CAPÍTULO 2.

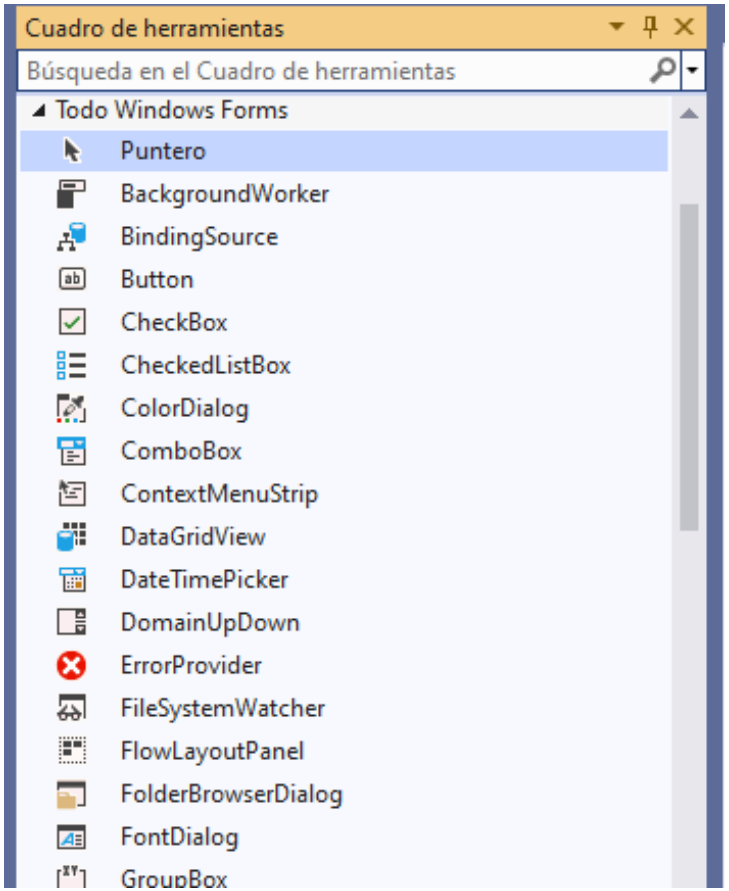
CREACIÓN DE COMPONENTES VISUALES

1. Concepto componente

Un **componente de software** es una unidad modular de un programa *software* con interfaces y dependencias bien definidas que permiten ofertar o solicitar un conjunto de servicios o funcionales.

La «programación orientada a componentes» (que también es llamada *basada en componentes*) es una rama de la ingeniería del software, con énfasis en la descomposición de sistemas ya conformados en componentes funcionales o lógicos con interfaces bien definidas usadas para la comunicación entre componentes.
Se considera que el nivel de abstracción de los componentes es más alto que el de los objetos y por lo tanto no comparten un estado y se comunican intercambiando mensajes que contienen datos.

1. Concepto componente



Como característica común, toda lenguaje que tenga como fin el desarrollo de aplicaciones con interfaz gráfica de usuaria contará con librerías con elementos básicos que agilicen el desarrollo de las misma.

Windows Forms provee de serie de una librería de componentes. Existe también la posibilidad de importar otras de otros desarrolladores.

En el cuadro de herramientas aparecen listadas y categorizadas siempre que te encuentre en el modo de diseñador de un elemento (ejemplo, un Form)

Los componentes quedan definidos por su funcionalidad, propiedades, eventos y aspecto gráfico en la propia interfaz.

2. Propiedades y atributos

Un propiedad es un parámetro modificable del componente. En el cuadro de propiedad se encuentran listadas todas aquellas a las que se tiene acceso durante el tiempo de vida del componente en tiempo de ejecución de la aplicación.

Están vinculados directamente a atributos de la propia clase que define el componente.

Ejemplo del componente “TrackBar”:

```
namespace System.Windows.Forms
{
    /// <summary>
    ///  The TrackBar is a scrollable control similar to the ScrollBar, but
    ///  has a different UI. You can configure ranges through which it should
    ///  scroll, and also define increments for off-button clicks. It can be
    ///  aligned horizontally or vertically. You can also configure how many
    ///  'ticks' are shown for the total range of values
    /// </summary>
    [DefaultProperty(nameof(Value))]
    [DefaultEvent(nameof(Scroll))]
    [DefaultBindingProperty(nameof(Value))]
    [Designer("System.Windows.Forms.Design.TrackBarDesigner, " + AssemblyRef.SystemDesign)]
    [SRDescription(nameof(SR.DescriptionTrackBar))]
    public partial class TrackBar : Control, ISupportInitialize
    {
        private static readonly object s_scrollEvent = new object();
        private static readonly object s_valueChangedEvent = new object();
        private static readonly object s_rightToLeftChangedEvent = new object();
```

Extracto de la definición de la clase TrackBar.

