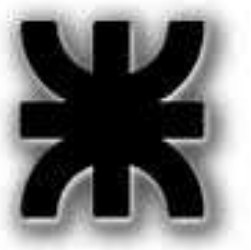


UNIVERSIDAD TECNOLOGICA NACIONAL

FACULTAD REGIONAL GENERAL PACHECO

TÉCNICO SUPERIOR EN PROGRAMACIÓN

PROGRAMACIÓN III

APUNTE TEORICO

<<< >>>

APLICACIONES WINDOWS II

Prof. ABEL OSCAR FAURE

APLICACIÓN WINDOWS II

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 círculo, el mismo puede o no mostrarse por defecto como seleccionado. Estos controles RadioButton se utilizan para seleccionar entre una o más opciones de forma excluyentes Ej. Hombres o Mujeres. Para colocar un grupo de Controles RadioButton y que de forma automática 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 más 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 automática.*
CheckAlign	*Indica la alineación de la casilla dentro del control.*
Checked	*Indica si el control esta activado o no. (True o False)*

Eventos más usados 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 más 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 más usados en el Control CheckBox:

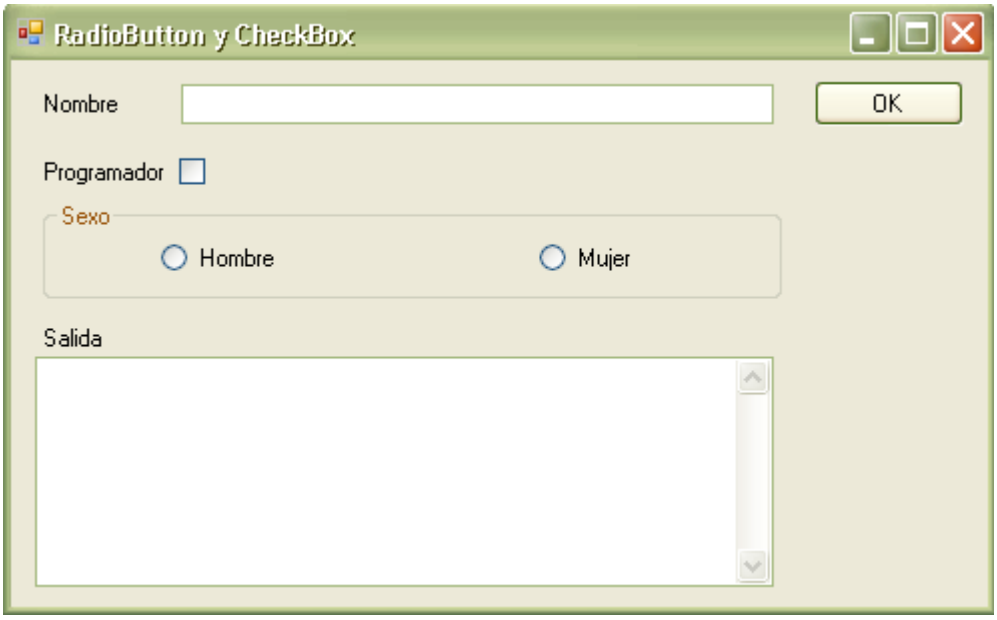
EVENTOS	DESCRIPCIÓN
CheckedChanged	*Se dispara cuando hay cambios en el control CheckBox.*
CheckStateChanged	*Se dispara cuando produce un cambio en la propiedad CheckState.*

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 estén 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:



1. Cambiar las Propiedades Name y Text de todos los Controles.
2. Los RadioButtons están dentro de un control GroupBox.
3. Cambiar la Propiedad CheckAling.
4. Cambiar la Propiedad Checked del CheckBox Programador, para activarlo.
5. Cambiar la Propiedad Checked del RadioButton Hombre, para activarlo.

6. Crear un Método ValidarOk.

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

7. 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();
}
```

8. Manejamos el Evento Load del Formulario.

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

9. Manejamos el Evento Click del Botón Ok.

