UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL GENERAL PACHECO

TÉCNICO SUPERIOR EN PROGRAMACIÓN

PROGRAMACIÓN III

APUNTE TEORICO

PARTE 1

APLICACIÓN WINDOWS I

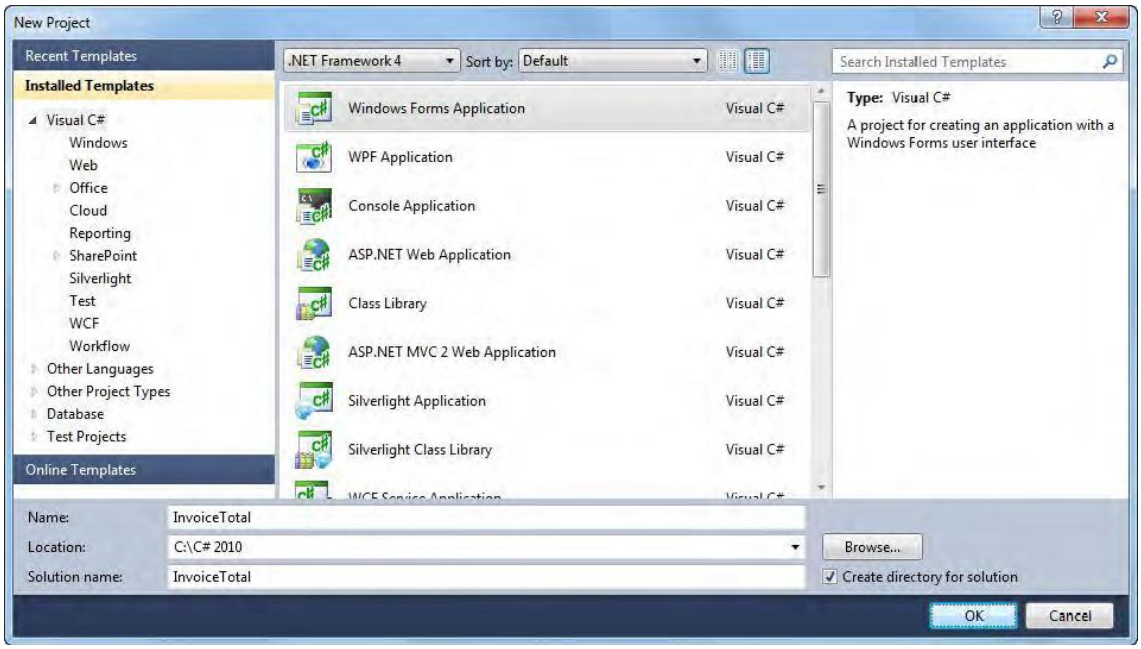
CREAR UN NUEVO PROYECTO

1. Vamos Archivo > Nuevo > Proyecto esto abrirá un Dialog Box para seleccionar el tipo de Proyecto Nuevo.

2. En la parte izquierda tenemos la ventana de los templates, seleccionamos Visual C# y luego en la categoría Windows. En el centro aparecerán los templates disponibles, seleccionamos Windows Forms Application.

3. Ahora en la parte inferior colocamos un nombre al proyecto, seleccionamos la ubicación y el nombre que le daremos a la solución.

4. Por ultimo pulsamos OK y se creara el Proyecto.



Aclaración:

Al seleccionar un proyecto desde un témplate se agregan archivos, código y se realiza una configuración inicial del proyecto.

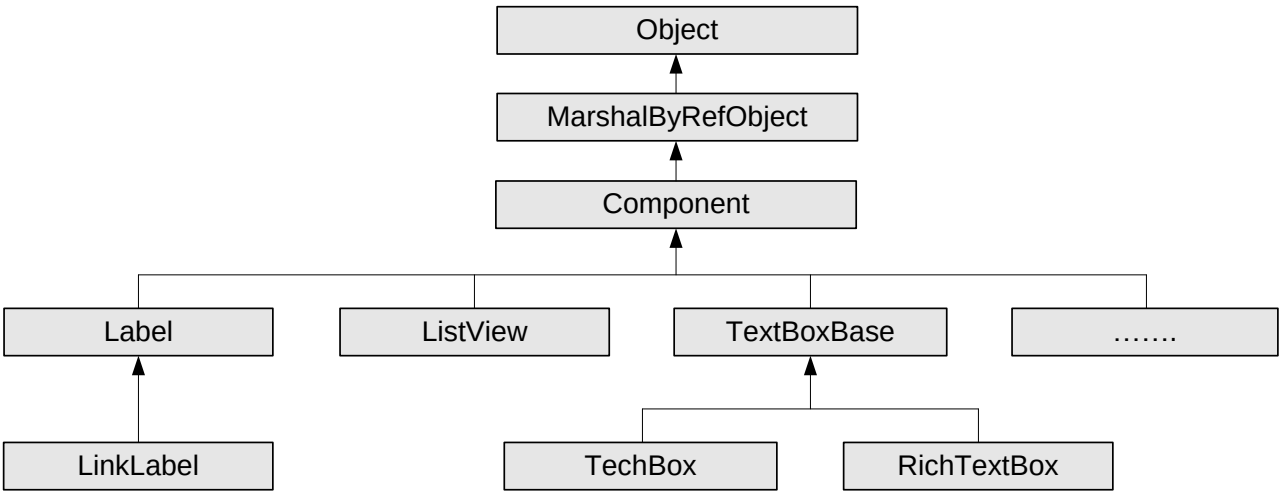
Si esta tildada la opción de crear directorio para la solución, primero crea un directorio para la solución y luego un subdirectorio para el proyecto.

También en la parte superior del Dialog Box puedes seleccionar la versión del .NET Framework con la cual trabajar.

CONTROLES

Cuando se trabaja con Aplicaciones Windows Forms, se esta trabajando con el espacio de nombre(namespace) System.Windows.Forms, este espacio de nombre esta incluido en los archivos Form de nuestra aplicación con la sentencia using.

Muchos de los controles que se utilizaran derivan de la clase System.Windows.Forms.Control, en esta clase se definen la funcionalidad básica de los controles por eso algunas de las propiedades y eventos son Iguales. Y algunas de estas clases son la base para otros controles.



PROPIEDADES

Todos los Controles tienen una cantidad de propiedades que sirven para manejar el comportamiento de los controles. La clase base para la mayoría de los controles es System.Windows.Forms.Control.

Las propiedades más comunes de estos controles son:

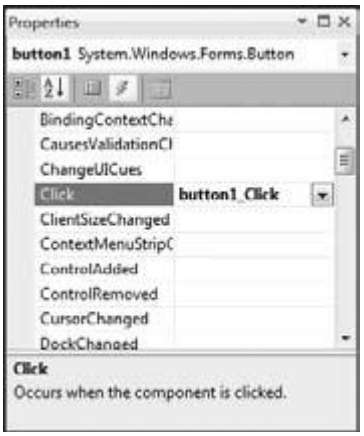
NOMBRE	DISPONIBILIDAD	DESCRIPCIÓN
Anchor	*Lectura/Escritura*	*Especifica como se comporta el control cuando el recipiente cambia de tamaño.*
BackColor	*Lectura/Escritura*	*Color de fondo del control*
Bottom	*Lectura/Escritura*	*Se especifica la distancia desde la parte superior de la ventana a la parte inferior del control.*
Dock	*Lectura/Escritura*	*Allows you to make a control dock to the edges of a window. See below for a more detailed explanation of this property.*
Enabled	*Lectura/Escritura*	*Habilita o Deshabilita el Control.*
ForeColor	*Lectura/Escritura*	*El color de primer plano del control.*
Height	*Lectura/Escritura*	*La altura del control.*

Left	*Lectura/Escritura*	*La distancia del borde izq. Del control al Borde izq. De la ventana.*
Name	*Lectura/Escritura*	*El nombre del control.*
Parent	*Lectura/Escritura*	*El padre del control.*
Right	*Lectura/Escritura*	*Distancia del borde derecho del control al borde izq. De la ventana.*
TabIndex	*Lectura/Escritura*	*El numero de orden del control en el Form (Recipiente).*
TabStop	*Lectura/Escritura*	*Especifica si el control puede ser accedido mediante la tecla Tab..*
Tag	*Lectura/Escritura*	*Este valor no es usado por el control y se utiliza para guarder información sobre el control en sí mismo.Cuando esta propiedad es cambiada mediante el diseñador de Formularios solo se le puede asignar un valor de tipo string.*
Top	*Lectura/Escritura*	*Distancia del borde sup. Del control con respecto al bor-de sup. De la ventana.*
Visible	*Lectura/Escritura*	*Si es visible o no.*
Width	*Lectura/Escritura*	*Largo del control.*

EVENTOS

Los eventos son generados por los controles, generalmente esto pasa cuando el usuario realiza alguna acción sobre el control, por Ej. Un click sobre un botón. La clase System.Windows.Forms.Control define varios eventos comunes para la mayoría de los controles.