2. Propiedades y atributos

En la definición de la clase se puede comprobar la herencia y las interfaces que implementa.
En la imagen de la derecha están declarados los atributos e inicializados a un valor por defecto.

```
public partial class TrackBar : Control, ISupportInitialize
{
    private static readonly object s_scrollEvent = new object();
    private static readonly object s_valueChangedEvent = new object();
    private static readonly object s_rightToLeftChangedEvent = new object();

    private bool _autoSize = true;
    private int _largeChange = 5;
    private int _maximum = 10;
    private int _minimum;
    private Orientation _orientation = Orientation.Horizontal;
    private int _value;
    private int _smallChange = 1;
    private int _tickFrequency = 1;
    private TickStyle _tickStyle = TickStyle.BottomRight;

    private int _requestedDim;

    // Mouse wheel movement
    private int _cumulativeWheelData;
```

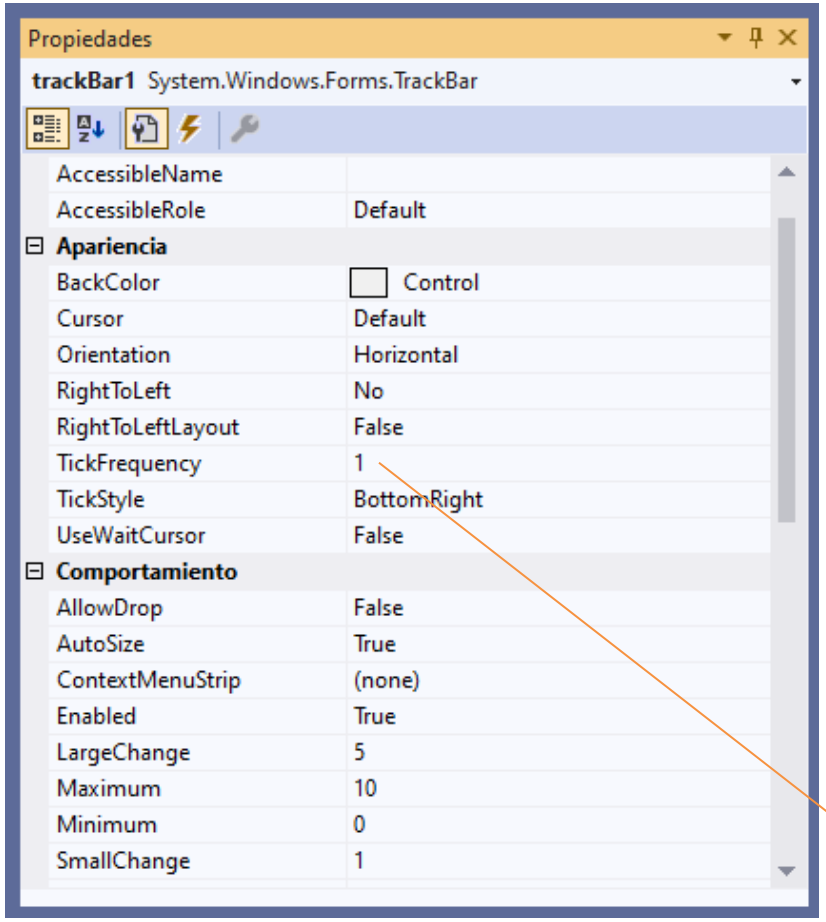
En la imagen de abajo, observamos la definición parcial de un *event handler*, acción desencadenada por un evento y la "sobreescritura" de otra función *event handler*.

```
/// <summary>
///  Actually fires the "ValueChanged" event.
/// </summary>
protected virtual void OnValueChanged(EventArgs e)
{
    // UIA events:
    AccessibilityObject.RaiseAutomationPropertyChangedEvent(UiaCore.UIA.ValueValuePropertyId, Name, Name);
    AccessibilityObject.RaiseAutomationEvent(UiaCore.UIA.AutomationPropertyChangedEventId);

    ((EventHandler)Events[s_valueChangedEvent])?.Invoke(this, e);
}

protected override void OnBackColorChanged(EventArgs e)
{
    base.OnBackColorChanged(e);
    RedrawControl();
}
```

3. Editores de propiedades



Habitualmente, el IDE te proporciona herramientas visuales de soporte y edición de las propiedades de los componentes.

