Eventos

1. Uso de delegados y eventos

El siguiente código se corresponde con la SOLUCIÓN 1 planteada al inicio del Tema 4 de Teoría:

```
namespace ClasesAcopladas
{
    class Acopladas
    {
        static void Main(string[] args)
        {
            observador obj=new observador();
            obj.funciona();
        }
    }

    class observador
    {
        TrabajoDuro tb;
        public observador() { tb = new TrabajoDuro(this); }

        public void funciona()
        {
            Console.WriteLine("Vamos a probar el informe");
            tb.ATrabajar();
            Console.WriteLine("Terminado");
        }

        public void InformeAvance(int x)
        {
            string str = String.Format("Ya llevamos el {0}", x);
            Console.WriteLine(str);
        }
    }

    class TrabajoDuro
    {
        int PocentajeHecho;
        observador eljefe;

        public TrabajoDuro(observador o)
        {
            PocentajeHecho = 0;
            eljefe = o;
        }

        public void ATrabajar()
        {
            int i;
            for (i = 0; i < 500; i++)
            {
                System.Threading.Thread.Sleep(1);  //Hacemos el trabajo
                switch (i)
                {
                    case 125:
                        PocentajeHecho = 25;
                        eljefe.InformeAvance (PocentajeHecho);
                        break;
```

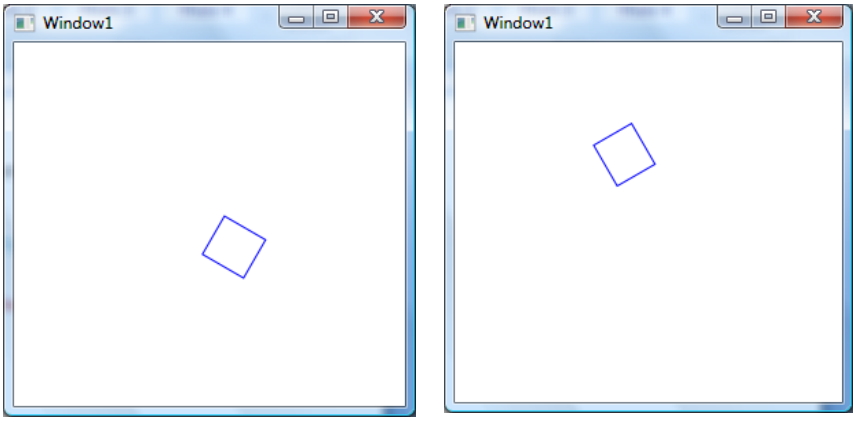
```
                    case 250:
                        PocentajeHecho = 50;
                        eljefe.InformeAvance(PocentajeHecho);
                        break;
                    case 375:
                        PocentajeHecho = 75;
                        eljefe.InformeAvance(PocentajeHecho);
                        break;
                }
            }
        }
    }
}
```

Se pide:

a) Crear un proyecto de tipo Consola y probar dicha solución
b) Crear un nuevo proyecto de tipo Consola que mejore el anterior haciendo uso de delegados. Comprobar que se pueden crear distintos métodos compatibles con un delegado y determinar en tiempo de ejecución cuál de ellos se ejecuta.
c) Crear un tercer proyecto del mismo tipo que modifique el anterior mediante el uso de eventos.

NOTA: se recomienda que los tres proyectos se incluyan en la misma solución para facilitar la comparación entre ellos.

2. Gestionando eventos de teclado y ratón



Escribir una aplicación que permita mover una figura por la pantalla usando las teclas de dirección. Además la figura girará cuando el ratón esté moviéndose por la ventana.

Se recomienda usar un grid (por ejemplo de 8x8) para establecer las posibles posiciones de la figura de modo que al pulsarse al teclas de dirección, se incremente o decremente el número de fila o columna según corresponda. Se deberán controlar los rangos posibles.