Para acceder a los eventos existen varias formas, una de ellas es hacer doble clic sobre el control, lo cual lo llevara al evento predeterminado del control. Otra forma es en la ventana de propiedades hacer clic en el icono de eventos y luego seleccionar el evento deseado.



Otra forma es directamente desde la ventana del código del Formulario.

La lista de algunos de los eventos comunes es:

NOMBRE	DESCRIPCIÓN
Click	Ocurre cuando se hace click en un control. En algunos casos este evento ocurre cuando el usuario aprieta Enter
DoubleClick	Ocurre cuando se hace doble click en un control. Cuando se maneja el evento Click en algunos controles como el Boton entonces el evento DoubleClick no llega ser llamado.
DragDrop	Ocurre cuando se completa una operación de drag y drop, es decir cuando un objeto es arrastrado y luego el usuario libera el botón del mouse
DragEnter	Ocurre cuando un objeto es arrastrado y se ingresa en los limites de otro control.
DragLeave	Ocurre cuando un objeto es arrastrado y se sale de los limites de otro control.
DragOver	Ocurre cuando un objeto fue arrastrado por encima de otro control.
KeyDown	Ocurre cuando se presiona una tecla mientras el control tiene el foco. Este evento ocurre siempre antes de los eventos KeyPress y KeyUp.
Name	Descripción del objeto
KeyPress	Ocurre cuando se presiona una tecla mientras el control tiene el foco. Este evento ocurre siempre después del evento KeyDown y antes del evento KeyUp. KeyDown pasa el código de la tecla que fue presionada mientras que KeyPress el valor de carácter de la tecla que fue presionada.
KeyUp	Ocure cuando una tecla es soltada mientras un control tiene el foco. Este evento ocurre siempre despues de los eventos KeyDown y KeyPress.
GotFocus	Ocure cuando un control recibe el foco. No utilice este evento para efectuar validaciones. Para ello se utilizan los eventos Validating y Validated.
LostFocus	Ocurre cuando un control pierde el foco. No utilice este evento para efectuar validaciones. Para ello se utilizan los eventos Validating y Validated.
MouseDown	Ocure cuando el puntero del mouse se ubica sobre un control y un boton del mouse es presionado. Esto no es lo mismo que el evento Clikc porque el evento MouseDown ocurre apenas el boton es presionado y antes de que sea liberado.
MouseMove	Ocure permanentemente cuando el mouse se mueve sobre un control.
MouseUp	Ocure cuando el puntero del mouse se ubica sobre un control y un boton del mouse es liberado.
Paint	Ocurres cuando dibujamos un control.
Validated	Este evento se dispara cuando un control con la propiedad CausesValidation en valor true esta por recibir el foco. Se dispara después de que el evento Validating finaliza e indica indica que la validación fue completada.
Validating	Este evento se dispara cuando un control con la propiedad CausesValidation en valor true esta por recibir el foco. El control que esta por ser validado es el control que pierde el foco, no el que lo esta recibiendo.

PRACTICA 1 (FORM)

1. Creamos un Proyecto nuevo con el nombre MiPrimerAplicacion.
2. Analizamos los archivos que nos creo el Proyecto.
3. Cambiamos el nombre del Form1, con la propiedad Name.
4. Cambiamos el titulo del Formulario, con la propiedad Text. (Run)
5. Cambiamos el color de fondo del Formulario, con la propiedad BackColor. (Run)
6. Cambiamos la posición del Formulario, con la propiedad StartPosition. (Run)
7. Cambiamos la vista inicial del Formulario (Normal, Minimizado o Maximizado), con la propiedad WindowState. ()
8. Manejar el evento Load del Formulario.
    1. DobleClick sobre el Formulario carga el método del evento Load por defecto.

```
private void Form1_Load(object sender, EventArgs e)
{

}
```

    2. Mostramos un Mensaje con MessageBox.Show. (Run)

```
private void Form1_Load(object sender, EventArgs e)
{
        MessageBox.Show("Bienvenidos a C#");
}
```

9. Manejar el evento FormClosed del Formulario.
    1. Mostramos un Mensaje con MessageBox.Show. (Run)

```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
        MessageBox.Show("Chau Chau ...");
}
```

10. No mostrar el botón de minimizar .del Formulario.
11. No mostrar el botón de maximizar del Formulario.
12. Modificar el valor de la propiedad Opacity.

EL CONTROL BUTTON

El Control Button deriva de la clase System.Windows.Forms.ButtonBase que brinda una funcionalidad básica, además el usuario puede usar esta clase para crear sus propios buttons personalizados.

De esta clase básica derivan otros 3 controles, Button, CheckBox, RadioButton.
Normalmente los Buttons se usan para aceptar o cancelar un DialogBox, llamar a otro formulario o aplicación y dar acción a los datos introducidos en un formulario.

Propiedades más comunes:

PROPIEDADES	DESCRIPCIÓN
FlatStyle	Cambia el Estilo del botón. Apariencia Plana o 3D.
Enabled	Habilita o Deshabilita el botón.(True o False)
Image	Carga una imagen(BMP, ICO ETC)
ImageAlign	Alinea la imagen en el botón.

LOS CONTROLES LABEL Y LINKLABEL

Las etiquetas (Label) son los controles mas comunes que se pueden encontrar en una Aplicación Windows y básicamente sirven para mostrar texto en los formularios.

En este .NET Framework existen dos controles Label y son.

- **Label:** Etiqueta estándar.
- **LinkLabel:** Igual a la estándar pero permite un hipervínculo.

Propiedades más comunes:

PROPIEDADES	DESCRIPCIÓN
BorderStyle	Especifica el tipo de borde de la Etiqueta. Por defecto sin borde.
FlatStyle	Estilo de la Etiqueta, plana o 3D.
Image	Cargar una Imagen.
ImageAlign	Alinear una imagen en la Etiqueta.
LinkArea	(LinkLabel)Especifica el rango del texto que debe mostrarse como Link.
LinkColor	(LinkLabel) Color del Link.
Links	---
LinkVisited	Cambia el color si el Link ya se visito.
TextAlign	Alinea el Texto.
VisitedLinkColor	Color para Link visitado.

PRACTICA 2 (CONTROL BUTTON)

1. Agregar un Control Button al Formulario.
2. Analizamos el archivo Form1.Designer.cs .
3. Cambiamos el nombre del Control Button, con la propiedad Name.
4. Cambiamos la etiquete del Control Button, con la propiedad Text. (Run)
5. Cambiamos el color del Control Button, con la propiedad BackColor. (Run)
6. Cambiamos el estilo del Control Button, con la propiedad FlatStyle. (Run)
7. Coloco un icono al Control Button, con la propiedad Image.
8. Deshabilitar el Control Button, con la propiedad Enabled.
9. Manejar el evento Click del Control Button.