```
private void btnOk_Click(object sender, EventArgs e)
{
    //No valido datos ya que si el Botón 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 ? evalúa una expresión si es true la primer expresión que sigue a ? se convierte en el resultado, si el false la segunda expresión 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 estándar 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 acción 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 método 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. Así 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 Información de la selección.*
ShowSelectionMargin	*En True deja un margen a la izq. Para que sea más fácil 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 más usados 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:

1. Cambiar las Propiedades Name y Text de todos los controles.

2. Cambiar la Propiedad MaxLength del TextBox Tamaño de Fuente.

3. 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.*

4. Cambiar la Propiedad MinimumSize del Formulario.

5. Abrir el Evento Click del Botón Negrita y colocar el siguiente código.

```
        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);

            //Establezco 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, Itálica etc.). Font(Font, FontStyle) mas info. en : http://msdn.microsoft.com/es-es/library/system.drawing.font.aspx

6. Abrir el Evento Click del Botón Subrayado y colocar el siguiente código.

```
        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);

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

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

7. Abrir el Evento Click del Botón Cursiva y colocar el siguiente código.

```
        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();
        }
```

8. Abrir el Evento Click del Botón Centrar y colocar el siguiente código.

```
        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();
        }
```

9. Creo un Método propio llamado AplicarTamanioText y coloco el siguiente código:

```
        //Método 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 Instancia el objeto nuevaFuente según
            //tipo de Fuente y tamaño.
            nuevaFuente = new Font(actualFuenteFamily, nuevoTam);

            rtxtEditor.SelectionFont = nuevaFuente;
        }
```

10. Abrir el Evento KeyPress del TextBox Tamaño y colocar el siguiente código.

```
        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 método Aplicar Tamaño al Texto.
                    AplicarTamanioText(txtTamanio.Text);

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

                rtxtEditor.Focus();
            }
        }
```

11. Abrir el Evento Validated del TextBox Tamaño y colocar el siguiente código:

```
        private void txtTamanio_Validated(object sender, EventArgs e)
        {
            //El Evento Validated ocurre después de la validación(Evento Validating).
            //Llama al Método AplicarTamanioText y como parámetro 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();
        }
```

12. Abrir el Evento LinkClicked del RichTextBox y Colocar lo siguiente:

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

13. Abrir el Evento Click del Botón Abrir y colocar el siguiente código.

```
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");
    }
}
```

14. Abrir el Evento Click del Botón Guardar y colocar el siguiente código.

```
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 compilación (Errores de sintaxis, semánticas, etc.) y otros errores de lógica de programación, que aparecen cuando se dan ciertas circunstancias.

C# ofrece un sistema para manejar estos errores, que se lo llama **“MANEJO DE EXCEPCIONES”** (exception handling).

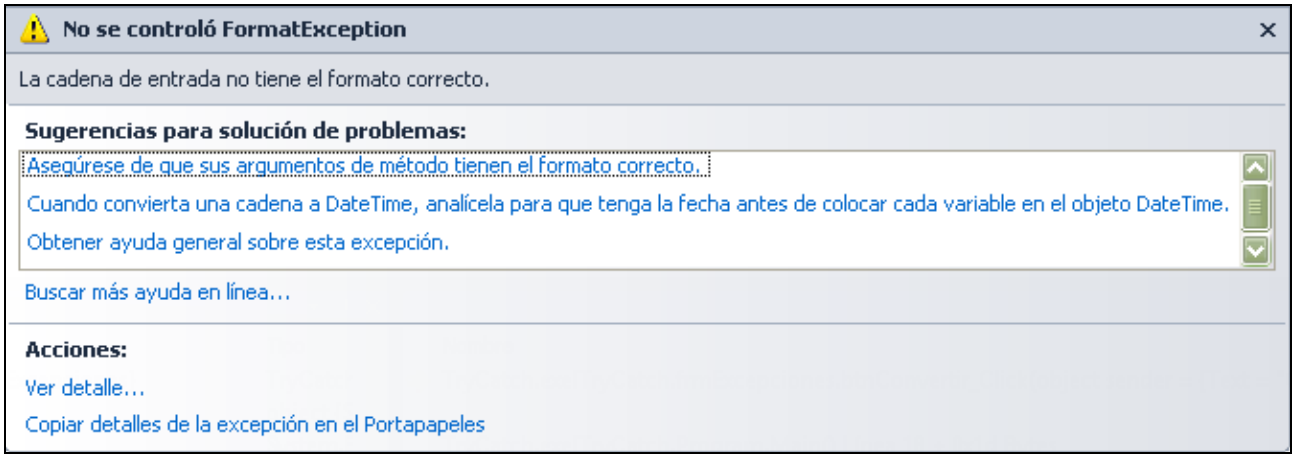
Una Excepción es una alteración en el flujo normal del programa, el sistema de manejo de excepciones, permite manejar estos errores.

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

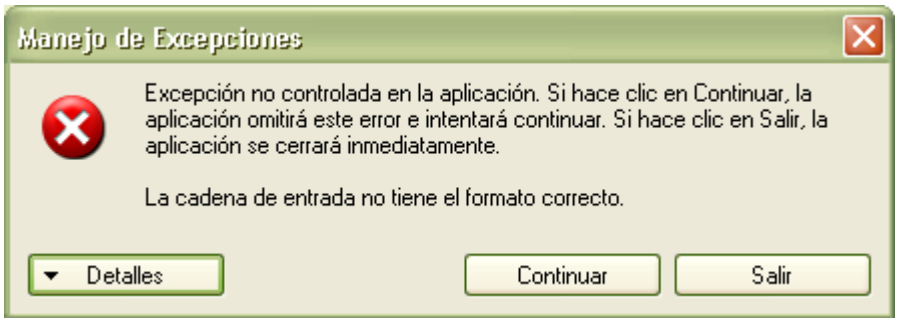

