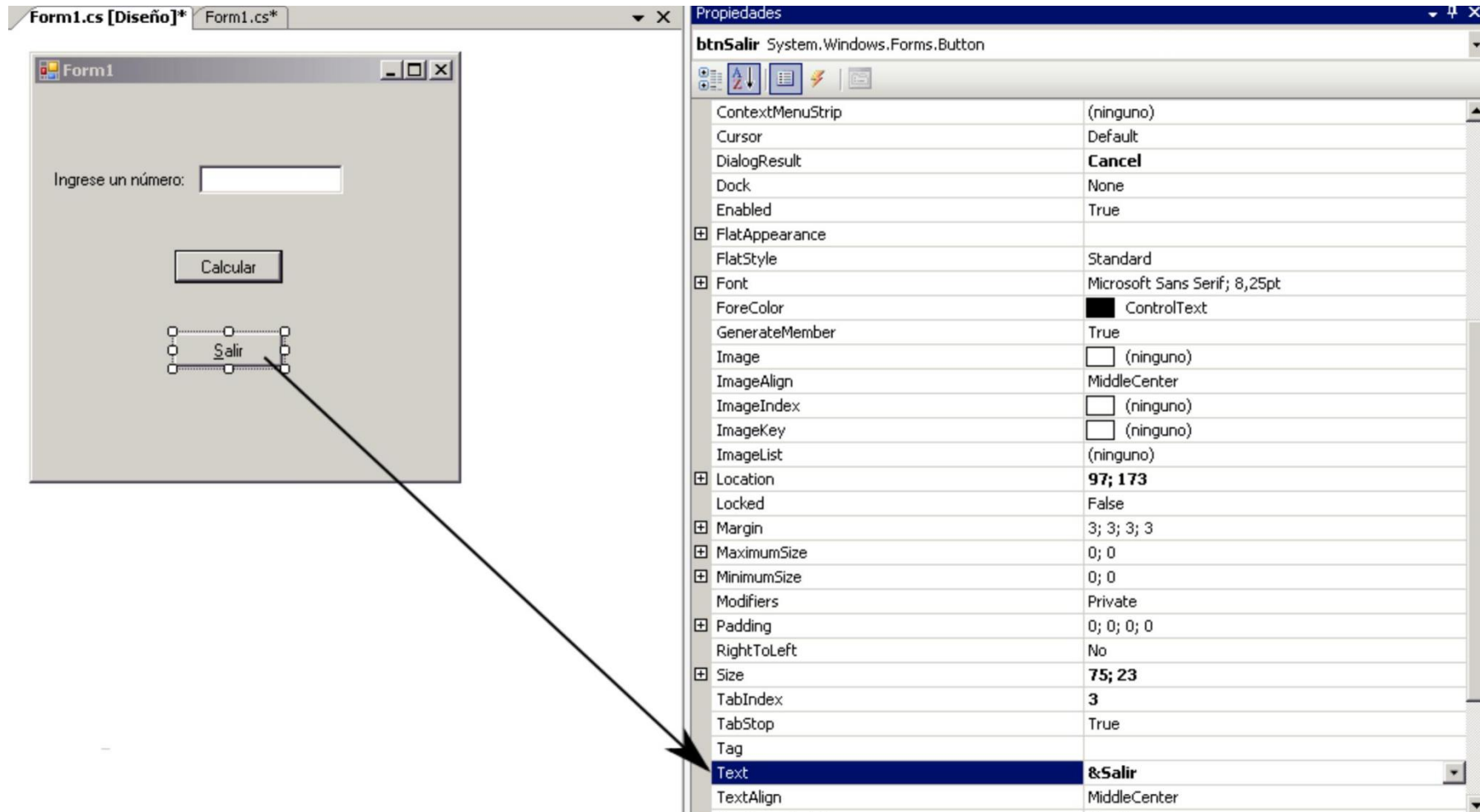
AcceptButton
Botón Aceptar del formulario. Si está establecido, el botón se 'activa' cuando el usuario presiona la tecla 'ENTR...



Tecla de acceso

- En la siguiente figura, se puede observar en el título del botón Aceptar, que la letra A aparece subrayada
 - Esto significa que el usuario podrá también ejecutar la acción especificada por el botón, pulsando las teclas *Alt* + *A*
 - Esta asociación tecla-control recibe el calificativo de nemónico y se realiza escribiendo el símbolo ampersand (&) antes de la letra que desea dé acceso al botón
 - Si no se ve la letra subrayada, es necesario pulsar la tecla *Alt*

Tecla de acceso



The screenshot displays a Visual Studio IDE window titled 'Form1.cs [Diseño]*'. The design view shows a form with a text box labeled 'Ingrese un número:', a 'Calcular' button, and a 'Salir' button. The 'Salir' button is selected, and an arrow points from it to the 'Text' property in the Properties window. The Properties window shows the 'btnSalir' control with various properties. The 'Text' property is highlighted and set to '&Salir', indicating the mnemonic character for the button.