1. obleClick sobre el Control Button carga el método del evento Click por defecto.

```
private void btnBoton_Click(object sender, EventArgs e)
{
}
```

2. Mostramos un Mensaje con MessageBox.Show. (Run)

```
private void btnBoton_Click(object sender, EventArgs e)
{
        MessageBox.Show("Se disparo el evento Click", "Atención");
}
```

10. Al dispararse el evento Click, se cambie el color del Formulario.

```
private void btnBoton_Click(object sender, EventArgs e)
{
        //MessageBox.Show("Se disparo el evento Click", "Atención");
        this.BackColor = Color.Blue;
}
```

11. Manejar el evento Click .del Formulario. Determinar que botón del Mouse se pulso.

```
private void Form1_Click(object sender, EventArgs e)
{
        MouseEventArgs click = (MouseEventArgs)e;

        if (click.Button == MouseButtons.Left)
                MessageBox.Show("Presiono el botón Izquierdo", "Atención");
        else if (click.Button == MouseButtons.Right)
                MessageBox.Show("Presiono el Botón Derecho", "Atención");
        else if (click.Button == MouseButtons.Middle)
                MessageBox.Show("Presiono el botón del Medio", "Atención");
}
```

PRACTICA 3 (CONTROL LABEL Y LINKLABEL)

1. Agregar un Label al Formulario.

2. Analizamos el archivo Form1.Designer.cs .

3. Cambiamos el nombre del Label, con la propiedad Name.

4. Cambiamos el Texto del Label, con la propiedad Text.

5. Desplegamos el Cuadro de Dialogo de la propiedad Font y cambiamos Tipo Letra, Tamaño y estilo.

6. Cambiamos el color del Texto, desplegando la paleta de colores personalizados de la propiedad ForeColor.

7. Aplicamos un borde 3D con la propiedad BorderStyle.

8. Manejar el evento MouseMove del Control Label.

   1. En la ventana de eventos elijo MouseMove.

```
private void lblEtiqueta_MouseMove(object sender, MouseEventArgs e)
{

}
```

   2. Cambiamos el color de la propiedad BackColor. (Run)

```
private void lblEtiqueta_MouseMove(object sender, MouseEventArgs e)
{
    lblEtiqueta.BackColor = Color.Cyan;
}
```

9. Al dispararse el evento MouseLeave, se restablezca el color de fondo de la etiqueta.

```
private void lblEtiqueta_MouseLeave(object sender, EventArgs e)
{
    lblEtiqueta.BackColor = System.Drawing.SystemColors.Control;
}
```

10. Cambiar el estilo del cursor al dispararse los eventos MouseMove y MouseLeave.

```
private void lblEtiqueta_MouseMove(object sender, MouseEventArgs e)
{
    lblEtiqueta.BackColor = Color.Cyan;
    lblEtiqueta.Cursor = Cursors.Hand;
}

private void lblEtiqueta_MouseLeave(object sender, EventArgs e)
{
    lblEtiqueta.BackColor = System.Drawing.SystemColors.Control;
    lblEtiqueta.Cursor = Cursors.Arrow;
}
```

EL CONTROL TEXTBOX

Los cuadros de Textos se utilizan siempre que deseemos que el usuario introduzca algún tipo de datos, en ellos se pueden introducir cualquier tipo de caracteres.

El Control TextBox deriva de la clase System.Windows.Forms.TextBoxBase, esta clase brinda la funcionalidad básica para la manipulación de texto, como cortar, pegar y también brinda todos los eventos básicos.

De la clase TextBoxBase también deriva el Control RichTextBox.

Propiedades más comunes:

PROPIEDADES	DESCRIPCIÓN
CausesValidation	Estando en true se disparan dos eventos(Validating y Validated) al recibir el foco.
CharacterCasing	Convierte el Texto ingresado a Minúscula, Mayúscula o Normal.
MaxLength	Cantidad Máxima de caracteres que acepta.
Multiline	True o False, para aceptar multilíneas de texto.
PasswordChar	Especifica si el carácter de la contraseña debe sustituir a los caracteres ingresados.
ReadOnly	Especifica si es de solo Lectura.
ScrollBars	En caso de ser multilínea si se muestra el Scroll.
SelectedText	
SelectionLength	
SelectionStart	
WordWrap	

Eventos más comunes:

PROPIEDADES	DESCRIPCIÓN
Enter ***Leave*** ***Validating*** ***Validated***	Estos Cuatro eventos se ejecutan en este orden, son conocidos como eventos Foco. Cada vez que un control recibe el foco se disparan estos eventos, salvo Validating y Validated cuya propiedad CauseValidation del control que recibe el foco debe estar en true.
KeyDown ***KeyPress*** ***KeyUp***	Estos eventos permiten controlar lo que se introduce en los controles. KeyDown y KeyUp, reciben el código de la tecla que se pulsa. KeyPress recibe el carácter de la tecla que se pulsa.
TextChanged	Se dispara cada vez que hay un cambio en el TextBox.

PRACTICA 4 (CONTROL TEXTBOX)

1. Agregar un Control TextBox al Formulario.

2. Analizamos el archivo Form1.Designer.cs .

3. Cambiamos el nombre del Control TextBox, con la propiedad Name.

4. Cambiamos la cantidad de Caracteres que acepta el Control TextBox, con la propiedad MaxLength. (Run)

5. Cambiamos la propiedad CharacterCasing del Control TextBox, para que cambie a mayúsculas los caracteres que se ingresan. (Run)

6. En el evento Click del botón creado anteriormente, cancelamos las líneas de código anteriores y colocamos el código para cambiar el color de fondo del TextBox(Propiedad BackColor) si el TextBox se encuentra vació. (Run)

```
private void btnBoton_Click(object sender, EventArgs e)
{
   //MessageBox.Show("Se disparo el evento Click", "Atención");
   //this.BackColor = Color.Blue;
   if (txtApellido.Text == "")
       txtApellido.BackColor = Color.Red;
   else
       txtApellido.BackColor = System.Drawing.SystemColors.Control;
}
```

7. Manejar el evento KeyPress, para ingresar solo Números

```
.       private void txtApellido_KeyPress(object sender, KeyPressEventArgs e)
        {
           if ((e.KeyChar < 48 || e.KeyChar > 59) && e.KeyChar != 8)
               e.Handled = true;
        }
```

8. Agregar otro Control TextBox, cambiar la propiedad Name.

9. Colocar en True la propiedad Multiline del nuevo Control TextBox. (Run)

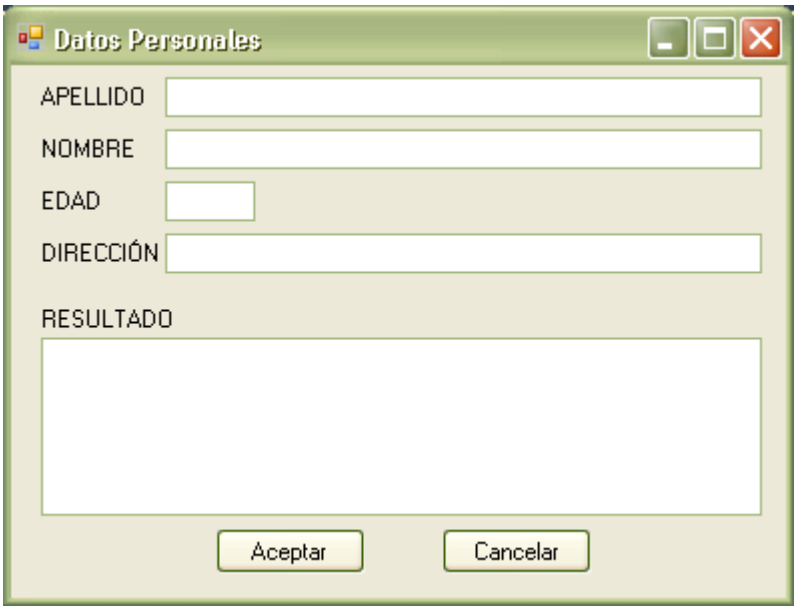
10. Cambiar la propiedad ScrollBars del nuevo Control TextBox. (Run)

11. Manejar el evento Leave del nuevo Control TextBox. Para mostrar cuantos Caracteres se ingresaron una vez que el control pierde el foco.

```
private void txtNuevo_Leave(object sender, EventArgs e)
{
   MessageBox.Show("Tiene "+ txtNuevo.Text.Length +" Caracteres");
}
```

PRACTICA 5 (APLICACIÓN WINDOWS)

Generar un Formulario con los controles y diseño que se muestran en la siguiente imagen:



Al presionar el botón aceptar se debe validar que los text Apellido, Nombre, Edad y Dirección tengan datos, en caso de estar vacíos marcarlos de color rojo.
Si pasa la validación los datos se deben escribir en el text de resultado (TextBox multilínea) con el siguiente formato:

Apellido y Nombre: XXXXXXXXXXXXXXX
Edad: XXX
Dirección: XXXXXXXXXXXXXXXXXXXX

En el campo Edad solo debe aceptar Números.
En todos los campos limitar la cantidad de caracteres y pasarlos a mayúsculas.

Al presionar el botón Cancelar se debe cerrar la aplicación.

MAS CONTROLES DE FORMULARIO

CONTROLES RADIOBUTTON Y CHECKBOX

Anteriormente comentamos que estos controles derivan de la misma clase Base que el control Button.

