Herramientas de Programación 1

Tema Nº3:

Manejo de Controles parte 2:

Indicador de logro Nº3:Utiliza los formularios y realiza controles comunes en la aplicación de Windows manipulando propiedades y eventos.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº3:**

DISEÑO DE FORMULARIOS Y MANEJO DE CONTROLES BÁSICOS (PARTE 2)

Control: CHECKBOX

Representa un control de dos estados que permite a los usuarios marcarlo como seleccionado o deseleccionado. Este control puede ser útil para recolectar información que responde a preguntas simples del estilo verdadero/falso como "¿quieres/aceptas/rechazas esto?".

Propiedades

* Las propiedades más comunes de una casilla de verificación son:

|  |  |
| --- | --- |
| Propiedad | Descripción |
| [Checked](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.checkbox.checked?view=netframework-4.8#System_Windows_Forms_CheckBox_Checked) | Obtiene o establece un valor que indica si [CheckBox](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.checkbox?view=netframework-4.8) está en el estado activado. |
| [Enabled](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.control.enabled?view=netframework-4.8#System_Windows_Forms_Control_Enabled) | Obtiene o establece un valor que indica si el control puede responder a la interacción del usuario. |
| Name | Obtiene o establece el nombre del control. |
| Text | Obtiene o establece el texto asociado al control. |
| Visible | Obtiene o establece un valor que indica si se muestran el control y todos sus controles secundarios. |

Eventos

* El evento más común de una casilla de verificación es:

|  |  |
| --- | --- |
| Evento | Descripción |
| CheckedChanged | Se produce cuando cambia el valor de la propiedad. |

Control: RADIOBUTTON

* Los RadioButtons proporcionan una interfaz adecuada para seleccionar una opción de una serie de opciones.

Propiedades

* Las propiedades más comunes de un RadioButton son:

|  |  |
| --- | --- |
| Propiedad | Descripción |
| [Checked](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.checkbox.checked?view=netframework-4.8#System_Windows_Forms_CheckBox_Checked) | Obtiene o establece un valor que indica si [CheckBox](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.checkbox?view=netframework-4.8) está en el estado activado. |
| Name | Obtiene o establece el nombre del control. |
| TabIndex | Obtiene o establece el orden de tabulación del control en su contenedor. |
| Text | Obtiene o establece el texto asociado al control. |

Eventos

* El evento más común de un radiobutton es:

|  |  |
| --- | --- |
| Evento | Descripción |
| CheckedChanged | Se produce cuando cambia el valor de la propiedad. |

Control: COMBOBOX

* El control ComboBox es una combinación entre una lista desplegable y un cuadro de texto, que permite al usuario introducir cualquier valor directamente en el teclado o, alternativamente, seleccionar un valor de la lista.

Propiedades

* Las propiedades más comunes de un ComboBox son:

|  |  |
| --- | --- |
| Propiedad | Descripción |
| DropDownStyle | Obtiene o establece un valor que especifica el estilo del cuadro combinado |
| Name | Obtiene o establece el nombre del control. |
| [SelectedIndex](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.combobox.selectedindex?view=netframework-4.8#System_Windows_Forms_ComboBox_SelectedIndex) | Obtiene o establece el índice que especifica el elemento seleccionado actualmente. |
| TabIndex | Obtiene o establece el orden de tabulación del control en su contenedor. |
| Text | Obtiene o establece el texto asociado al control. |

Eventos

* El evento más común de un ComboBox es:

|  |  |
| --- | --- |
| Evento | Descripción |
| SelectedIndexChanged | Se produce cuando ha cambiado la propiedad [SelectedIndex](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.combobox.selectedindex?view=netframework-4.8#System_Windows_Forms_ComboBox_SelectedIndex). |

Control: LISTBOX

* ListBox un control en el que se puede mostrar varios ítems (registros o líneas), teniendo uno o varios de ellos seleccionado(s). Si en la lista hay más registros de los que se pueden mostrar al mismo tiempo, se añade automáticamente un ScrollBar.