```
private void btnConvertir_Click(object sender, EventArgs e)
{
     int num = Convert.ToInt32(txtNumero.Text);
}
```

Acá 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 números, la Conversion fallara y aparecerá un mensaje indicando el error y luego la aplicación se cerrara.

Si ejecutamos la aplicación desde el entorno de desarrollo mostraría el siguiente mensaje indicando la línea que causo el error:



Si ejecutamos la aplicación externamente veríamos el siguiente mensaje:

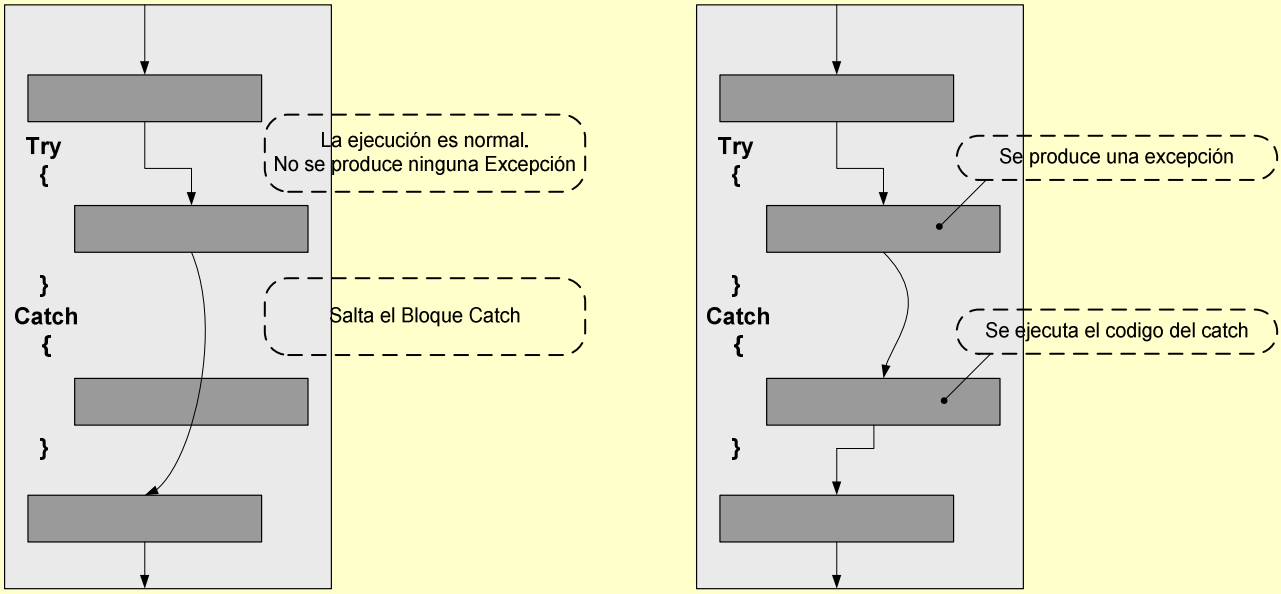


Entonces lo que sucede es que el método ToInt32 de la clase Convert lanzo una Excepción, nosotros deberíamos capturar estas excepciones para que nuestra aplicación siga funcionando.

