

Facultad: Ingeniería
Escuela: Computación
Asignatura: Programación Orientada a objetos

GUIA 7:

Validaciones en C#

Materiales y Equipo

Nº	Cantidad	Descripción
1	1	Guía de Laboratorio #7 de Programación Orientada a Objetos
2	1	Computadora con programa: ➤ Microsoft Visual C#
3	1	Dispositivo de memoria externo

Introducción

La mayoría de las veces nuestras aplicaciones fallan porque los datos que reciben y que intentan tratar no son los correctos y no se ha comprobado previamente.

En principio la validación debe hacerse control a control, ya que para cada uno las validaciones pueden ser diferentes.

Si queremos validar datos que sean obligatorios de ingresar en el lenguaje de programación C#, podemos utilizar el control `ErrorProvider`.

ErrorProvider: Interfaz de Usuario para indicar al usuario que el control de un formulario tiene un error asociado.

La instrucción `try-catch` consta de un bloque `try` seguido de una o más cláusulas `catch`, las cuales especifican controladores para diferentes excepciones.

Las expresiones regulares son un lenguaje que permite simbolizar conjuntos de cadenas de texto formadas por la concatenación de otras cadenas. Es decir, permite buscar subcadenas de texto dentro de una cadena de texto.

La definición de un patrón de búsqueda de expresión regular se establece a través de un

intrincado conjunto de axiomas de tipo matemático.

La idea más importante es que una expresión regular es un patrón de búsqueda en una cadena, es algo parecido a los caracteres comodines del sistema operativo.

Pero la mejor forma de tratar cualquier posible error que provenga de los datos ingresados desde el usuario será el no permitir que se equivoque, es decir utilizar una serie de herramientas que faciliten la manipulación de datos y que no den pie a error:

RadioButton

ComboBox

DateTimePicker

MonthCalendar

CheckBox

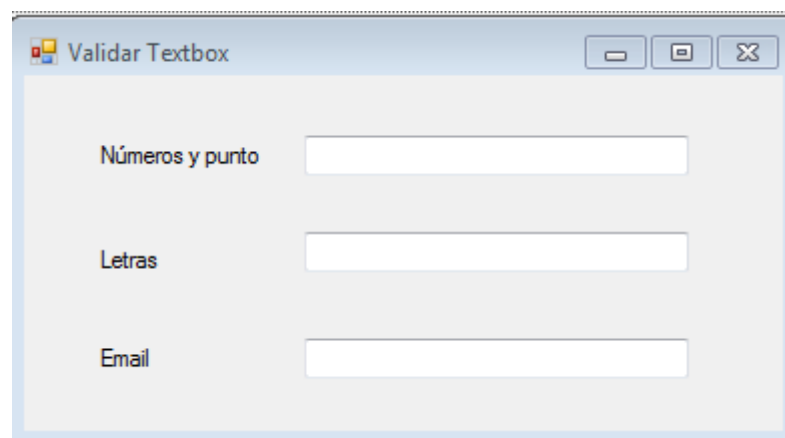
Entre otros.

Procedimiento

EJEMPLO 1:

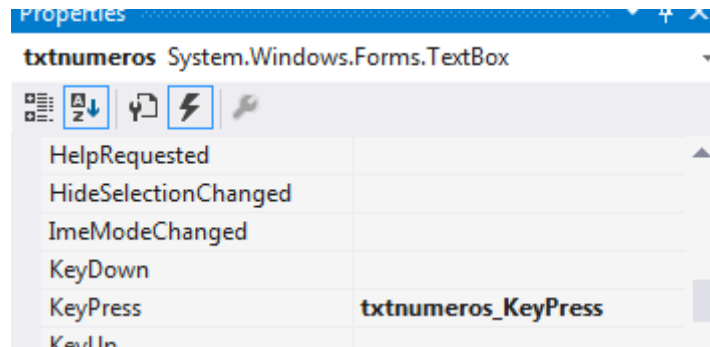
El primer ejercicio nos permitirá validar tres TextBox de forma que únicamente puedan ingresarse el tipo de información que nosotros deseamos. Para ello nos valdremos de otros eventos y también de las expresiones regulares.