Propiedades

* Las propiedades más comunes de un ListBox son:

|  |  |
| --- | --- |
| Propiedad | Descripción |
| Name | Obtiene o establece el nombre del control. |
| [SelectedIndex](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.combobox.selectedindex?view=netframework-4.8#System_Windows_Forms_ComboBox_SelectedIndex) | Obtiene o establece el índice que especifica el elemento seleccionado actualmente. |
| TabIndex | Obtiene o establece el orden de tabulación del control en su contenedor. |
| Text | Obtiene o establece el texto asociado al control. |

Eventos

* El evento más común de un ComboBox es:

|  |  |
| --- | --- |
| Evento | Descripción |
| SelectedIndexChanged | Se produce cuando ha cambiado la propiedad [SelectedIndex](https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.combobox.selectedindex?view=netframework-4.8#System_Windows_Forms_ComboBox_SelectedIndex). |

**Ejemplos:**

**RECURSOS**

1. Hardware

* Una computadora con 4 GB e RAM y procesador de 2.0 GHz.

1. Software

* Visual Studio Professional 2015 / 2017

**PROCEDIMIENTO**

**Laboratorio 3:**

3.1 Trabajar con el control CheckBox.

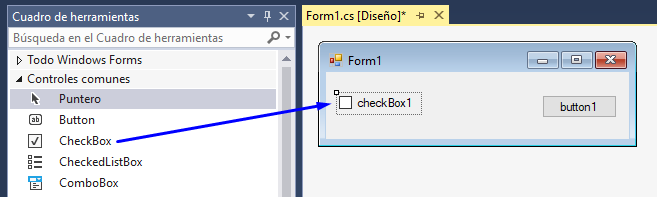
**Objetivos**

* Desarrollar aplicaciones utilizando el control CheckBox, modificar sus propiedades y entender sus eventos.

**Introducción**

Durante esta actividad, entenderá el uso del control CheckBox en los formularios de Windows.

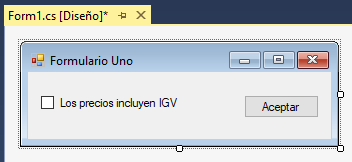
Paso 1: Creamos el proyecto *Semana03* agregamos un nuevo formulario. En este formulario agregamos una casilla de verificación y un botón.



1. Cambiamos las propiedades de los controles de acuerdo a:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| Form1 | (Name) | | frmUno |
|  | Text | | Formulario Uno |
| CheckBox1 | (Name) | | chkIgv |
|  | Text | | Los precios incluyen IGV |
| Button1 | (Name) | | btnAceptar |
|  | Text | | Aceptar |

1. El diseño debe quedar de esta forma:



Paso 2: Para este ejemplo vamos hacer uso de la estructura condicional if.

Esta estructura evalúa una condición y de acuerdo a ésta ejecuta una u otra secuencia de código.

1. Agregamos código al evento Click del botón btnAceptar de acuerdo a:

private void btnAceptar\_Click(object sender, EventArgs e)

{

if(chkIgv.Checked)

{

MessageBox.Show("Los precios SI incluyen IGV","Factura");

}

else

{

MessageBox.Show("Los precios NO incluyen IGV", "Factura");

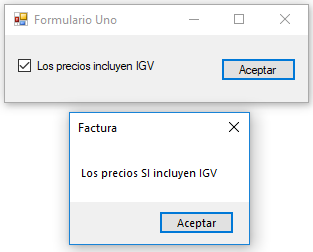
}

}

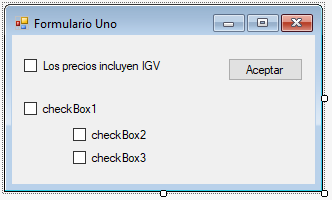
La propiedad Checked de una casilla de verificación puede tomar dos valores: verdadero (true) o falso (false), es verdadero cuando la casilla está seleccionada y falso cuando no está seleccionada.

Entonces, si la casilla está seleccionada se ejecutará el bloque de sentencias if, caso contrario se ejecutará el bloque de sentencias else.

1. Ejecutamos la aplicación.



Paso 3: Para el siguiente ejemplo vamos agregar más casilla de verificación de acuerdo al siguiente diseño:



1. Configuramos las siguientes propiedades:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| CheckBox1 | (Name) | | chkTerminos |
|  | Text | | Acepto los términos y condiciones |
| CheckBox2 | (Name) | | chkCorreos |
|  | Text | | Acepto enviarme ofertas a mi correo electrónico |
| CheckBox3 | (Name) | | chkPagos |
|  | Text | | Acepto descuentos mensuales de mi tarjeta de crédito |

1. Agregamos código al evento Load del formulario de acuerdo a:

private void frmUno\_Load(object sender, EventArgs e)

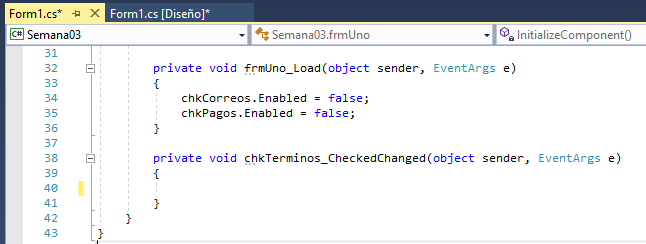
{

chkCorreos.Enabled = false;

chkPagos.Enabled = false;

}

1. Ahora vamos a agregar código al evento CheckedChanged de la casilla chkTerminos, para ello hacemos doble click en la casilla chkTerminos.