El control RadioButton se muestra como una etiqueta con un circulo, el mismo puede o no mostrarse por defecto como seleccionado. Estos controles RadioButton se utilizan para seleccionar entre una o mas opciones de forma excluyentes Ej. Hombres o Mujeres. Para colocar un grupo de Controles RadioButton y que de forma automatica siempre se seleccione una sola de las opciones, los mismos deben colocarse dentro de un GroupBox.

Un Control CheckBox se muestra como una etiqueta con una pequeña caja a su izquierda, estos controles CheckBox suelen utilizarse para seleccionar varias opciones, Ej. Un Cuestionario.

Propiedades mas usadas en el Control RadioButton:

PROPIEDADES	DESCRIPCIÓN
Appearance	*Como se muestra el control, con el circula a la izq. O derch.*
AutoCheck	*Indica si al seleccionarlo se marca de forma automatica.*
CheckAlign	*Indica la alineación de la casilla dentro del control.*
Checked	*Indica si el control esta activado o no. (True o False)*

Eventos mas usadas en el Control RadioButton:

EVENTOS	DESCRIPCIÓN
CheckedChanged	*Se dispara cuando hay cambios en el control RadioButton.*
Click	*Se dispara cuando se hace click sobre el control RadioButton, no es lo mismo que CheckedChanged ya que varios click solo cambia el estado una sola vez y CheckedChanged no vuelve a ser comprobado.*

Propiedades mas usadas en el Control CheckBox::

PROPIEDADES	DESCRIPCIÓN
CheckState	*Indica el estado, Chequeado, no Chequeado y Indeterminado.*
ThreeState	*Indica si el control permite tres estados o solo dos.*

Eventos mas usadas en el Control CheckBox:

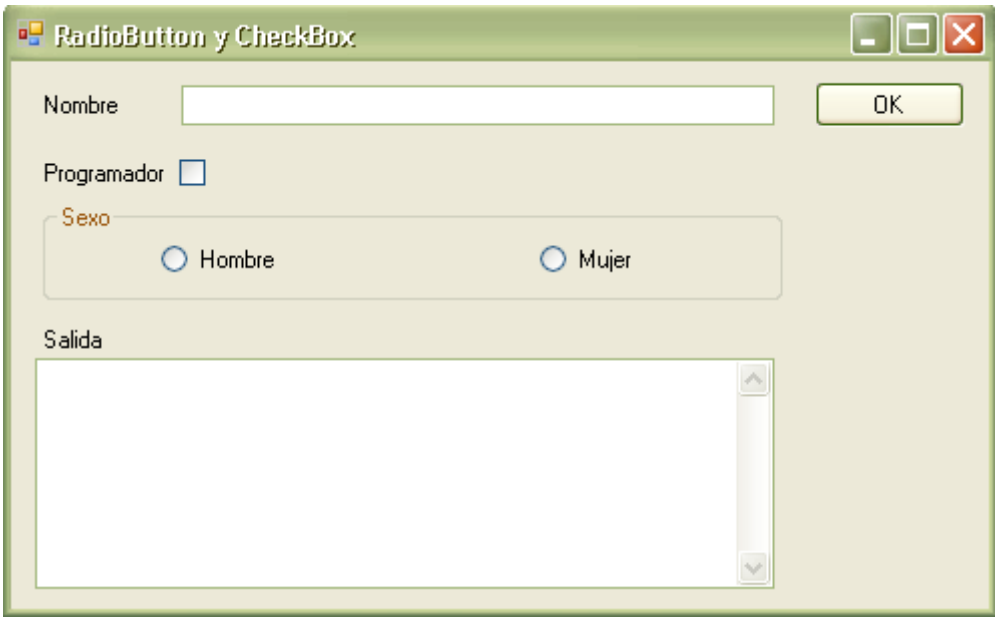
EVENTOS	DESCRIPCIÓN
CheckedChanged	*Se dispara cuando hay cambios en el control CheckBox.*
CheckStateChanged	*Se dispara cuando produce un cambio en la propiedad CheckState.*

CONTROL GROUPBOX

El control GroupBox se utiliza a menudo para agrupar lógicamente un conjunto de controles tales como el RadioButton y CheckBox, y para proporcionar un título y un marco alrededor de este conjunto. Tenga en cuenta que si coloca la propiedad Enabled en false del Control GroupBox, todos los controles que esten dentro de el se deshabilitaran.

PRACTICA 6 (RADIOBUTTON Y CHECKBOX)

Generar un Formulario con los controles y diseño que se muestran en la siguiente imagen:



Cambiar las Propiedades Name y Text de todos los Controles.

Los RadioButtons estan dentro de un control GroupBox.

Cambiar la Propiedad CheckAling.

Cambiar la Propiedad Checked del CheckBox Programador, para activarlo.

Cambiar la Propiedad Checked del RadioButton Hombre, para activarlo.

Crear un Metodo ValidarOk.

```
private void ValidarOk()
{
     //Habilita el Boton, siempre y cuando txtNombre tenga datos.
     btnOk.Enabled = (txtNombre.BackColor != Color.Red);
}
```

Manejamos el Evento Validating del Control TextBox Nombre.

```
private void txtNombre_Validating(object sender, CancelEventArgs e)
{
    TextBox tb = (TextBox)sender;

    if (tb.Text.Length == 0)
        tb.BackColor = Color.Red;
    else
        tb.BackColor = System.Drawing.SystemColors.Window;

    ValidarOk();
}
```

Manejamos el Evento Load del Formulario.

```
private void fmrRbtnCb_Load(object sender, EventArgs e)
{
     //Deshabilito el Boton Ok.
     btnOk.Enabled = false;
}
```

Manejamos el Evento Click del Boton Ok.

```
private void btnOk_Click(object sender, EventArgs e)
{
    //No valido datos ya que si el Boton esta Habilitado
    //es porque paso el Evento Validating del Nombre.

    String salida; //Declaro una variable para armar la salida.

    salida = "Nombe: " + txtNombre.Text + "\r\n";
    salida += "Ocupacion: " + (string)(cbxProgramer.Checked ?
    "Programador" : "No es Programador") + "\r\n";
    salida += "Sexo: " + (string)(rbtnHombre.Checked ? "Hombre" :
    "Mujer") + "\r\n";

    //Vuelco la salida al TextBox Salida.
    txtSalida.Text = salida;
}
```

El operador condicional ? evalua una expresion si es true la primer expresion que sigue a ? se convierte en el resultado, si el false la segunda expresion se convierte en el resultado.

CONTROL RICHTEXTBOX

El Control RichTextBox deriva de la clase TextBoxBase al igual que el control TextBox, por esto mismo es que comparten muchas de las propiedades y eventos.
A diferencia del control TextBox el RichTextBox soporta Texto con formato (Negrita, subrayado, cursiva), esto se logra utilizando un estandar conocido como Texto con Formato Enriquecido o RTF.
Posee mas propiedades que el Control TextBox, estas propiedades que se agregan tienen que ver mas con la selección del texto, ya que normalmente cuando uno da formato a un texto se lo hace por partes y en forma diferente.
Si uno no realiza una selección y aplica un cambio de formato este comenzara a aplicarse desde donde quedo el cursor.

Las Propiedades mas utilizadas son:

PROPIEDADES	DESCRIPCIÓN
CanRedo	*True cuando la ultima operación deshecha puede aplicar rehacer.*
CanUndo	*True si es posible deshacer la ultima accion en el control. CanUndo se define en TextBoxBase, por lo que está disponible para los controles TextBox también.*
RedoActionName	*Contiene el nombre de una acción que se llevara acabo con el metodo rehacer.*
DetectUrls	*S e coloca en true para detectar url y darle formato.*
Rtf	*Igual a text, solo que esta contiene el Texto en formato RTF.*
SelectedRtf	*Se utiliza para mantener el texto seleccionado en el control en formato RTF. Asi al copiar no pierde el formato.*
SelectedText	*Igual a SelectedRtf pero se pierde el formato.*
SelectionAlignment	*Representa la alineación del Texto Seleccionado.*
SelectionBullet	
BulletIndent	
SelectionColor	*Color del Texto seleccionado.*
SelectionFont	*Cambia la fuente del Texto en la selección.*
SelectionLength	*Establece o recupera la longitud de la selección.*
SelectionType	*Contiene Informacion de la selección.*
ShowSelectionMargin	*En True deja un margen a la izq. Para que sea mas facil seleccionar el texto.*
UndoActionName	*Obtiene el nombre de la acción que se utilizara si se opta por*

	deshacer algo.
SelectionProtected	*Puede establecer que ciertas partes de la selección no deben ser cambiadas.*

Eventos mas usadas en el Control RichTextBox:

EVENTOS	DESCRIPCIÓN
LinkClicked	*Se dispara cuando se hace Click en un Enlace dentro del Texto.*
Protected	*Se dispara cuando se intenta modificar un texto que se marco como protegido.*
SelectionChanged	*Se dispara cuando se cambia la selección, si no quiero cambiar la selección puedo parar el cambio aquí.*

PRACTICA 7 (RICHTEXTBOX)

Generar un Formulario con los controles y diseño que se muestran en la siguiente imagen:

Cambiar las Propiedades Name y Text de todos los controles.