Para manejar estas excepciones encerramos el código 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 básica de manejar una excepción, si se produce un error en el Try salta automáticamente al Catch y luego sigue el flujo de ejecución normal de la aplicación.

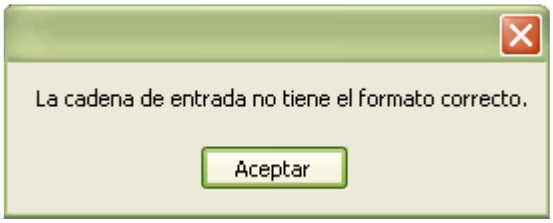


Clases de Excepciones

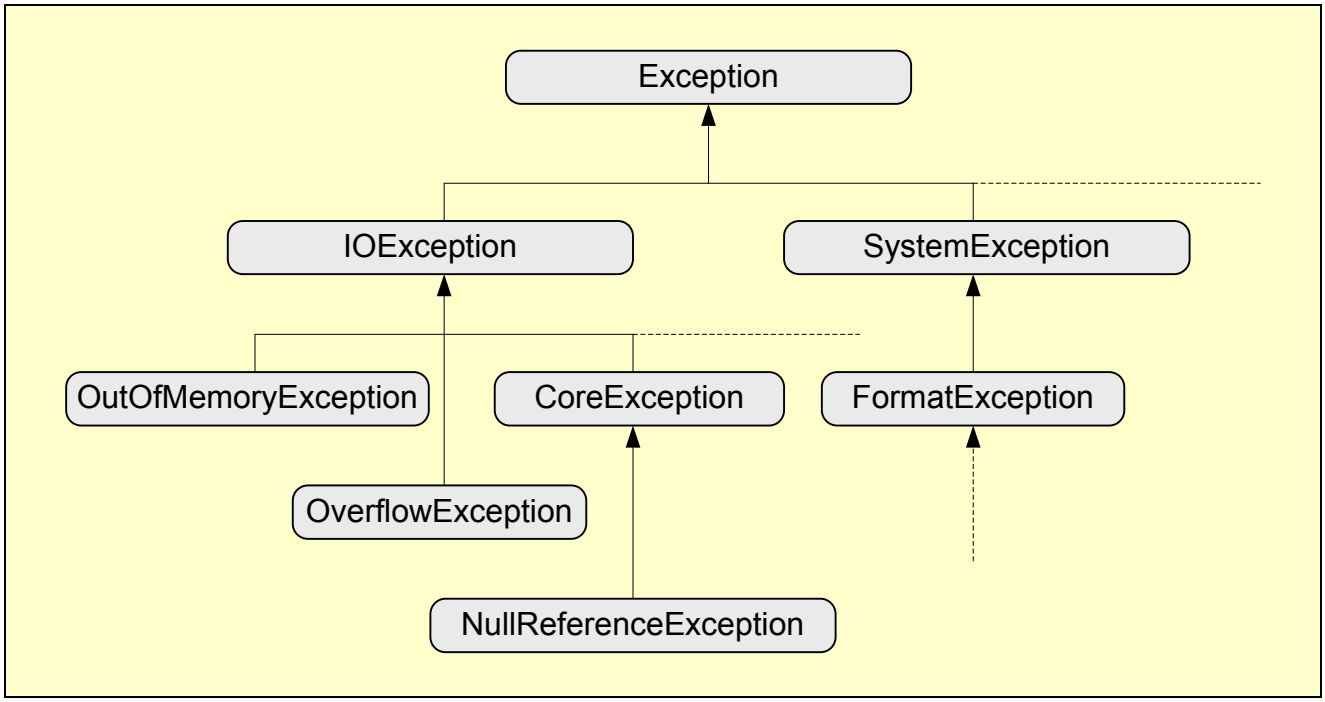
Hasta ahora capturamos una excepción que lanzo un método encerrado en un bloque Try…Catch. ¿Qué paso? ¿Qué produjo el error? ¿Qué le informamos al usuario? Existe más información que se puede manejar en el bloque Catch.

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 ámbito
     //del bloque Catch.
    catch(FormatException fe)
    {
        MessageBox.Show(fe.Message);
    }
}
```