1. Modifcamos de auerdo a:

private void chkTerminos\_CheckedChanged(object sender, EventArgs e)

{

//si la casilla de verificacion pasa de inactivo a activo

if (chkTerminos.Checked)

{

chkCorreos.Enabled = true;

chkPagos.Enabled = true;

}

//si la casilla de verificacion pasa de activo a inactivo

else

{

chkCorreos.Checked = false;

chkCorreos.Enabled = false;

chkPagos.Checked = false;

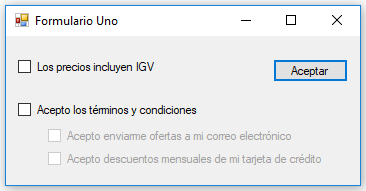
chkPagos.Enabled = false;

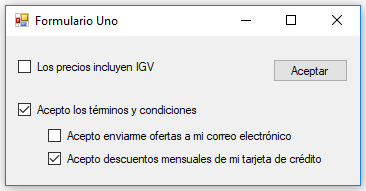
}

}

Observamos dos líneas de código que están precedidas de dos barras inclinadas (//), esto indica que la línea de código es un comentario, estas líneas no son consideradas en la ejecución de la aplicación.

1. Ejecutamos la aplicación.





**3.2 Trabajar con el control RadioButton.**

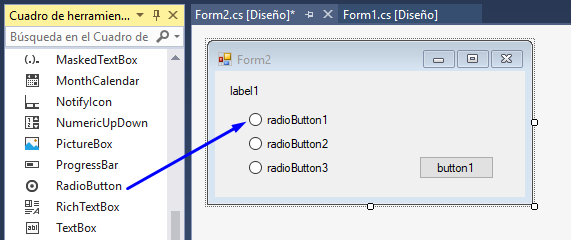
**Objetivos**

* Desarrollar aplicaciones utilizando el control RadioButton, modificar sus propiedades y entender sus eventos.

**Introducción**

Durante esta actividad, entenderá el uso del control RadioButton en los formularios de Windows.

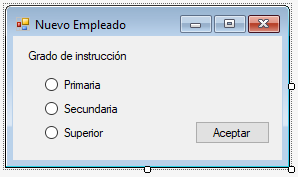
Paso 1: En el proyecto Semana03 agregamos un nuevo formulario y agregamos los controles RadioButtons de acuerdo a la siguiente figura.



1. Cambiamos las propiedades de los controles de acuerdo a:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| Form2 | (Name) | | frmDos |
|  | StartPosition | | CenterScreen |
|  | Text | | Nuevo Empleado |
| Label1 | (Name) | | Label1 |
|  | Text | | Grado de instrucción |
| RadioButton1 | (Name) | | rbtPrimaria |
|  | Text | | Primaria |
| RadioButton2 | (Name) | | rbtSecundaria |
|  | Text | | Secundaria |
| RadioButton3 | (Name) | | rbtSuperior |
|  | Text | | Superior |
| Button1 | (Name) | | btnAceptar |
|  | Text | | Aceptar |

1. El diseño debe quedar de esta forma:



Paso 2: Agregamos código en el evento Load del formulario de acuerdo a:

private void frmDos\_Load(object sender, EventArgs e)

{

rbtPrimaria.Checked = true;

}

1. Agregamos código en el evento CheckedChanged del RadioButton rbtSuperior de acuerdo a:

(hacer doble click en el RadioButton para agregar código en el evento CheckedChanged)

private void rbtSuperior\_CheckedChanged(object sender, EventArgs e)

{

//si el radiobutton pasa de inactivo a activo

if (rbtSuperior.Checked)

{

MessageBox.Show("Seleccionó la opción 'Superior'.", "Opción");

}

}

1. Agregamos código en el evento Click del botón de acuerdo a:

private void btnAceptar\_Click(object sender, EventArgs e)