Cambiar la Propiedad MaxLength del TextBox Tamaño de Fuente.

Cambiar la Propiedad Anchor de todos los controles con los siguientes valores:

CONTROL	VALOR
Guardar y Abrir	*Bottom.*
RichTextBox	Top, Left, Bottom, Right
Otros	*Top.*

Cambiar la Propiedad MinimumSize del Formulario.

Abrir el Evento Click del Boton Negrita y colocar el siguiente codigo.

```
private void btnNegrita_Click(object sender, EventArgs e)
{
    Font viejaFuente; //Declaro una variable Class Font para la Fuente Vieja.
    Font nuevaFuente; //Declaro una variable Class Font para la Fuente Nueva.

    //Asigno a viejaFuente el tipo de fuente seleccionado.
    //Obtengo el tipo de Fuente atravez de la Propiedad SelectionFont
    //del Control RichTextBox.
    viejaFuente = rtxtEditor.SelectionFont;

    if (viejaFuente.Bold) //Pregunto si ya es Negrita.
        //Si ya Tenia Negrita, establece nuevaFuente sin Negrita.
        nuevaFuente = new Font(viejaFuente, viejaFuente.Style & ~FontStyle.Bold);
    else
        //Si no Tenia Negrita, establece nuevaFuente con Negrita.
        nuevaFuente = new Font(viejaFuente, viejaFuente.Style | FontStyle.Bold);

    //Establesco la nuevaFuente a la Selección.
    rtxtEditor.SelectionFont = nuevaFuente;

    //Realizo un foco para posicionarme en el Control.
    rtxtEditor.Focus();
}
```

Font Inicializa un nuevo objeto Font que utiliza el objeto Font existente especificado y la enumeración FontStyle (Negrita, Italic etc). Font(Font, FontStyle) mas Info en : http://msdn.microsoft.com/es-es/library/system.drawing.font.aspx

Abrir el Evento Click del Boton Subrayado y colocar el siguiente codigo.

```
private void btnSubrayado_Click(object sender, EventArgs e)
{
    Font viejaFuente; //Declaro una variable Class Font para la Fuente Vieja.
    Font nuevaFuente; //Declaro una variable Class Font para la Fuente Nueva.

    //Asigno a viejaFuente el tipo de fuente seleccionado.
    //Obtengo el tipo de Fuente atravez de la Propiedad SelectionFont
    //del Control RichTextBox.
    viejaFuente = rtxtEditor.SelectionFont;

    if (viejaFuente.Underline) //Pregunto si ya es Subrayado.
        //Si ya Tenia Subrayado, establece nuevaFuente sin Subrayado.
        nuevaFuente = new Font(viejaFuente, viejaFuente.Style &
~FontStyle.Underline);
```

```
        else
            //Si no Tenia Subrayado, establece nuevaFuente con Subrayado.
            nuevaFuente = new Font(viejaFuente,viejaFuente.Style | FontStyle.Underline);

        //Establesco la nuevaFuente a la Selección.
        rtxtEditor.SelectionFont = nuevaFuente;

        //Realizo un foco para posicionarme en el Control.
        rtxtEditor.Focus();
    }
```

Abrir el Evento Click del Boton Cursiva y colocar el siguiente codigo.

```
    private void btnCursiva_Click(object sender, EventArgs e)
    {
        Font viejaFuente; //Declaro una variable Class Font para la Fuente Vieja.
        Font nuevaFuente; //Declaro una variable Class Font para la Fuente Nueva.

        //Asigno a viejaFuente el tipo de fuente seleccionado.
        //Obtengo el tipo de Fuente atravez de la Propiedad SelectionFont
        //del Control RichTextBox.
        viejaFuente = rtxtEditor.SelectionFont;

        if (viejaFuente.Italic) //Pregunto si ya es Cursiva.
            //Si ya Tenia Cursiva, establece nuevaFuente sin Cursiva.
            nuevaFuente = new Font(viejaFuente, viejaFuente.Style & ~FontStyle.Italic);
        else
            //Si no Tenia Cursiva, establece nuevaFuente con Cursiva.
            nuevaFuente = new Font(viejaFuente, viejaFuente.Style | FontStyle.Italic);

        //Establesco la nuevaFuente a la Selección.
        rtxtEditor.SelectionFont = nuevaFuente;

        //Realizo un foco para posicionarme en el Control.
        rtxtEditor.Focus();
    }
```

Abrir el Evento Click del Boton Centrar y colocar el siguiente codigo.

```
    private void btnCentrado_Click(object sender, EventArgs e)
    {
        //Atravez de la propiedad SelectionAlignment, obtiene la alineación
        //del texto que se selecciono.
        //Pregunto si es en centrado.
        if (rtxtEditor.SelectionAlignment == HorizontalAlignment.Center)

            //Si estaba Centrado lo alinea a la Izq.
            rtxtEditor.SelectionAlignment = HorizontalAlignment.Left;

        else

            //Si no estaba centrado lo centra.
            rtxtEditor.SelectionAlignment = HorizontalAlignment.Center;

        rtxtEditor.Focus();
    }
```

Creo un Metodo propio llamado AplicarTamanioText y coloco el siguiente codigo:

```
    //Metodo propio Aplicar Tamaño de Texto
    private void AplicarTamanioText(string textSize)
    {
        //Declaro una Var. y le asigno el valor pasado, previa Conversion.
```

```
        float nuevoTam = Convert.ToSingle(textSize);

        //Declaro un Objeto FontFamily para el tipo de Fuente actual.
        FontFamily actualFuenteFamily;

        //Declaro un Objeto Font para la nueva Fuente.
        Font nuevaFuente;

        //De la selección actual del RichTextBox obtengo
        //el tipo de familia de la fuente.
        actualFuenteFamily = rtxtEditor.SelectionFont.FontFamily;

        //New Font Instancio el objeto nuevaFuente segun
        //tipo de FFuente y tamaño.
        nuevaFuente = new Font(actualFuenteFamily, nuevoTam);

        rtxtEditor.SelectionFont = nuevaFuente;
    }
```

Abrir el Evento KeyPress del TextBox Tamaño y colocar el siguiente codigo.

```
    private void txtTamanio_KeyPress(object sender, KeyPressEventArgs e)
    {
        //Valido que si se presiono una tecla que no se un numero,
        //Backspace o enter, se controle el evento.
        if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar != 8 && e.KeyChar != 13)
            e.Handled = true;

        //Pregunto si es Enter.
        else if (e.KeyChar == 13)
        {
            //Si es Enter, pregunto si la cantidad de caracteres es mayor a 0.
            if (txtTamanio.Text.Length > 0)
                //Llamo al metodo Aplicar Tamaño al Texto.
                AplicarTamanioText(txtTamanio.Text);

            //Si es Enter y no se ingresaron caracteres, controlo el evento.
            e.Handled = true;

            rtxtEditor.Focus();
        }
    }
```

Abrir el Evento Validated del TextBox Tamaño y colocar el siguiente codigo:

```
    private void txtTamanio_Validated(object sender, EventArgs e)
    {
        //El Evento Validated ocurre despues de la validación(Evento Validating).
        //Llama al Metodo AplicarTamanioText y como parametro pasa,
        //La propiedad Text del objeto sender(Objeto que disparo el evento)
        //Del cual se hace un Cast del tipo TextBox.
        //Es lo mismo que pasarle txtTamanio.text.
        AplicarTamanioText(((TextBox)sender).Text);
        rtxtEditor.Focus();
    }
```

Abrir el Evento LinkClicked del RichTextBox y Colocar lo siguiente:

```
    private void rtxtEditor_LinkClicked(object sender, LinkClickedEventArgs e)
    {
        //La calse Process brinda acceso a procesos locales y remotos.
        //El metodo Start inicia un proceso y asocia el recurso a un nuevo componentes.
        System.Diagnostics.Process.Start(e.LinkText);
```

```
}
```