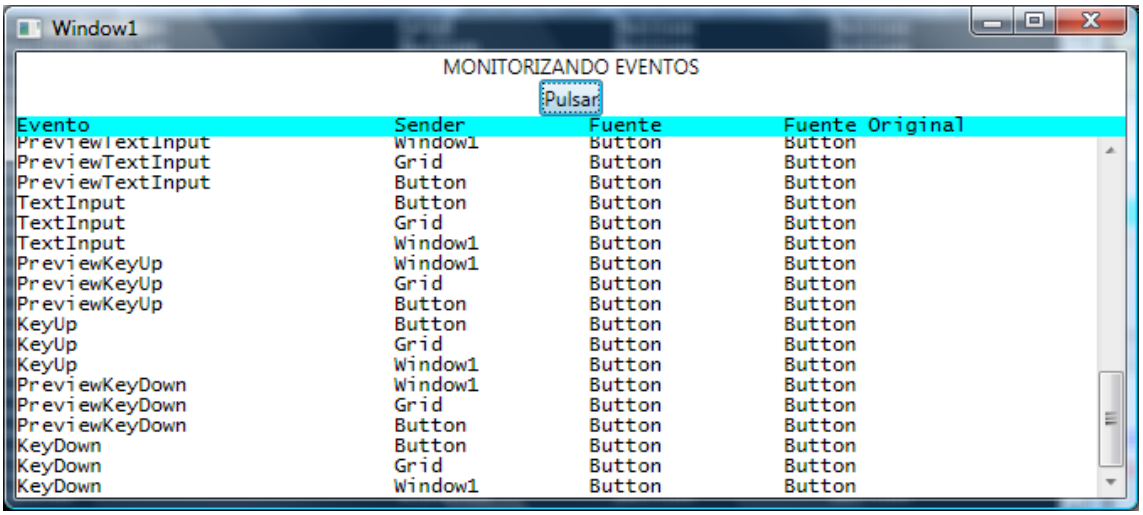
Revisar la documentación sobre los métodos estáticos `GetColumn`, `GetRow`, `SetColumn` y `SetRow` de la clase `Grid`.

3. Aplicación para monitorizar eventos enrutados

Creamos una aplicación que muestre información sobre los eventos que se producen en la propia aplicación.

Se trata de gestionar una serie de eventos mediante un único gestor de eventos cuya funcionalidad consistirá en añadir a un panel una línea por cada evento gestionado mostrando el nombre del evento, el objeto que envía el evento, el objeto fuente y la fuente original del evento. De este modo, se puede observar el funcionamiento de los eventos enrutados.

Un ejemplo de la aplicación se muestra en la siguiente imagen:



Crear la interfaz de la aplicación que consta de un grid que contiene:

1. Un texto (`TextBlock`) de título en la primera fila
2. Un botón (`Button`) en la segunda fila. El contenido del botón lo definiremos también mediante un elemento de bloque de texto
3. Otro texto (`TextBlock`) con fondo coloreado en la tercera línea (no se le dará contenido en el XAML, lo haremos luego en el código para poder darle el formato deseado).
4. Un `ScrollViewer` en la cuarta fila que contendrá un `StackPanel`

Todos los elementos tendrán su propio nombre para poder ser manejados como miembros de la ventana desde el código subyacente.

Para que el texto sea fácil de encolumnar nos interesa que la fuente sea de ancho fijo. Por tanto pondremos en la ventana un valor de “Lucida Console” para la propiedad FontFamily.

Gestor de eventos

Vamos a escribir un único gestor de eventos que será el que se ejecute para varios de los eventos enrutados que se puedan producir. Este gestor se escribirá como método de la clase derivada de Window y simplemente generará un bloque de texto cuyo texto será una string

con información extraída de los parámetros `sender` y e. Luego añadirá el bloque de texto al StackPanel y colocará el scroll en la posición de abajo.

El siguiente código es un ejemplo de cómo puede hacerse. Se hace uso de una función auxiliar, `nombreobjeto`, que recibe el nombre de los objetos precedidos de la cadena de namespaces, separa la cadena en varias string (usando los puntos como marca de separación) que almacena en el array de llamada parseada y devuelve el último elemento de la matriz que es el que corresponde al nombre del objeto sin los namespaces.

Revisar la documentación sobre el método `Format` de `string`. La cadena de formato `strFormat` indica el número de caracteres que se destinarán a cada uno de los cuatro elementos que componen la cadena.

Revisar la documentación sobre el método `GetType` de `Object` y el método `Split` de `String`

Nótese que se deben sustituir los nombres de miembro *`panel`* y *`scroll`* por los que se hayan usado al definir el `StackPanel` y el `ScrollViewer` respectivamente.

```
string strFormat = "{0,-30} {1,-15} {2,-15} {3,-15}";


void gestorglobal(Object sender, RoutedEventArgs e)
{
    TextBlock linea = new TextBlock();
    linea.Text = String.Format(strFormat,
        e.RoutedEvent.Name,
        nombreobjeto(sender),
        nombreobjeto(e.Source),
        nombreobjeto(e.OriginalSource));
    panel.Children.Add(linea);
    scroll.ScrollToBottom();
}

string nombreobjeto(Object obj)
{
    string[] parseada = obj.GetType().ToString().Split('.');
    return parseada[parseada.Length - 1];
}
```

Faltaría registrar el método `gestorglobal` como gestor de los diversos eventos. Como queremos comprobar que un mismo evento puede ser tratado por varios elementos a los largo de la ruta, vamos a registrar el gestor de eventos para los eventos asociados a los distintos elementos de nuestra interfaz: la ventana, el botón, el grid, el stackpanel, el texto de contenido del botón, … (se pueden añadir más elementos si se desea).