{

if (rbtPrimaria.Checked)

{

MessageBox.Show("Seleccionó la opción 'Primaria'.", "Opción");

}

if (rbtSecundaria.Checked)

{

MessageBox.Show("Seleccionó la opción 'Secundaria'.", "Opción");

}

if (rbtSuperior.Checked)

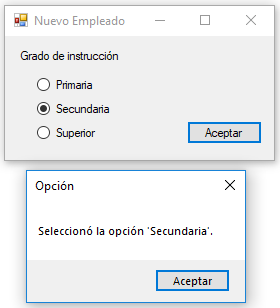
{

MessageBox.Show("Seleccionó la opción 'Superior'.", "Opción");

}

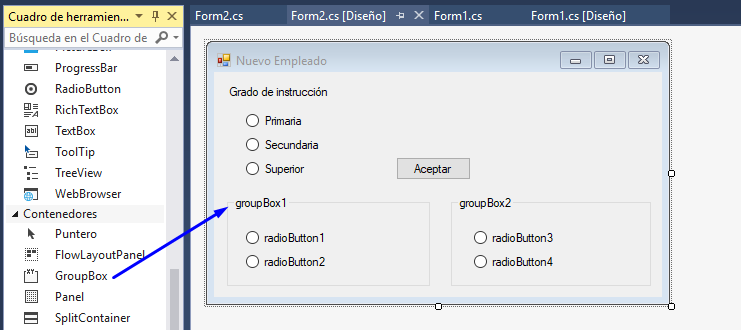
}

1. Modificamos el archivo Program.cs para que al ejecutar se muestre el formulario frmDos.
2. Ejecutamos la aplicación.



Paso 3: Los RadioButton deben estar agrupados en contenedores (controles tipo contenedor), en el ejemplo anterior el contenedor es el formulario. Existen vario controles de tipo contendor, en nuestro ejemplos usaremos el control GroupBox.

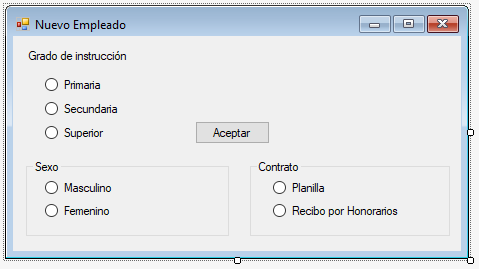
1. Modificamos el diseño del formulario frmDos de acuerdo a:



1. Cambiamos las propiedades de los controles de acuerdo a:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| GroupBox1 | (Name) | | GroupBox1 |
|  | Text | | Sexo |
| GroupBox2 | (Name) | | GroupBox2 |
|  | Text | | Contrato |
| RadioButton1 | (Name) | | rbtMasculino |
|  | Text | | Masculino |
| RadioButton2 | (Name) | | rbtFemenino |
|  | Text | | Femenino |
| RadioButton3 | (Name) | | rbtPlanilla |
|  | Text | | Planilla |
| RadioButton4 | (Name) | | rbtRecibo |
|  | Text | | Recibo por Honorarios |

1. El diseño debe quedar de esta forma:



Paso 4: Agregamos código en el evento Load del formulario de acuerdo a:

private void frmDos\_Load(object sender, EventArgs e)

{

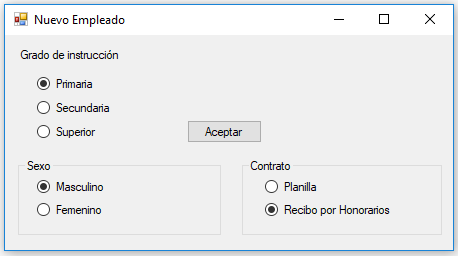
rbtPrimaria.Checked = true;

rbtFemenino.Checked = true;

rbtPlanilla.Checked = true;

}

1. Ejecutamos la aplicación.



1. Observamos que cada grupo de radiobuttons tiene dominio dentro de su contenedor.

3.3 Trabajar con el control ComboBox.

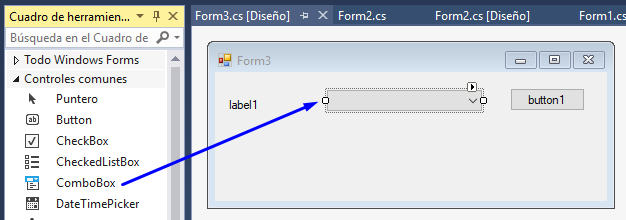
Objetivos

* Desarrollar aplicaciones utilizando el control ComboBox, modificar sus propiedades y entender sus eventos.