1. Cree un formulario que luzca de la siguiente manera



Se han ocupado únicamente tres label y tres textbox

2. Ahora lo que haremos será programar los textbox correspondientes, en el primer caso el textbox para números, al cual se le ha dado el nombre de **txtnumeros**, buscaremos en el listado de eventos el evento **KeyPress**



3. Programaremos el código correspondiente

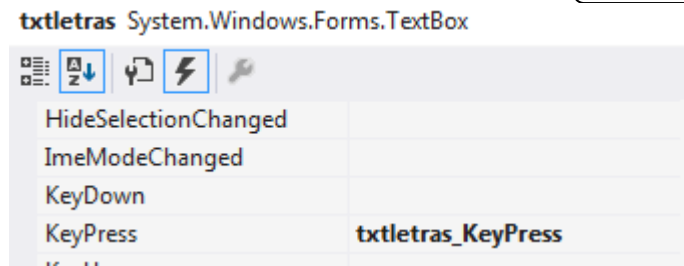
```
private void txtnumeros_KeyPress(object sender, KeyPressEventArgs e)
{
    //condicion para solo números
    if(char.IsDigit(e.KeyChar))
    {
        e.Handled = false;
    }

    //para tecla backspace
    else if(char.IsControl(e.KeyChar))
    {
        e.Handled = false;
    }

    /*verifica que pueda ingresar punto y también que solo pueda
    ingresar un punto*/
    else if((e.KeyChar=='.')&&(!txtnumeros.Text.Contains(".")))
    {
        e.Handled = false;
    }

    //si no se cumple nada de lo anterior entonces que no lo deje pasar
    else
    {
        e.Handled = true;
        MessageBox.Show("Solo se admiten datos numéricos", "validación de
números", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
```

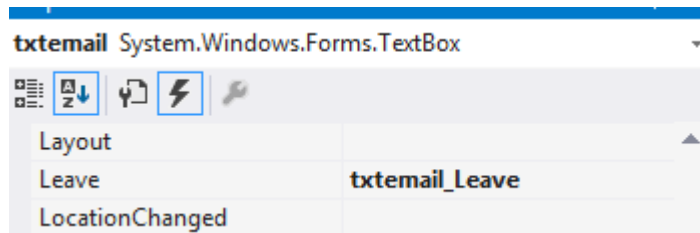
4. Ahora es el turno del textbox para letras, al cual se le ha dado el nombre de **txtletras**, buscaremos en el listado de eventos el evento **KeyPress**



5. Programaremos el código para este evento

```
private void txtletras_KeyPress(object sender, KeyPressEventArgs e)
{
    //condicion para solo números
    if (char.IsLetter(e.KeyChar))
    {
        e.Handled = false;
    }
    //para backspace
    else if (char.IsControl(e.KeyChar))
    {
        e.Handled = false;
    }
    //para que admita tecla de espacio
    else if (char.IsSeparator(e.KeyChar))
    {
        e.Handled = false;
    }
    //si no cumple nada de lo anterior que no lo deje pasar
    else
    {
        e.Handled = true;
        MessageBox.Show("Solo se admiten letras", "validación de texto",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
```

6. El último textbox que utilizaremos será el del correo electrónico nombrado txtemail, para ello cambiaremos el evento por Leave (es decir que cuando salgamos del textbox ahí hará la verificación o validación)



7. El código estará compuesto por dos partes, primero un método fuera de cualquier evento llamado **validaremail**, pero antes en las librerías del proyecto agregaremos la librería `using System.Text.RegularExpressions;`

```

public static bool validaremail(string email)
{
    //cadena o expresion regular que verifica a un formato de correo electrónico
    string expresion = "^[_a-z0-9-]+(.[_a-z0-9-]+)*@[a-z0-9-]+(.[a-z0-9-]+)*(.{2,4})$";
    //verifica que el email ingresado corresponda con la expresion válida
    if(Regex.IsMatch(email,expresion))
    {
        //verifica que la direccion corresponda y que la longitud de la cadena no
        esté vacía
        if(Regex.Replace(email,expresion,string.Empty).Length==0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}

```

8. Ahora sí programamos el evento Leave

```

private void txtemail_Leave(object sender, EventArgs e)
{
    if(validaremail(txtemail.Text))
    {
        //si es correcto no debe hacer nada
    }
    else
    {
        /*sino es correcto que envíe este mensaje y se posicione para
        verificar recuerde que se activará la validación al dar click en otro textbox o
        simplemente dejar o salir de esa casilla de email*/
        MessageBox.Show("Dirección de correo no válida");
        txtemail.SelectAll(); //selecciona todo lo de la casilla
        txtemail.Focus(); //se posiciona ahí de nuevo
    }
}

```

EJEMPLO 2:

Realizaremos un ejercicio apoyándonos de la herramienta ErrorProvider. Para ello haremos un formulario como el de la imagen.