Abrir el Evento Click del Boton Abrir y colocar el siguiente codigo.

```
 private void btnAbrir_Click(object sender, EventArgs e)
{
    try
    {
        rtxtEditor.LoadFile("Test.rtf");
    }
    catch (System.IO.FileNotFoundException)
    {
        MessageBox.Show("No se pudo cargar el archivo");
    }
}
```

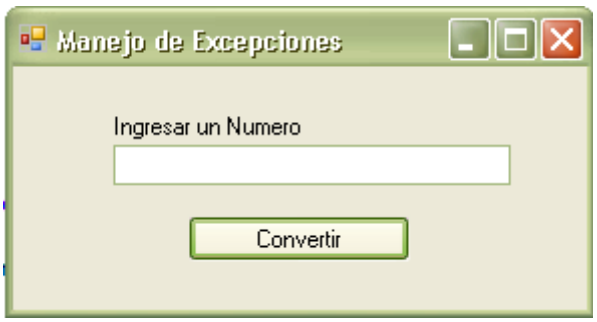
Abrir el Evento Click del Boton Guardar y colocar el siguiente codigo.

```
private void btnGuardar_Click(object sender, EventArgs e)
{
    try
    {
        rtxtEditor.SaveFile("Test.rtf");
    }
    catch (System.Exception err)
    {
        MessageBox.Show(err.Message);
    }
}
```

TRATAMIENTO DE ERRORES

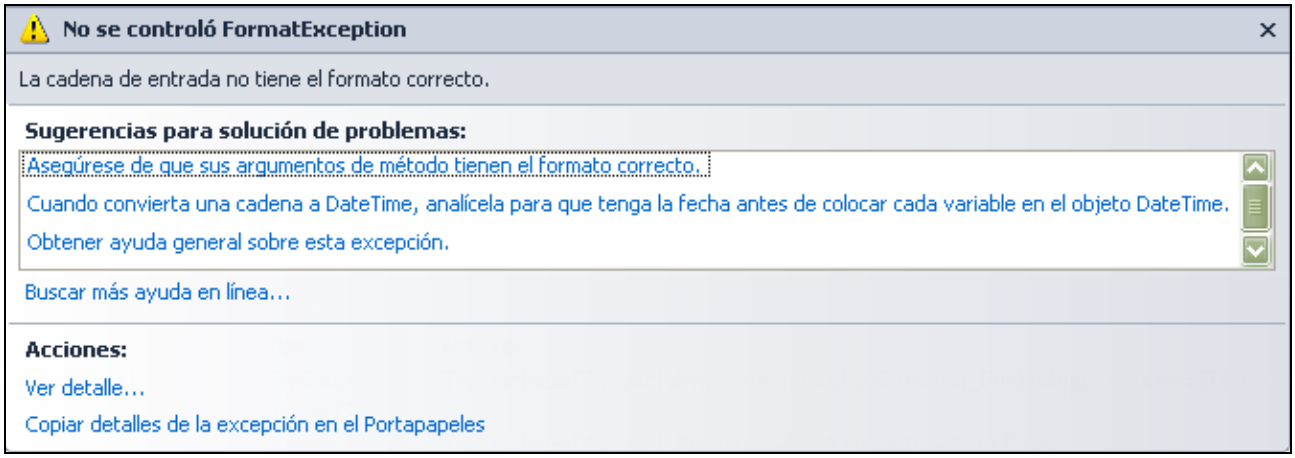
Las aplicaciones que creamos normalmente poseen errores, algunos pueden ser detectados durante la compilacion (Errores de sintaxis, semanticos, etc.) y otros errores de logica de programacion, que aparecen cuando se dan ciertas circustancias.
C# ofrece un sistema para manejar estos errores, que se lo llama **“MANEJO DE EXCEPCIONES”** (exception handling).
Una Excepcion es una alteracion en el flujo normal del programa, el sistema de manejo de excepciones, permite manejar estos errores.

Supongamos que tenemos una aplicación, en la que ingresamos un numero y lo convertimos a entero:

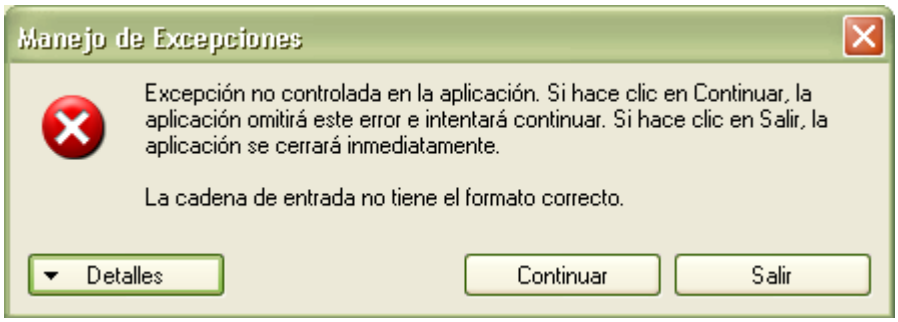



```
private void btnConvertir_Click(object sender, EventArgs e)
{
    int num = Convert.ToInt32(txtNumero.Text);
}
```

Aca estamos tratando de asignar a la variable num del tipo entero el valor que se ingresa en el TextBox, previa Conversion. Si el usuario ingresa un numero la aplicación funcionaria correctamente, pero si se ingresaran otros caracteres que no fueran numeros, la Conversion fallara y aparecera un mensaje indicando el error y luego la aplicación se cerrara.
Si ejecutamos la aplicación desde el entorno de desarrollo mostraria el siguiente mensaje indicando la linea que causo el error:



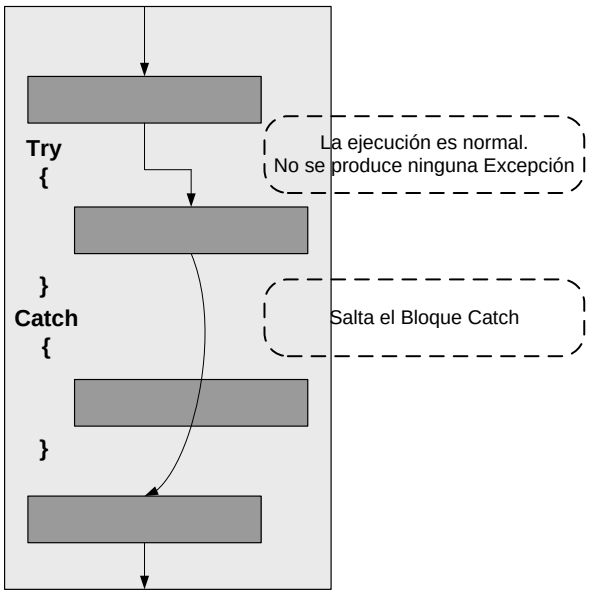
Si ejecutamos la aplicación externamente veriamos el siguiente mensaje:

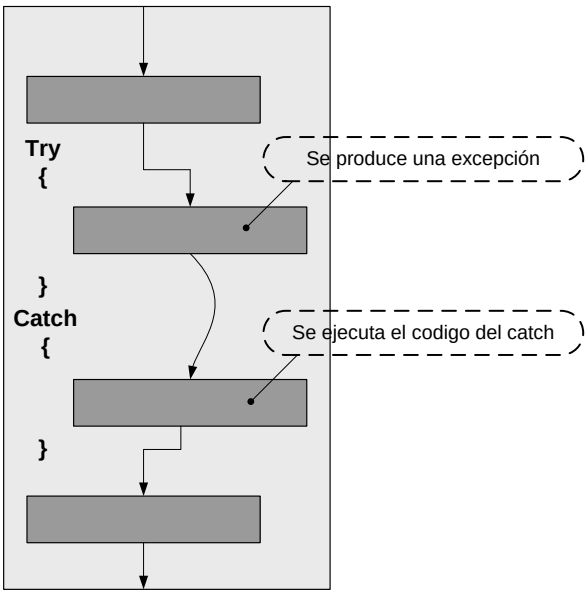


Entonces lo que sucede es que el metodo ToInt32 de la clase Convert lanzo una Excepcion, nosotros deberiamos capturar estas excepciones para que nuestra aplicación siga funcionando.
Para manejar estas excepciones encerramos el codigo que puede generar un error en un bloque Try…Catch.

```
private void btnConvertir_Click(object sender, EventArgs e)
{
    try
    {
        int num = Convert.ToInt32(txtNumero.Text);
    }
    catch
    {
        MessageBox.Show("Se ha producido una Excepción.");
    }
}
```

Esta es la forma basica de manejar una excepcion, si se produce un error en el Try salta automaticamente al Catch y luego sigue el flujo de ejecucion normal de la aplicación.