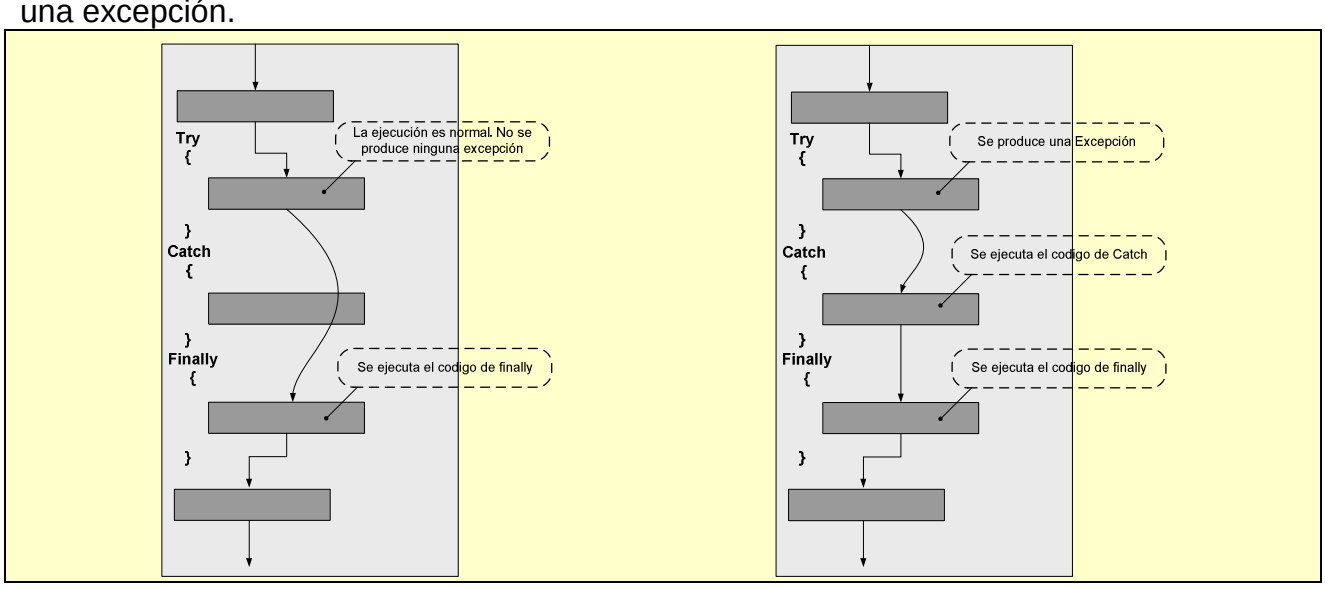


Ahora cuando el método ToInt32 lance un error, no mostrara un mensaje genérico, si no algo mas detallado de lo que produjo el errarlas clases de Excepciones en .Net descienden de la clase Excepción, la cual ofrece una cantidad de propiedades y métodos que nos ayudaran en la obtención de mayor información y en el tratamiento de lo que genero las excepciones. Algunas de las clases que descienden de Excepción son:



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

También se puede especificar un bloque finally el cual se ejecutara ocurra o no una excepción.



Reglas de los Bloques Catch

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

CONTROLES LISTBOX Y CHECKEDLISTBOX

Los cuadros de 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 básica 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 índice seleccionado de la lista. Si es una selección múltiple, devuelve el primer índice seleccionado.*
ColumnWidth	*Especifica el ancho de columna en un cuadro de lista con varias columnas.*
Ítems	*Devuelve la colección de ítems 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 más de una columna, esta propiedad especifica si el valor se muestra en cantidades de columnas.*
SelectedIndices	*Devuelve una colección con todos los índices 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 existen 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 índices de los elementos del CheckedListBox que tengan un estado chequeado.*
CheckedItems	*Devuelve una colección con todos los elementos del Checked-ListBox 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.*

métodos comunes para ListBox y CheckedListBox:

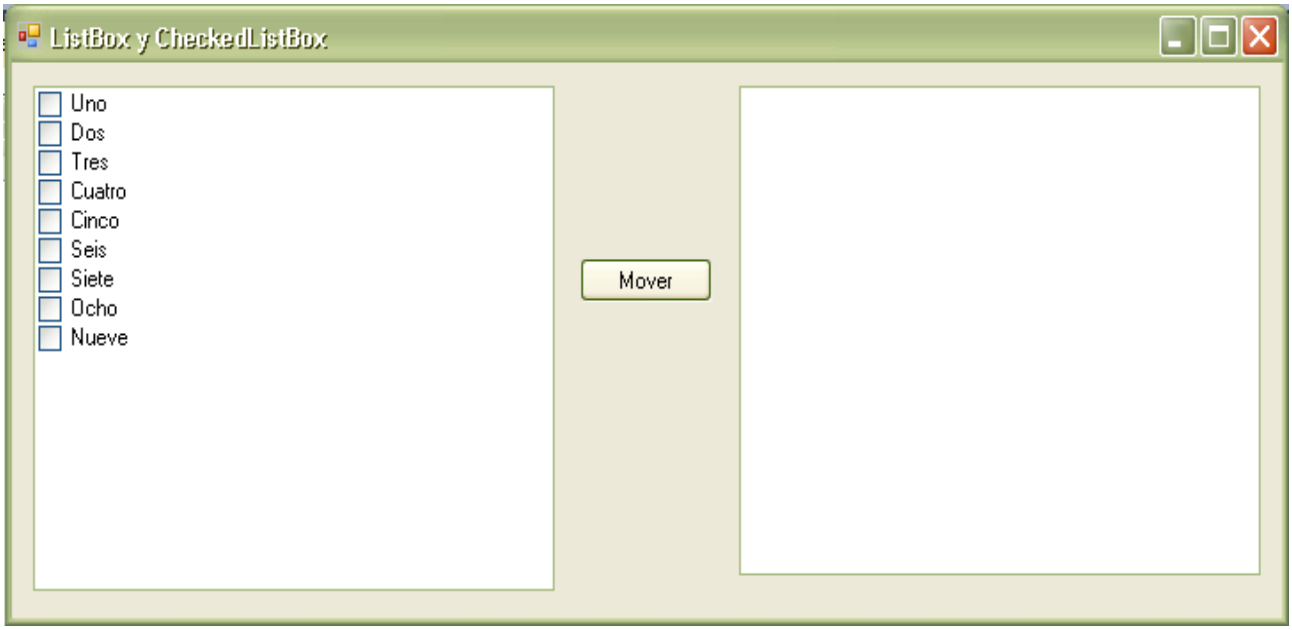
METODOS	DESCRIPCIÓN
ClearSelected()	*Limpia toda la selección del ListBox.*
FindString()	*Busca en el ListBox la primer cadena que en el comienzo coinci-da con la cadena que se especifica en este método.*
FindStringExact()	*Igual a la anterior pero la búsqueda es exacta.*
GetSelected()	*Devuelve un valor que indica si esta seleccionado.*
SetSelected()	*Establece o Borra la selección de un elemento.*
ToString()	*Devuelve el Ítem seleccionado en formato String.*
GetItemChecked()	*(Solo CheckedListBox) Devuelve un valor si esta seleccionado.*
GetItemCheckState()	*(Solo CheckedListBox) Devuelve un valor que indica el estado de activación del Ítem.*
SetItemChecked()	*(Solo CheckedListBox) Establece el elemento seleccionado a un estado seleccionado.*
SetItemCheckState()	*(Solo CheckedListBox) Establece el estado de activación 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 índices de los elemen-tos seleccionados.*

PRACTICA 8 (LISTBOX Y CHECKEDLISTBOX)

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



1. Cambiar las Propiedades Name de todos los controles.

2. Cambiar la Propiedad Text del Formulario y Botón.

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

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

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

```
public frmListBoxCheckedListBox()
 {

      InitializeComponent();

     //Con el método Add agregamos un ultimo Ítem a la
     //Colección de Ítems. Como parámetro la cadena de caracteres.
     clbValores.Items.Add("Diez");
 }
```

6. Agregar el evento Click del Botón Mover, para colocar el código que mueve al ListBox los Ítems seleccionados en el CheckedListBox.

```
        private void btnMover_Click(object sender, EventArgs e)
        {
            //Pregunta si la cantidad (Count) de elementos chequeados
            //en la colección de ítems chequeados(CheckedItems) es mayor a cero.
            if (clbValores.CheckedItems.Count > 0)
            {
                //Si es mayor a cero. Limpia los Ítem en el ListBox.
                lbSeleccionados.Items.Clear();

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

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

                    //Con el Método SetItemChecked, establezco en falso la
                    //casilla de verificación (No esta seleccionado).
                    //Como parámetros i-El índice y el valor de estado en este caso fal-
se.
                    clbValores.SetItemChecked(i, false);
            }
```

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