Para abreviar vamos a crear un array con todos estos elementos y lo recorreremos con un buche `foreach` para registrar el gestor de eventos (nótese que los elementos de la matriz deben ser los nombres de los elementos creados en el XAML). Todo ello lo incluiremos en el constructor de la ventana:

```
        UIElement[] elementos = { this, grid, boton, cabecera, panel,
textoboton };

            foreach (UIElement i in elementos)
            {
                i.PreviewKeyDown += gestorglobal;
                i.PreviewKeyUp += gestorglobal;
                i.PreviewTextInput += gestorglobal;
                i.KeyDown += gestorglobal;
                i.KeyUp += gestorglobal;
                i.TextInput += gestorglobal;

                i.PreviewMouseDown += gestorglobal;
                i.PreviewMouseUp += gestorglobal;
                i.MouseDown += gestorglobal;
                i.MouseUp += gestorglobal;
            }
            boton.Click += gestorglobal;
```

Aprovechamos también para incluir en el constructor de la ventana la inicialización del texto formateado de la cabecera (sustituir *cabecera* por el nombre que se le haya dado al elemento correspondiente):

```
            cabecera.Text = string.Format(strFormat, "Evento",
                              "Sender", "Fuente", "Fuente Original");
```

Compilar y ejecutar.

Comprobar que el enrutamiento en los eventos Preview es de tunelado, mientras que en los demás es de burbuja.

Uso de la consola para monitorizar

También podemos usar la consola para mostrar mensajes de monitorización. Para ello en las propiedades del proyecto, en la sección “Aplicación” fijar el “Tipo de resultado” a “Aplicación de consola”.

Ahora, podemos, por ejemplo, añadir una sentencia a nuestro `gestorglobal` para que también escriba en la consola:

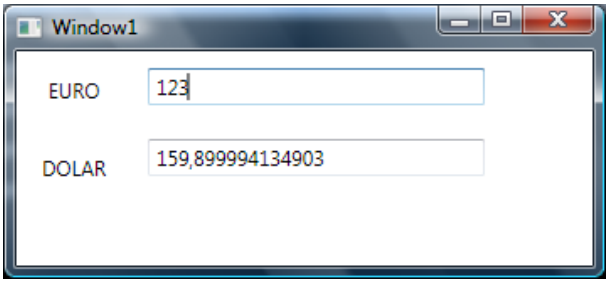
```
            Console.WriteLine(linea.Text);
```

Vamos a sobreescribir alguno de los métodos On*Evento* para que quede rastro en la consola de en qué momento se ejecuta. Por ejemplo:

```
        protected override void OnPreviewKeyDown(KeyEventArgs e)
        {
            base.OnPreviewKeyDown(e);
            Console.WriteLine("Ejecutado ONPreviewKeyDown");
        }
```

4. Crear una clase para personalizar el comportamiento

Queremos una aplicación que permita calcular la conversión de Euros en Dólares y viceversa:



Pero queremos ajustar el comportamiento de los cuadros de texto para que solo admitan la escritura de números. Además queremos que cuando pulsemos Enter en cualquiera de ellos, automáticamente se realice la conversión y se muestre el valor adecuado en el otro cuadro.

Para ello, no vamos a usar directamente un TextBox, sino que vamos a crear una nueva clase derivada de TextBox y vamos a justar su comportamiento.

Crear una nueva clase en el proyecto

En el explorador de soluciones, apuntar al proyecto, pulsar el botón derecho del ratón y elegir Agregar/Clase y poner un nombre (por ejemplo NumericTextBox). Se creará un nuevo fichero cs con la definición de la clase. Modificaremos la definición para que derive de TextBox.