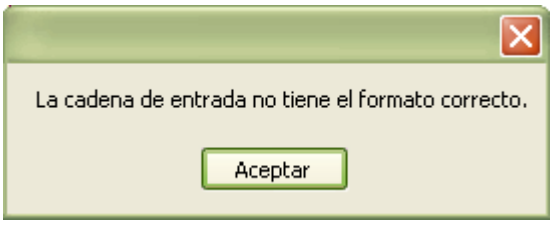




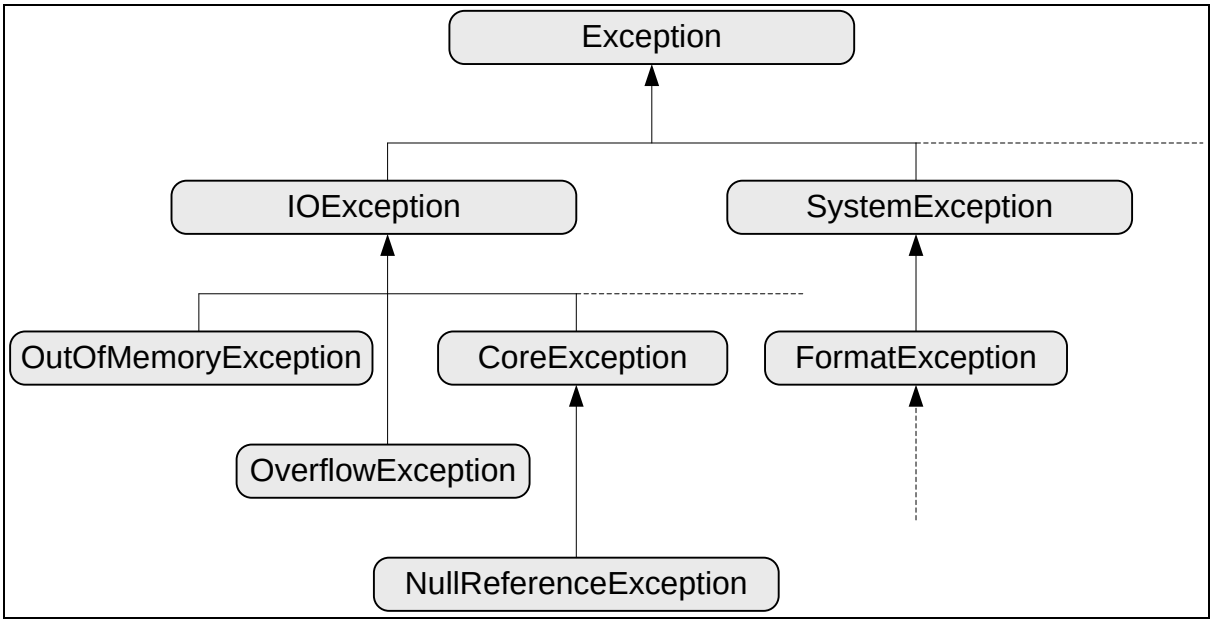
Clases de Excepciones

Hasta ahora capturamos una excepcion que lanzo un metodo encerrado en un bloque Try…Catch. ¿Qué paso? ¿Qué produjo el error? ¿Qué le informamos al usuario? Existe mas informacion que se puede manejar en el bloque Catch.
Vamos a modificar el bloque Catch anterior para saber que produjo el error.

```
private void btnConvertir_Click(object sender, EventArgs e)
{
    try
    {
        int num = Convert.ToInt32(txtNumero.Text);
    }
     //Los identificadores de objetos que se utilizan
     //como fe son arbitrarios y solo existen en el ambito
     //del bloque Catch.
    catch(FormatException fe)
    {
        MessageBox.Show(fe.Message);
    }
}
```

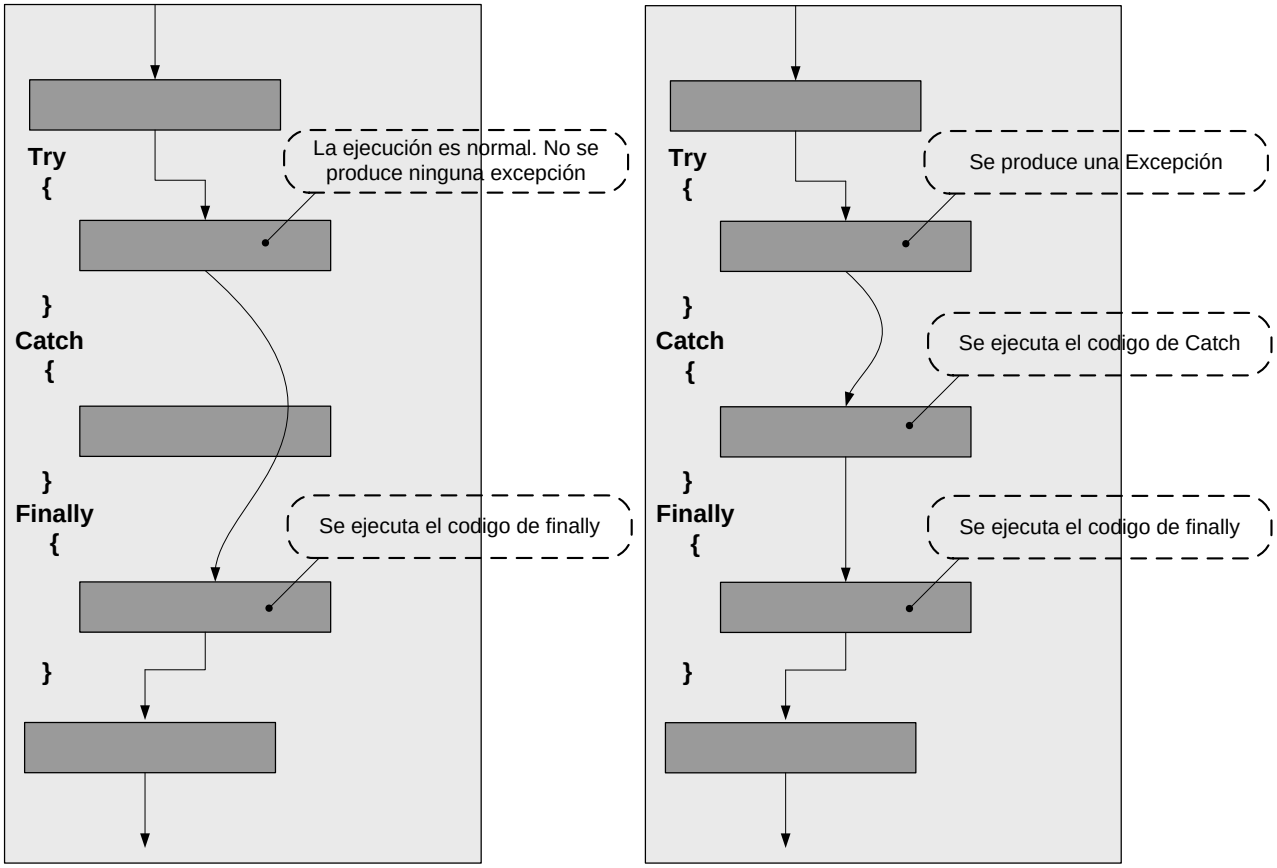


Ahora cuando el metodo ToInt32 lance un error, no mostrara un mensaje generico, si no algo mas detallado de lo que produjo el error.
Las clases de Excepciones en .Net descienden de la clase Excepcion, la cual ofrece una cantidad de propiedades y metodos que nos ayudaran en la obtension de mayor informacion y en el tratamiento de lo que genero las excepciones.
Algunas de las clases que decienden de Excepcion son:



Si queremos capturar un error del tipo FormatException, pasamos como parametro un objeto del tipo FormatException, o un objeto del tipo superior como SystemException.

Tambien se puede especificar un bloque finally el cual se ejecutara ocurra o no una excepcion.