Introducción

Durante esta actividad, entenderá el uso del control ComboBox en los formularios de Windows.

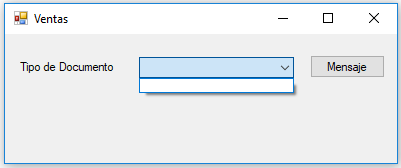
Paso 1: Creamos un nuevo formulario con el siguiente diseño:



1. Cambiamos las siguientes propiedades:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| Form3 | (Name) | | frmTres |
|  | Text | | Ventas |
| Label1 | (Name) | | Label1 |
|  | Text | | Tipo de Documento: |
| ComboBox1 | (Name) | | cmbTipoDocumento |
|  | DropDownStyle | | DropDownList |
| Button1 | (Name) | | btnMensaje |
|  | Text | | Mensaje |

1. Modificamos el archivo Program.cs para que se ejecute este último formulario.
2. Ejecutamos la aplicación.

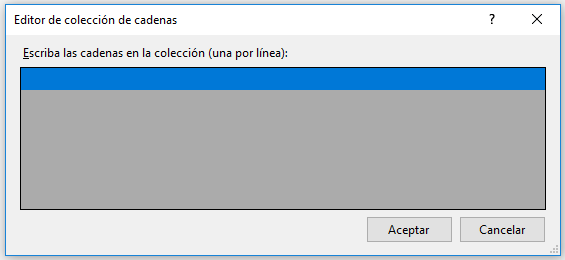


Observamos que el ComboBox no tiene ningún elemento en su lista. Además, no se puede ingresar texto dentro del control, esto se debe a que la propiedad DropDownStyle tiene el valor DropDownList.

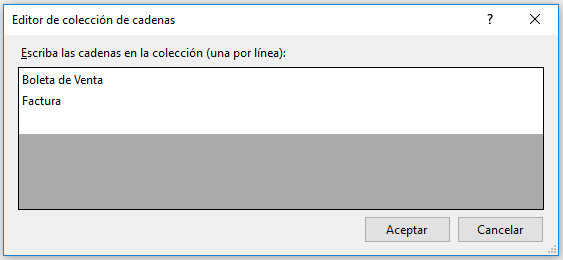
Paso 2: Para agregar elementos a la lista existen dos formas: durante el diseño del control o a través de código.

1. Primero vamos a agregar elementos a la lista del ComboBox en el diseño, para ello seleccionamos el control y en la ventana de propiedades presionamos el botón de la propiedad Items.

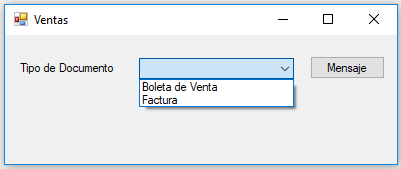




1. En esta ventana agregamos los elementos de la lista, debe quedar de la siguiente forma:



Presionamos en el botón Aceptar y ejecutamos la aplicación.



Ahora podemos verificar que el ComboBox tiene los elementos que hemos agregado en el diseño.

1. Agregamos código en el evento Click del botón cmbTipoDocumento de acuerdo a:

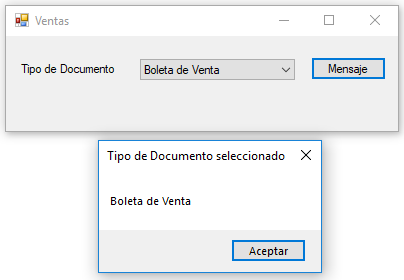
private void btnMensaje\_Click(object sender, EventArgs e)

{

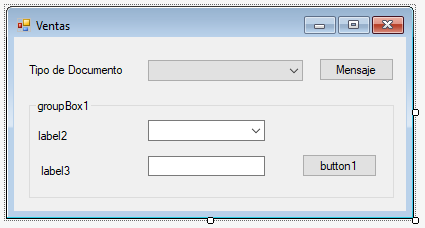
MessageBox.Show(cmbTipoDocumento.Text,"Tipo de Documento seleccionado");

}

1. Ejecutamos la aplicación.



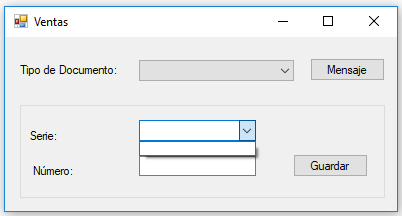
Paso 3: Ahora vamos agregar elementos de la lista a otro ComboBox a través de la edición de código, previamente agregamos controles al formulario:



1. Cambiamos las siguientes propiedades:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| GroupBox1 | (Name) | | GroupBox1 |
|  | Text | | [vacío] |
| Label1 | (Name) | | Label1 |
|  | Text | | Serie: |
| Label2 | (Name) | | Label2 |
|  | Text | | Número: |
| Button1 | (Name) | | btnGuardar |
|  | Text | | Guardar |
| ComboBox1 | (Name) | | cmbSerie |
|  | DropDownStyle | | DropDown |
| TextBox1 | (Name) | | txbNumero |

1. Ejecutamos la aplicación.



Paso 4: Los elementos del ComboBox cmbSerie van a variar de acuerdo al Tipo de Documento que se seleccione.

1. Para ello vamos agregar el código siguiente en el evento SelectedIndexChanged del ComboBox cmbTipoDocumento

private void cmbTipoDocumento\_SelectedIndexChanged(object sender, EventArgs e)

{

groupBox1.Text = cmbTipoDocumento.Text;

//Quitamos los elementos del ComboBox cmbSerie (si es que previamente tenía elementos)

cmbSerie.Items.Clear();

//Quitamos el texto que se muestra en el ComboBox (si es que previamente muestra algún texto)

cmbSerie.Text = "";

//Si se selecciona "Boleta de Venta"

if (cmbTipoDocumento.Text.Equals("Boleta de Venta"))

{

//Agregamos los elementos de la lista

cmbSerie.Items.Add("BV01");

cmbSerie.Items.Add("BV02");

cmbSerie.Items.Add("BV03");

}

//Si se selecciona "Factura"

if (cmbTipoDocumento.Text.Equals("Factura"))

{

cmbSerie.Items.Add("F111");

cmbSerie.Items.Add("F222");

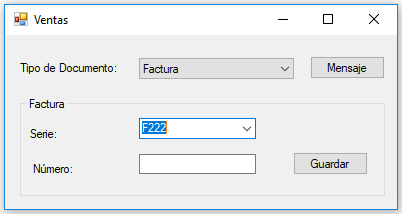
cmbSerie.Items.Add("F333");

}

}

La función Equals valida si el argumento de la función tiene el mismo texo que el texto del ComboBox.

1. Ejecutamos la aplicación.



Comprobamos que se cumple la funcionalidad requerida. Observamos también que se puede ingresar texto dentro del control, esto se debe a que la propiedad DropDownStyle tiene el valor DropDown.

Paso 5: Ahora vamos a agregar una validación básica de datos cuando se presione el botón Guardar.

private void btnGuardar\_Click(object sender, EventArgs e)

{

if(cmbSerie.Text.Length < 1)

{

MessageBox.Show("El campo 'Serie' es obligatorio", "Error");

}

if (txbNumero.Text.Length < 1)

{

MessageBox.Show("El campo 'Número' es obligatorio", "Error");

}

if (cmbSerie.Text.Length > 0 && txbNumero.Text.Length > 0)

{

MessageBox.Show("Datos guardados correctamente.", "Ok");

}

}

La propiedad Length devuelve la longitud de una cadena de texto, si el texto esta vacio devolverá cero (0).

El operador && verifca que se cumplan las dos comparaciones dentro de la sentencia If. En capítulos posteriores trataremos a detalle estos operadores.

1. Ejecutamos la aplicación.



3.4 Trabajar con el control ListBox.

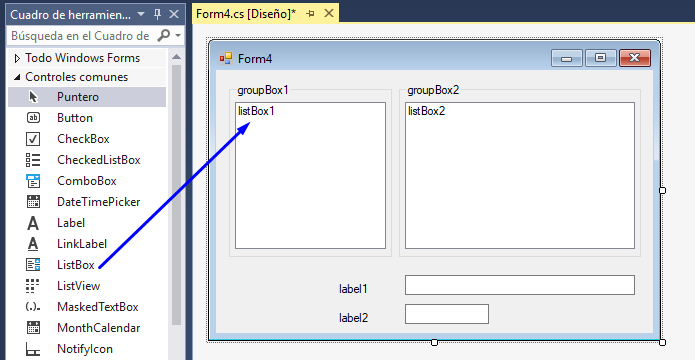
Objetivos

* Desarrollar aplicaciones utilizando el control ListBox, modificar sus propiedades y entender sus eventos.