Se puede comprobar la vinculación directa de los atributos declarados con las propiedades mostradas.

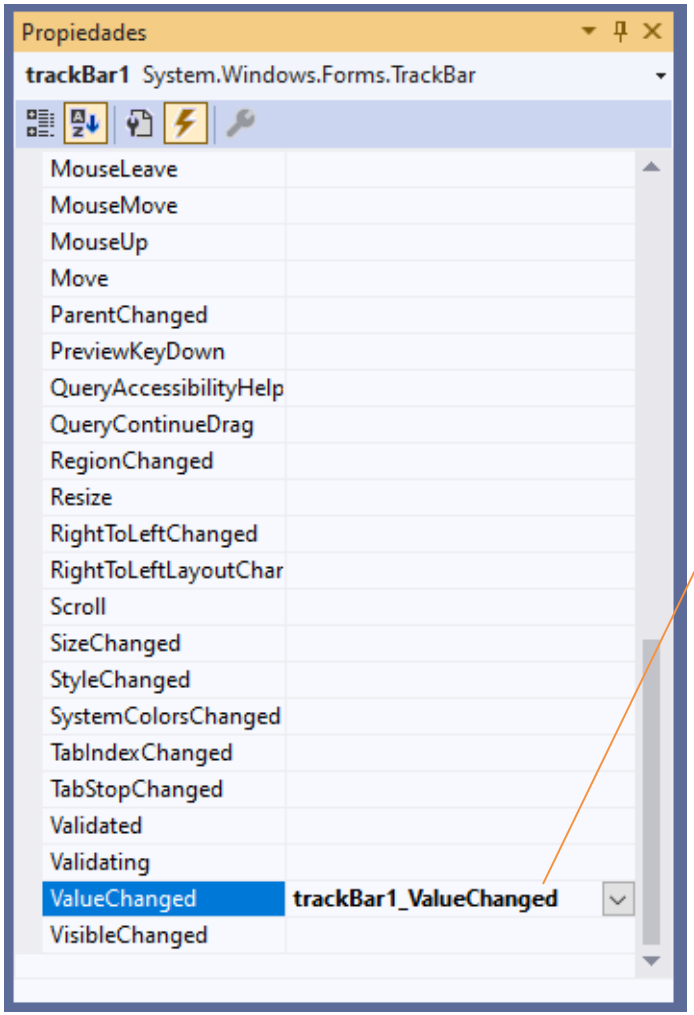
```
public partial class TrackBar : Control, ISupportInitialize
{
    private static readonly object s_scrollEvent = new object();
    private static readonly object s_valueChangedEvent = new object();
    private static readonly object s_rightToLeftChangedEvent = new object();

    private bool _autoSize = true;
    private int _largeChange = 5;
    private int _maximum = 10;
    private int _minimum;
    private Orientation _orientation = Orientation.Horizontal;
    private int _value;
    private int _smallChange = 1;
    private int _tickFrequency = 1;
    private TickStyle _tickStyle = TickStyle.BottomRight;

    private int _requestedDim;

    // Mouse wheel movement
    private int _cumulativeWheelData;
```

4. Eventos y asociación de acciones a eventos



En el mismo cuadro nos muestra los eventos para los cuales se ha definido un escuchador, *listener*.

```
/// <summary>
///  Actually fires the "ValueChanged" event.
/// </summary>
protected virtual void OnValueChanged(EventArgs e)
{
    // UIA events:
    AccessibilityObject.RaiseAutomationPropertyChangedEvent(UiaCor
    AccessibilityObject.RaiseAutomationEvent(UiaCore.UIA.Automatic

    ((EventHandler)Events[s_valueChangedEvent])?.Invoke(this, e);
}

protected override void OnBackColorChanged(EventArgs e)
{
    base.OnBackColorChanged(e);
    RedrawControl();
}
```

4. Eventos y asociación de acciones a eventos

La programación orientada a eventos es un paradigma de programación. Un **evento o suceso** es una acción que resulta de la **interacción de un usuario** o de un proceso interno del programa que está vinculado a un componente de la aplicación.