Reglas de los Bloques Catch

- Los bloques Catch pueden contener cualquier cantidad de lineas, pero es buena practicas mantenerlas breves.
- Solo puede haber un bloque Catch sin especificación de clase de Excpcion.
- Cada bloque Catch debe tratar un solo tipo de Excpcion.
- Las clases de Excepciones mas especificas deben ir primero.

CONTROLES LISTBOX Y CHECKEDLISTBOX

Los cuadrosde listan se usan para mostrar una lista de cadenas o valores donde el usuario puede seleccionar uno o mas a la vez.
La clase ListBox se deriva de la clase ListControl que proporciona la funcionalidad basica para todos los controles de cuadro de lista que proporciona el .NET Framework.
Otro tipo de cuadro de lista disponible se llama CheckedListBox que deriva de la clase ListBox, que proporciona una lista al igual que el ListBox , pero además de las cadenas de texto proporciona una casilla de verificación para cada elemento en la lista.

Propiedades comunes para ListBox y CheckedListBox:

PROPIEDADES	DESCRIPCIÓN
SelectedIndex	*Indica el indice seleccionado de la lista. Si es una selección multiple, devuelve el primer indice seleccionado.*
ColumnWidth	*Especifica el ancho de columna en un cuadro de lista con varias columnas.*
Items	*Devuelve la colección de items de todos los elementos en el cuadro de lista. Se pueden agregar o quitar elementos de esta colección.*
MultiColumn	*Un cuadro de lista puede tener mas de una columna, esta propiedad especifica si el valor se muestra en cantidades de columnas.*
SelectedIndices	*Devuelve una colección con todos los indices de los elementos seleccionados.*
SelectedItem	*Devuelve el elemento seleccionados, si son varios devuelve el primero.*
SelectedItems	*Devuelve una colecciones con todos los elementos seleccionados.*
SelectionMode	*Modo de selección esisten 4 modos.*
Sorted	*En true ordena los elementos de la lista*
Text	*Se selecciona en el list el elemento que coincida con el Text.*

CheckedIndices	*Devuelve una colección con todos los indices de los elementos del CheckedListBox que tengan un estado chequeado.*
CheckedItems	*Devuelve una colección con todos los elementos del CheckedListBox que tengan un estado chequeado.*
CheckOnClick	*(Solo CheckedListBox) En true el estado del control cambia con un click.*
ThreeDCheckBoxes	*Cambia el estilo de la casilla de verificación.*

Metodos comunes para ListBox y CheckedListBox:

METODOS	DESCRIPCIÓN
ClearSelected()	*Limpia toda la selección del ListBox.*
FindString()	*Busca en el ListBox la primer cadena que en el comienzo coincida con la cadena que se especifica en este metodo.*
FindStringExact()	*Igual a la anterior pero la busqueda es exacta.*
GetSelected()	*Devuelve un valor que indica si esta seleccionado.*
SetSelected()	*Establece o Borra la selección de un elemento.*
ToString()	*Devuelve el Item seleccionado en formato String.*
GetItemChecked()	*(Solo CheckedListBox) Devuelve un valor si esta seleccionado.*
GetItemCheckState()	*(Solo CheckedListBox) Devuelve un valor que indica el estado de activacion del Item.*
SetItemChecked()	*(Solo CheckedListBox) Establece el elemento seleccionado a un estado seleccionado.*
SetItemCheckState()	*(Solo CheckedListBox) Establece el estado de activacion de un elemento.*

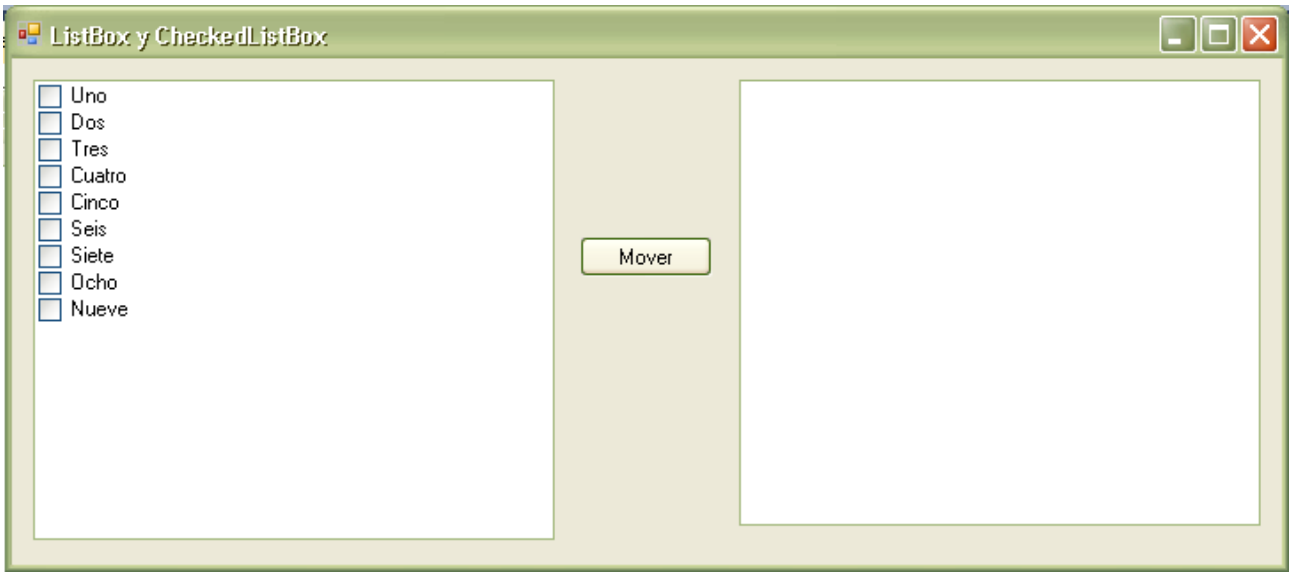
Eventos comunes para ListBox y CheckedListBox:

EVENTOS	DESCRIPCIÓN
ItemCheck	*(Solo CheckedListBox) Ocurre cuando cambia el estado del check de un elemento.*
SelectedIndexChanged	*Se produce cuando hay cambios en los indices de los elementos seleccionados.*

PRACTICA 8 (LISTBOX Y CHECKEDLISTBOX)

Ejercicio 1

Generar un Formulario con los controles y diseño que se muestran en la siguiente imagen:



Cambiar las Propiedades Name de todos los controles.

Cambiar la Propiedad Text del Formulario y Boton.

Cambiar la Propiedad CheckOnClick del CheckedListBox a True. Para que las casillas de verificacion se activen con un Click.

Abrir el editor de Items del CheckedListBox, esto se hace desde la propiedad Items hacer click en Colección …. Luego Agregar Uno, Dos, Tres, Cuatro, Cinco, Seis, Siete, Ocho, Nueve.

Agregar un elemento mas, pero desde el codigo del formulario, en este caso en el constructor de la clase del formulario.

```
public frmListBoxCheckedListBox()
{
    InitializeComponent();

    //Con el metodo Add agregamos un ultimo Item a la
    //Colección de Items. Como parametro la cadena de caracteres.
    clbValores.Items.Add("Diez");
}
```

Agregar el evento Click del Boton Mover, para colocar el codigo que mueve al ListBox los Items seleccionados en el CheckedListBox.

```
private void btnMover_Click(object sender, EventArgs e)
{
    //Pregunta si la cantidad (Count) de elementos chequeados
    //en la coleccion de items chequeados(CheckedItems) es mayor a cero.
    if (clbValores.CheckedItems.Count > 0)
    {
        //Si es mayor a cero. Limpia los Item en el ListBox.
```

```
                    lbSeleccionados.Items.Clear();

                    //La sentencia foreach recorre la coleccion de Items Seleccionados
                    //Y los asigna de a uno a la variable item del tipo String.
                    foreach (string item in clbValores.CheckedItems)
                    {
                        //Agrego los items seleccionados en la colección
                        //Al Listbox con el Metodo Add.
                        lbSeleccionados.Items.Add(item.ToString());
                    }

                    //Recorro todos los Items del CheckedListBox.
                    for (int i = 0; i < clbValores.Items.Count; i++)

                        //Con el Metodo SetItemChecked, establesco en falso la
                        //casilla de verificación (No esta seleccionado).
                        //Como parametros i-El Indice y el valor de estado en este caso
    false.
                        clbValores.SetItemChecked(i, false);
                }
```

Para mas información sobre algunos metodos la pueden encontrar en http://social.msdn.microsoft.com/search/es-es?query=SetItemChecked&x=0&y=0