Introducción

Durante esta actividad, entenderá el uso del control ListBox en los formularios de Windows.

Paso 1: Creamos un nuevo formulario con el siguiente diseño:

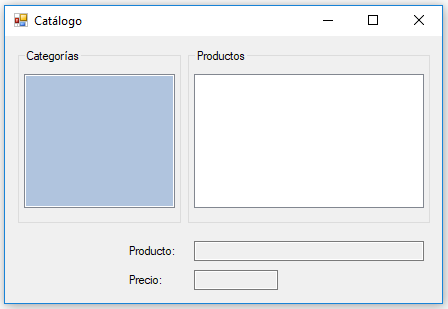


1. Cambiamos las siguientes propiedades:

|  |  |  |  |
| --- | --- | --- | --- |
| Control | Propiedad | Valor | |
| Form4 | (Name) | | frmCuatro |
|  | Text | | Catálogo |
| GroupBox1 | (Name) | | GroupBox1 |
|  | Text | | Categorías |
| GroupBox2 | (Name) | | GroupBox2 |
|  | Text | | Productos |
| ListBox1 | (Name) | | lsbCategorias |
|  | BackColor | | LightSteelBlue |
|  | Sorted | | True |
| ListBox2 | (Name) | | lsbProductos |
|  | HorizontalScrollbar | | True |
| Label1 | (Name) | | Label1 |
|  | Text | | Producto: |
| Label2 | (Name) | | Label2 |
|  | Text | | Precio: |
| TextBox1 | (Name) | | txbProducto |
|  | ReadOnly | | True |
| TextBox2 | (Name) | | txbPrecio |
|  | ReadOnly | | True |
|  | RightToLeft | | Yes |

Modificamos el archivo Program.cs para que se ejecute este último formulario.

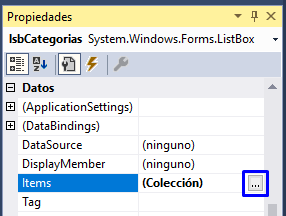
1. Ejecutamos la aplicación.



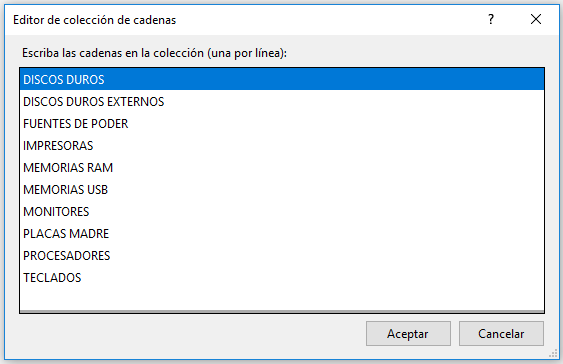
Observamos que los ListBox no tienen ningún elemento en su lista.

Paso2: Para agregar elementos a la lista existen dos formas: durante el diseño del control o a través de código.

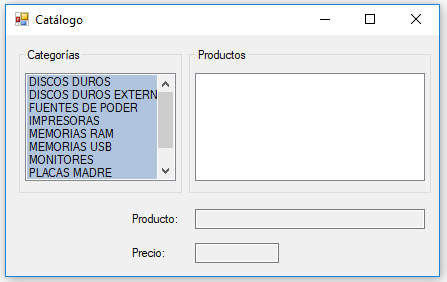
1. Primero vamos a agregar elementos a la lista del ListBox lsbCategorias en el diseño, para ello seleccionamos el control y en la ventana de propiedades presionamos el botón de la propiedad Items.



En esta ventana agregamos los elementos de la lista, debe quedar de la siguiente forma:



1. Presionamos en el botón Aceptar y ejecutamos la aplicación.



Ahora podemos verificar que el ListBox lsbCategorias tiene los elementos que hemos agregado en el diseño pero de forma ordenada, esto debido a que la propiedad Sorted tiene el valor True.

Cuando la cantidad de elementos supera el tamaño vertical del ListBox automáticamente se muestra el scroll vertical, no necesita una configuración especial para que tenga este comportamiento.

Paso 3: Ahora vamos agregar elementos al ListBox lsbProductos a través de la edición de código, los elementos de este ListBox van a variar de acuerdo a la Categoría que se seleccione.