Formulario

VALIDACIONES

Nombre:

Apellido:

Fecha de nacimiento: martes , 18 de septiembre de 2018

Edad:

GUARDAR

errorProvider1

Para el formulario se han utilizado:

5 Labels

3 Textbox (pero uno tiene la propiedad ReadOnly en TRUE, es el de edad para que no pueda escribirse sobre él)

1 DateTimePicker

1 button

1 ErrorProvider

1. Lo primero que haremos será programar un método auxiliar para las validaciones

```
private bool validarCampos()
{
    //variable que verifica si algo ha sido validado
    bool validado = true;
    if(txtnombre.Text == "") //vefica que no quede vacío el campo
    {
        validado = false; //si está vacío validado es falso
        errorProvider1.SetError(txtnombre, "Ingresar nombre"); //por lo tanto manda
a llamar a errorprovider
        //en los parámetros de setError se identifica a quién estoy validando y el
mensaje que deseo mandar
    }

    //verifico la casilla de apellido
}
```

```

        if (txtapellido.Text == "")
        {
            validado = false;
            //digo que verifico a txtapellido y si no cumple mando ese mensaje
            errorProvider1.SetError(txtapellido, "Ingreso apellido");
        }
        return validado;
    }
}

```

2. Nos apoyaremos de otro método que permite limpiar los mensajes de alerta cuando demos click a GUARDAR de nuevo, pero solo lo borrará si los datos ingresados fueron correctos

```

private void BorrarMensaje()
{
    //borra los mensajes para que no se muestren y pueda limpiar
    errorProvider1.SetError(txtnombre, "");
    errorProvider1.SetError(txtapellido, "");
}

```

3. Ahora lo que haremos será programar el botón GUARDAR porque es a partir de dar click a este botón que se harán todas las verificaciones, dentro de este botón también haremos que nos calcule la edad a partir de la fecha de nacimiento que ingresemos, el código es el siguiente:

```

private void btnguardar_Click(object sender, EventArgs e)
{
    //limpia cualquier mensaje de error de alguna corrida previa
    BorrarMensaje();
    //llamamos al método para validar campos, el de nombre y apellido
    if( validarCampos())
    {
        MessageBox.Show("Los datos se ingresaron correctamente");
    }
    //verificamos la fecha de nacimiento que nos den
    //DateTimePicker se llama dtpFechaNacimiento
    DateTime fechaNacimiento = dtpFechaNacimiento.Value;
    //verificamos la fecha del sistema (solo calculamos con los años
    int anios = System.DateTime.Now.Year - fechaNacimiento.Year;
    /*verificamos aparte del año si ya pasamos la fecha de nacimiento de este año o nos
faltan días*/
    if (System.DateTime.Now.Subtract(fechaNacimiento.AddYears(anios)).TotalDays < 0)
        //si nos faltan días para cumplir años al cálculo le resta uno
        txtedad.Text = Convert.ToString(anios - 1);
    else
        //si ya pasó nuestra fecha de nacimiento manda el valor correspondiente
        txtedad.Text = Convert.ToString(anios);
}

```

EJEMPLO 3:

Realizaremos un ejercicio utilizando la instrucción try catch para capturar todos los posibles errores que podrían ocurrir en el caso de querer dividir un número entre otro. Haremos un formulario con la siguiente apariencia

1. Procedamos a programar el botón calcular que será el verificador de la instrucción

```
private void btncalcular_Click(object sender, EventArgs e)
{
    //tratará de realizar la acción solicitada
    try
    {
        float numera = float.Parse(txtdividendo.Text);
        float denomina = float.Parse(txtdivisor.Text);

        float resultado = numera / denomina;

        txtresultado.Text = Convert.ToString(resultado);
    }
    //si no pudiera hacerlo entonces verificará cual es el error y nos los mostrará
    catch (Exception error)
    {
        MessageBox.Show("El problema es: " + error.Message);
    }
}
```