Incluir en la interfaz los elementos de la nueva clase

Creamos la interfaz en el XAML. Para incluir los elementos de la nueva clase no podemos usar el atributo Class, ya que éste solo está permitido para el elemento raíz. Para los demás elementos es necesario indicar el espacio de nombres en el que están incluidos (y el ensamblado que los contiene, en el caso de que no se incluyan en el mismo proyecto) luego, nos referiremos a los elementos de la clase personalizada con el prefijo su namespace. Ver http://msdn.microsoft.com/es-es/library/vstudio/ms747086(v=vs.90).aspx . Por ejemplo:

```
<Window x:Class="Conversor_Euro_Dolar.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:custom="clr-namespace:Conversor_Euro_Dolar"
    Title="Window1" Height="300" Width="300" >
…
     <custom:NumericTextBox Width="200" Margin="10" x:Name="Euro" />
     <custom:NumericTextBox Width="200" Margin="10" x:Name="Dolar"/>
…
</Window>
```

Nótese que, cuando se usan clases personalizadas, no se puede usar directamente el atributo Name de WPF y hay que usar el Name definido para XAML por lo que tenemos que poner el prefijo x: (por ejemplo **x:Name**="Euro").

Personalizar el comportamiento de la clase

Añadir propiedades específicas

Para que resulte fácil obtener los valores numéricos correspondientes a los textos introducidos, añadimos dos nuevas propiedades:

```
        public int IntValue
         {
             get
             {
                 return Int32.Parse(this.Text);
             }
         }

        public double DoubleValue
         {
             get
             {
                 return Double.Parse(this.Text);
             }
         }
```

Filtrar los caracteres de entrada mediante el evento PreviewTextInput

Añadimos un gestor para el evento PreviewTextInput. Este evento se va a producir cada vez que las pulsaciones de teclado den lugar a la generación de un nuevo carácter en el control. Vamos a examinar esos caracteres y en el caso de que no se correspondan con un carácter adecuado, detendremos el enrutamiento del evento para que el carácter no se llegue a añadir a la cadena. Una primera aproximación (mejorable) puede ser la siguiente:

```
   void NumericTextBox_PreviewTextInput(object sender,
TextCompositionEventArgs e)
   {
       NumberFormatInfo numberFormatInfo =
System.Globalization.CultureInfo.CurrentCulture.NumberFormat;
       string decimalSeparator =
numberFormatInfo.NumberDecimalSeparator;
       string negativeSign = numberFormatInfo.NegativeSign;

       string caracter = e.Text;

       if (Char.IsDigit(e.Text[0]))
       {
           // No hacemos nada porque aceptamos los dígitos
       }
       else if (caracter.Equals(decimalSeparator) ||
caracter.Equals(negativeSign))
       { // No hacemos nada porque aceptamos el punto y el signo
       }
       else if (caracter == "\b")
       {
        // No hacemos nada porque aceptamos el Backspace
       }
       else
       {
         // Nos saltamos el carácter deteniendo el enrutamiento
```

```
            e.Handled = true;
        }
    }
```

En el constructor de la clase, registramos el gestor para el evento:

```
this.PreviewTextInput += new
TextCompositionEventHandler(NumericTextBox_PreviewTextInput);
```

Compilamos y comprobamos que los caracteres no permitidos son ignorados. Probamos a cambiar el evento gestionado haciendo que se gestione `TextInput` en vez de `PreviewTextInput`. Para eso, sustituimos el registro anterior por:

```
this.TextInput += new
TextCompositionEventHandler(NumericTextBox_PreviewTextInput);
```

Comprobamos que en este caso el comportamiento no es el deseado, pues cuando se gestiona el evento, el carácter ya ha aparecido en la caja.

Nótese que se está haciendo uso de la clase CultureInfo, que gestiona los caracteres de separación decimal según la configuración del sistema. Para poder usar esta clase hay que añadir el using del namespace adecuado.

Añadir la funcionalidad de la conversión a la aplicación

Ahora queremos que la ventana de la aplicación se entere de cuándo se pulsa la tecla enter en cualquiera de las dos cajas y que en ese momento se realice la conversión.

Para eso tenemos que escribir un gestor para el evento y registrarlo en el constructor de la clase derivada de Window:

```
        static float factor_ED = 1.30F;

        public Window1()
        {
            InitializeComponent();
            Dolar.KeyDown += new KeyEventHandler(Euro_KeyDown);
            Euro.KeyDown += new KeyEventHandler(Euro_KeyDown);

        }

        private void Euro_KeyDown(object sender, KeyEventArgs e)
        {

            if (e.Key == Key.Enter)
                if (sender == Euro)
                        Dolar.Text = ((Euro.DoubleValue *
factor_ED)).ToString();
                else if (sender == Dolar)
                        Euro.Text = ((Dolar.DoubleValue /
factor_ED)).ToString();

        }
```

Modificaciones

Añadir la interfaz necesaria para que el factor de conversión se muestre en la ventana y sea posible modificarlo mediante una caja de texto similar a las usadas.