Property	Value
ContextMenuStrip	(ninguno)
Cursor	Default
DialogResult	Cancel
Dock	None
Enabled	True
FlatAppearance	
FlatStyle	Standard
Font	Microsoft Sans Serif; 8,25pt
ForeColor	ControlText
GenerateMember	True
Image	(ninguno)
ImageAlign	MiddleCenter
ImageIndex	(ninguno)
ImageKey	(ninguno)
ImageList	(ninguno)
Location	97; 173
Locked	False
Margin	3; 3; 3; 3
MaximumSize	0; 0
MinimumSize	0; 0
Modifiers	Private
Padding	0; 0; 0; 0
RightToLeft	No
Size	75; 23
TabIndex	3
TabStop	True
Tag	
Text	&Salir
TextAlign	MiddleCenter



Eventos asociados al teclado

- Cuando el usuario escribe sobre un control, por ejemplo sobre una caja de texto, cada pulsación produce los eventos:
 - **KeyDown:** Se genera cuando se pulsa la tecla
 - **KeyPress:**
 - Se genera cuando se va a escribir el carácter
 - Se genera sólo cuando se introducen caracteres ASCII
 - Se excluyen teclas como: funciones (F1 a F12), cursores (←↑↓→), especiales (Del)
 - **KeyUp:** Se genera cuando se suelta la tecla



Eventos asociados al teclado

```
private void ctGradosC_KeyPress(object sender, KeyPressEventArgs e)
{
    //...
}
```

- El controlador anterior está asociado a una caja de texto cuyo nombre es *ctGradosC* y ejecutará el código cuando se produzca el evento *KeyPress* sobre dicha caja
- El controlador recibe un primer parámetro, *sender*, de tipo *object*
 - Hace referencia al objeto para el cual ha sido invocado
- Y un segundo parámetro, *e*, de tipo *KeyPressEventArgs*
 - El cual proporciona las siguientes propiedades:



Eventos asociados al teclado

- **Handled.** Propiedad de tipo bool, permite obtener o establecer si se controló (true) o no (false) el evento *KeyPress*
 - Si el evento no está controlado, se enviará el carácter tecleado al sistema operativo para que él realice el procesamiento predeterminado
- **KeyChar.** Propiedad de tipo char, permite obtener, y modificar si fuera necesario, el carácter correspondiente a la tecla pulsada



Eventos asociados al teclado

- Una forma de interceptar la tecla *Enter* es asociando un manejador de eventos *KeyPress* con el componente de texto que recibe tal evento, y verificar si se pulsó la tecla *Enter*

```
private void CajaTexto_KeyPress(object sender, KeyPressEventArgs e) {  
    if (e.KeyChar == Convert.ToChar(13)) //Se pulso la tecla Enter {  
        e.Handled = true;  
        Conversion(sender);  
    }  
}
```

- La propiedad *KeyChar* se utiliza para comprobar si se presionó la tecla *Enter*
- Si se presionó, la propiedad *Handled* se establece en *true*, lo que indica que será nuestra aplicación la que controlará el evento, no el sistema operativo



Eventos asociados al teclado

- Validación de un campo de texto (**Tarea**):
 - El contenido de la caja de texto será válido cuando sus caracteres pertenezcan al siguiente conjunto: + - ,1234567890
 - El signo + o – sólo puede aparecer al principio del dato y éste sólo puede contener una coma decimal



Enfocar un objeto

- Cuando un control posee el punto de inserción, se dice que dicho control está enfocado o que tiene el foco
- Un usuario de una aplicación puede enfocar un determinado control:
 - Haciendo clic sobre él
 - O pulsando la tecla *Tab* una o más veces hasta situar el foco sobre él
- Asimismo, un control también puede ser enfocado desde el código de la propia aplicación y lo podemos hacer de la siguiente manera:
 - Invocando al método *Focus* para el control que requiere el foco una vez abierto el formulario
 - Ejemplo:

```
private void Form1_Load(object sender, EventArgs e)
{
    ctGradosC.Focus();
}
```



Seleccionar el texto de una caja de texto

- En la mayoría de las veces es necesario que cuando una caja de texto obtenga el foco, todo su contenido quede seleccionado
- Esto es fácil si sabemos que cuando un control obtiene el foco, genera el evento “foco obtenido” (Enter o GotFocus) y cuando lo pierde, “foco perdido” (Leave o LostFocus)

- Ejemplo:

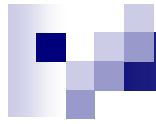
```
private void CajaTexto_Enter(object sender, EventArgs e)
{
    TextBox ObjTextBox = (TextBox) sender;
    ObjTextBox.SelectAll();
}
```

- El anterior es un método que contiene un código genérico sin importar el nombre de la caja de texto sobre la cual se ha ganado el foco



Métodos relacionados a la selección de texto

- Otras propiedades/métodos relacionadas con la selección de texto en un control de texto son las siguientes:
 - **SelectionStart.** Propiedad que permite obtener o establecer el punto de inicio del texto seleccionado en la caja de texto
TextBox1.SelectionStart = 10;
pos = TextBox1.SelectionStart;
 - **SelectionLength.** Propiedad que permite obtener o establecer el número de caracteres seleccionados en la caja de texto
TextBox1.SelectionLength = 5;
n = TextBox1.SelectionLength;
 - **SelectedText.** Propiedad que permite obtener el texto seleccionado, o bien reemplazar el texto seleccionado (puede ser nulo) por otro
TextBox1.SelectionStart = TextBox1.Text.Length;
TextBox1.SelectedText = "NuevoTexto";
 - **Select (int pos_inicial, int pos_final).** Selecciona el texto que se encuentra entre las posiciones especificadas



Bibliografía

Enciclopedia de Microsoft Visual C#, 4a edición

Fco. Javier Ceballos Sierra

RA-MA