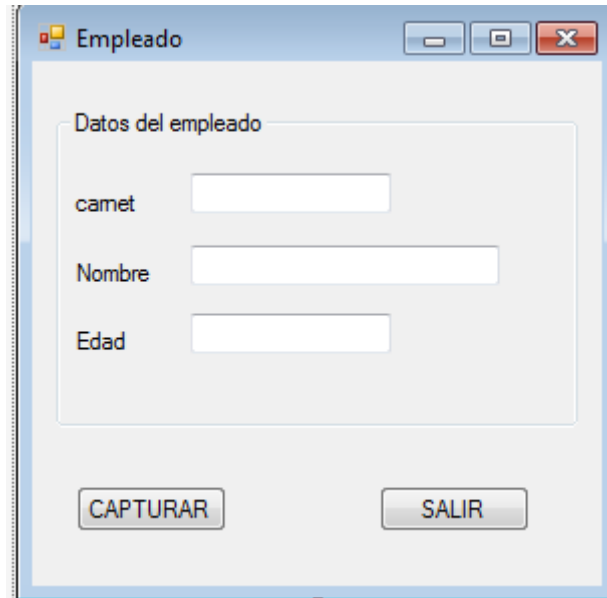
2. Haremos un botón adicional por si deseamos hacer más intentos

```
private void btnreintentar_Click(object sender, EventArgs e)
{
    txtdividendo.Clear();
    txtdivisor.Clear();
    txtdividendo.Focus();
}
```


EJEMPLO 4:

Realizaremos un ejercicio utilizando la instrucción try catch personalizando nuestros propios errores por medio de un throw new Exception y lo haremos directamente en la clase, para que sin importar desde qué formulario lo mandemos a llamar él pueda reconocer siempre todos los errores.

1. Haremos un formulario como el siguiente:



2. Ahora haremos la clase y dentro de la clase haremos las validaciones, la clase se llamará

Empleado

```
class Empleado
{
    //atributos
    private int carnet;
    private string nombre;
    private int edad;

    //propiedades con validación
    public int Carnet
    {
        get { return carnet; }
        set
        {
            carnet = value;
            if (carnet <= 0)
                throw new Exception("Dato incorrecto para carnet");
        }
    }

    public string Nombre
    {
        get { return nombre; }
```

```

        set
        {
            nombre = value;
            if (nombre == "")
                throw new Exception("No debe dejar en blanco el nombre");

            foreach (char letra in nombre)
            {
                //caracteres permitidos
                switch (letra)
                {
                    case 'Á': continue;
                    case 'É': continue;
                    case 'Í': continue;
                    case 'Ó': continue;
                    case 'Ú': continue;

                }
                if (letra < 'A' || letra > 'Z')
                {
                    throw new Exception("Solamente se permiten mayúsculas en el nombre");
                }
            }
        }

    public int Edad
    {
        get { return edad; }
        set { edad = value;
            if (edad < 0 || edad > 110)
                throw new Exception("Dato fuera del rango");
            }
    }
}

```

3. Ahora en el formulario programamos el botón CAPTURAR

```

private void btncapturar_Click(object sender, EventArgs e)
{
    Empleado miEmpleado = new Empleado();

    try
    {
        miEmpleado.Carnet = int.Parse(txtcarnet.Text);
        miEmpleado.Nombre = txtnombre.Text;
        miEmpleado.Edad = int.Parse(txtedad.Text);
    }
    catch (Exception x)
    {
        MessageBox.Show(x.Message);
        txtcarnet.Text = "";
        txtnombre.Text = "";
        txtedad.Text = "";
        txtcarnet.Focus();
        return;
    }
}

```

4. Finalmente programamos el botón SALIR

```
private void btnsalir_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Desarrollo de habilidades

G7_Ejercicio_01:

Modifique el ejemplo 2 de forma que si alguien selecciona una fecha de nacimiento del futuro (es decir mayor a la fecha actual) le mande un mensaje de advertencia que la fecha es inválida.

G7_Ejercicio_02:

Realice dos formularios en un mismo proyecto, en el primer formulario capturará los datos de ingreso para un estudiante como carnet (con formato UDB), nombres, fecha de nacimiento, correo electrónico y responsables.

En el segundo formulario podrá ingresar las notas de los tres períodos de ese estudiante para una materia en específico que haya inscrito y le enviará el cálculo del promedio de dicha asignatura.

Se le pide validar TODOS los elementos de ambos formularios de forma que el sistema sepa cómo reaccionar en caso de que el usuario ingrese un valor incorrecto en cualquiera de ellos.

Investigación complementaria

1. Investigue cómo funciona MaskedTextBox y elabore un ejemplo
2. Realice un ejemplo utilizando try-catch-finally y explique la diferencia con try-catch
3. Investigue otras cadenas para expresiones regulares comunes (teléfono, URL de sitio web u otros)