- Es necesario definir cada **evento** con la acción a la que va asociada. Esta acción es conocida como administrador de eventos, *event handler*.

- La idea es: cuando se ejecuta el programa, se realizan inicializaciones y después **el programa esperará a que ocurra cualquier evento.**

- **Cuando se dispara un evento el programa ejecuta el código correspondiente del administrador de eventos** (es decir, esa acción que hemos programado que haga). Ejemplo: un botón que cuando el usuario lo pulsa se reproduce un sonido.

- **La programación dirigida a eventos es la base de la interfaz de usuario** (ya que el usuario interactúa con ésta constantemente).

4. Eventos y asociación de acciones a eventos

- Los **eventos** son interacciones del usuario con la interfaz o sucesos internos del programa que afectarán a la respuesta del propio programa.

- Cada componente esta asociado a un evento o series de eventos.

- El desarrollador implementará una función de respuesta al evento en concreto, es decir, programará la gestión del evento creado, el *event handler (handler).*

- **El *event listener (listener)* es un mecanismo de respuesta, asíncrono**, para capturar eventos que pueden ocurrir en otras clases.

- Para dotar de reacción al *listener*, este **tiene que tener asociado un *handler*** que de respuesta al evento concreto.

4. Eventos y asociación de acciones a eventos

Notas

- Un listener solo puede tener un handler. Ej: La respuesta al click sobre el botón de cerrar aplicación, siempre será el cierre de la aplicación.

- Un handler puede estar asociado a distintos listener. Ej: un listener click de botón y de tecla pulsada puede tener asignado un mismo handler que muestre un mensaje modal diciendo “operación no aceptada”.

- Un handler podrá ser más o menos sofisticado si evalúa las propiedades del evento listener que lo ha disparado.

```
1 referencia
private void trackBar1_ValueChanged(object sender, EventArgs e)
{
    //Definición de lo que realizará el programa en respuesta
    //al evento evaluado.
}
```

5. Introspección y reflexión

En los lenguajes dinámicos, como por ejemplo, Javascript, es posible alterar los objetos en tiempo de ejecución, sea cual sea su clase, **añadiendo atributos o haciendo comprobaciones sobre ellos**. Esto permite hacer algunas operaciones bastante creativas en nuestros desarrollos, o disponer de librerías que hacen «magia».

Pero en los lenguajes estáticos como por ejemplo Java o C#, esto ya es menos común. Solemos partir de la base de que tenemos clases prácticamente inamovibles, de las cuales surgen los objetos, y tampoco nos suele cuadrar en la cabeza hacer algún tipo de consulta en tiempo de ejecución sobre sus atributos y propiedades, o directamente su modificación: no cuadra mucho con la programación orientada a objetos puramente dicha.

Como vemos, **C# es un lenguaje fuertemente tipado**, pero puede **permitir hacer algunas operaciones especiales** ayudándose de dos conceptos:

•**Introspección**: la capacidad para inspeccionar los metadatos de un objeto, como atributos, propiedades, visibilidad…

•**Reflexión**: la capacidad para alterar en tiempo de ejecución los metadatos, como añadir atributos, alterar visibilidad…

https://www.adictosaltrabajo.com/2016/08/23/introspeccion-y-reflexion-en-java-para-testear-clases-de-utilidades/

La hamburguesería del vecino

1. Análisis de la estructura de clases, código e información adjunta.
2. Reformulación, creación de una App escritorio que te permita:

a. Página de bienvenida (puede incluir un pequeño texto de ayuda que facilite la utilización de la app)
b. Registro de pedido del cliente, cesta (elementos, cantidad, precio total)
c. Expedición de número de orden de pedido y recibo en una página nueva **con un componente específico diseñado** por el programador.
d. Información de los ingredientes del producto (obligatorio en el producto principal)
e. Registro de información adicional para la comanda (cuadro de texto donde pueda haber observación importantes, cambio de ingredientes, alergias, etc)
f. Posibilidad de guardar pedido para realizar un pedido rápido previo.
g. Claridad en la interfaz

¡Recuerda!: *Es el vecino el que elige la hamburguesería y es la hamburguesería la que quiere que sean los vecinos la hamburguesería.*