1. Para ello vamos agregar el código siguiente en el evento SelectedIndexChanged del ListBox clbxCategorias.

private void lsbCategorias\_SelectedIndexChanged(object sender, EventArgs e)

{

txbProducto.Text = "";

txbPrecio.Text = "";

lsbProductos.Items.Clear();

if (lsbCategorias.Text.Equals("DISCOS DUROS"))

{

lsbProductos.Items.Add("DISCO DURO TOSHIBA 500GB P300 PARA PC");

lsbProductos.Items.Add("DISCO DURO TOSHIBA 1TB P300 PARA PC");

lsbProductos.Items.Add("DISCO DURO TOSHIBA 2TB P300 PARA PC");

lsbProductos.Items.Add("DISCO DURO PARA LAPTOP 500GB TOSHIBA L200");

lsbProductos.Items.Add("DISCO DURO PARA LAPTOP 1TB TOSHIBA L200");

lsbProductos.Items.Add("DISCO DURO SATA 1TB WD PURPLE");

lsbProductos.Items.Add("DISCO DURO SATA 2TB WD PURPLE");

lsbProductos.Items.Add("DISCO DURO SATA 4TB WD PURPLE");

}

if (lsbCategorias.Text.Equals("DISCOS DUROS EXTERNOS"))

{

lsbProductos.Items.Add("DISCO EXTERNO CANVIO 1TB TOSHIBA USB 3.0.");

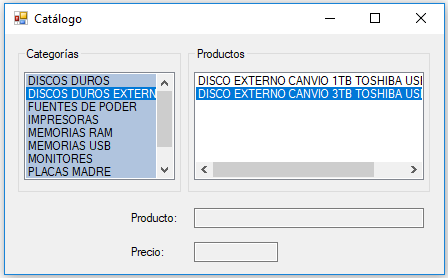
lsbProductos.Items.Add("DISCO EXTERNO CANVIO 3TB TOSHIBA USB 3.0");

}

}

La función Equals valida si el argumento de la función tiene el mismo texo que el texto seleccionado del ListBox.

1. Ejecutamos la aplicación.



Comprobamos que se cumple la funcionalidad requerida.

Paso 4: Ahora vamos agregar el código siguiente en el evento SelectedIndexChanged del ListBox clbxProductos. El objeto es mostrar en los TextBox el nombre del producto seleccionado y su precio.

private void lsbProductos\_SelectedIndexChanged(object sender, EventArgs e)

{

txbProducto.Text = lsbProductos.Text;

//También podemos usar:

//txbProducto.Text = lsbProductos.SelectedItem.ToString();

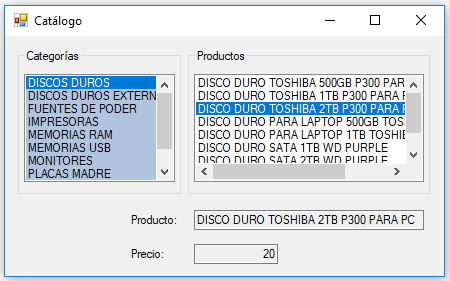
txbPrecio.Text = Convert.ToString(lsbProductos.SelectedIndex \* 10);

}

La propiedad SelectedIndex devuelve el índice del elemento seleccionado, el indice del primer elemento de la lista es cero (0) el índice del segundo elemento es uno (1) y asi sucesivamente.

El precio se calcula multiplicando el índice del producto seleccionado por 10, el precio calculado es un dato de tipo entero, sin embargo para mostrarlo en un TextBox necesitamos convertirlo a un dato de tipo texto, este es el objetivo de utilizar la función de conversión Convert.ToString().

1. Ejecutamos la aplicación.



**Actividad:**

Ingresa a la plataforma virtual, revisar los siguientes enlaces, luego desarrollar las actividades propuestas:

* Creación de un proyecto Windows Forms en Visual Studio con C#

<https://www.youtube.com/watch?v=AN79L5B7a58&list=PLM>p96nOrGcZezSAPW08ulofVcEZRpYfH&index=1

* Primera aplicación Windows Forms en Visual Studio con C#

https://www.youtube.com/watch?v=B2DLZAXuFKY&list=PLMp96nOrGcZezSAPW08ulofVcEZRpYfH&index=2

* Lenguaje C#

<https://www.youtube.com/watch?v=6EBNIgkrU74&list=PLU8oAlHdN5BmpIQGDSHo5e1r4ZYWQ8m4B&index=1>

1. Crear una solución con tres proyectos: una como Windows Forms las otras dos como Biblioteca de Clases.
2. Explicar otros dos tipos de proyectos que se pueden crear con Visual Studio.