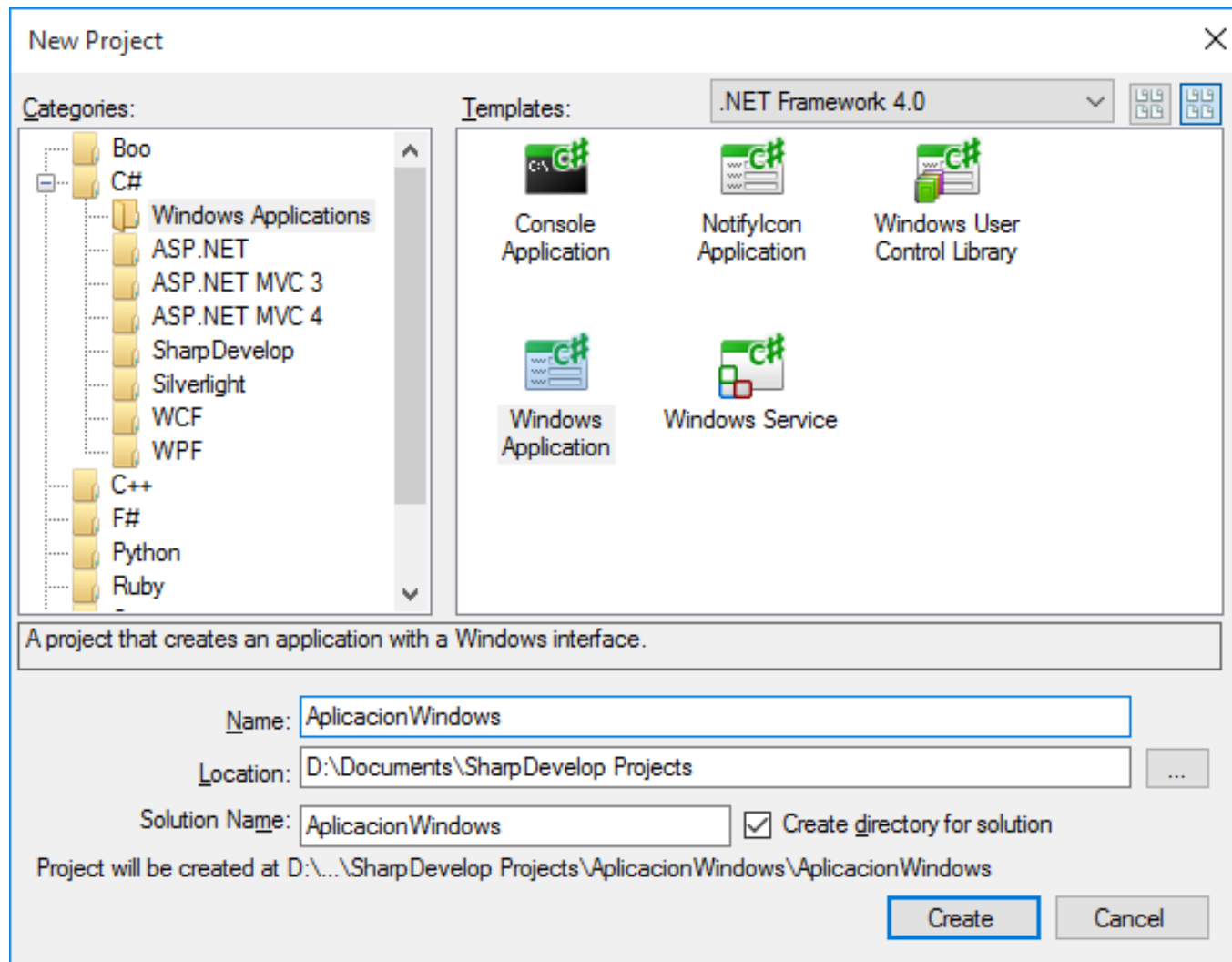


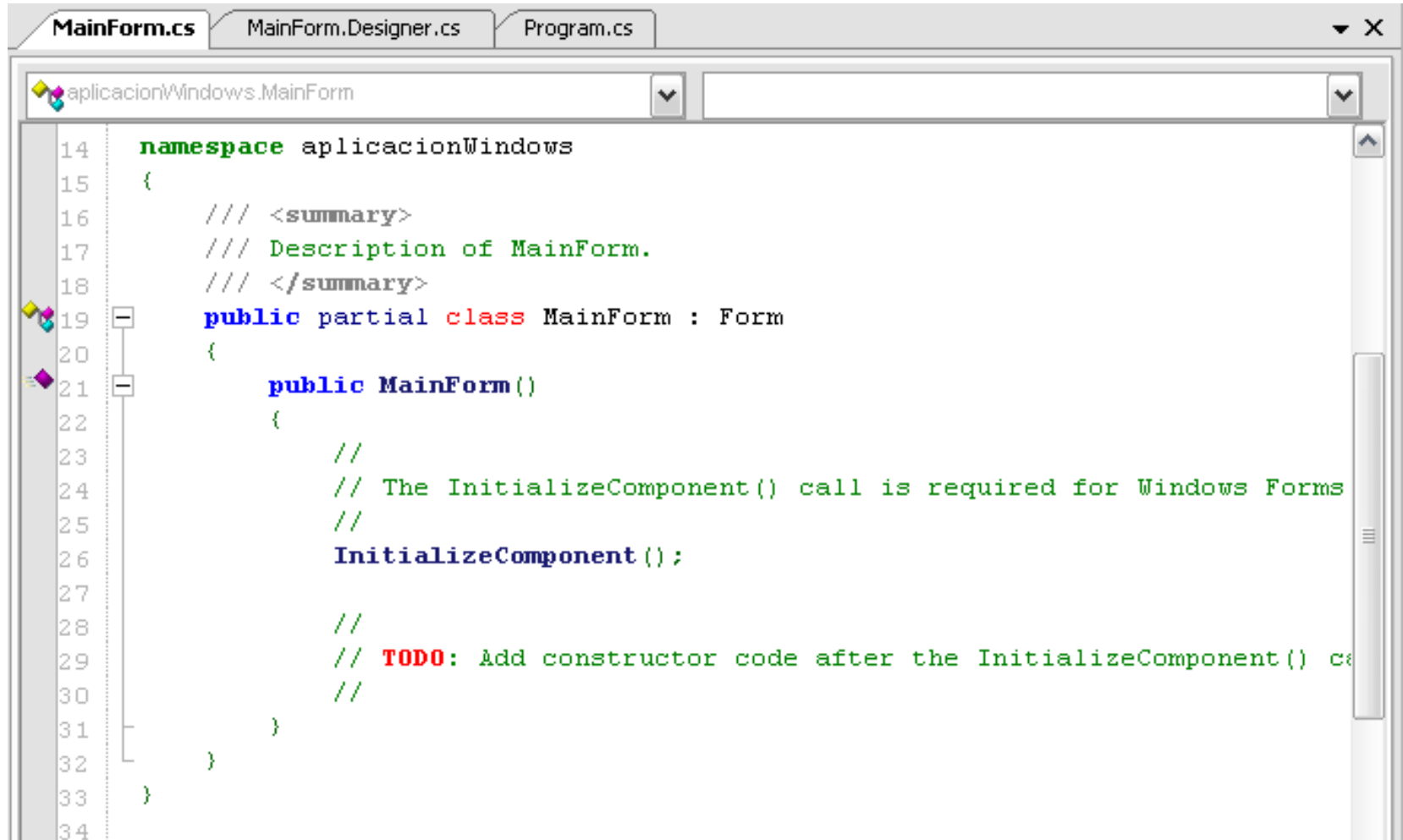
C#.Net

Aplicaciones Windows

Utilizando SharpDevelop cree una nueva aplicación windows



¿De qué clase hereda MainForm?



```
14 namespace aplicacionWindows
15 {
16     /// <summary>
17     /// Description of MainForm.
18     /// </summary>
19     public partial class MainForm : Form
20     {
21         public MainForm()
22         {
23             //
24             // The InitializeComponent() call is required for Windows Forms
25             //
26             InitializeComponent();
27
28             //
29             // TODO: Add constructor code after the InitializeComponent() call
30             //
31         }
32     }
33 }
```

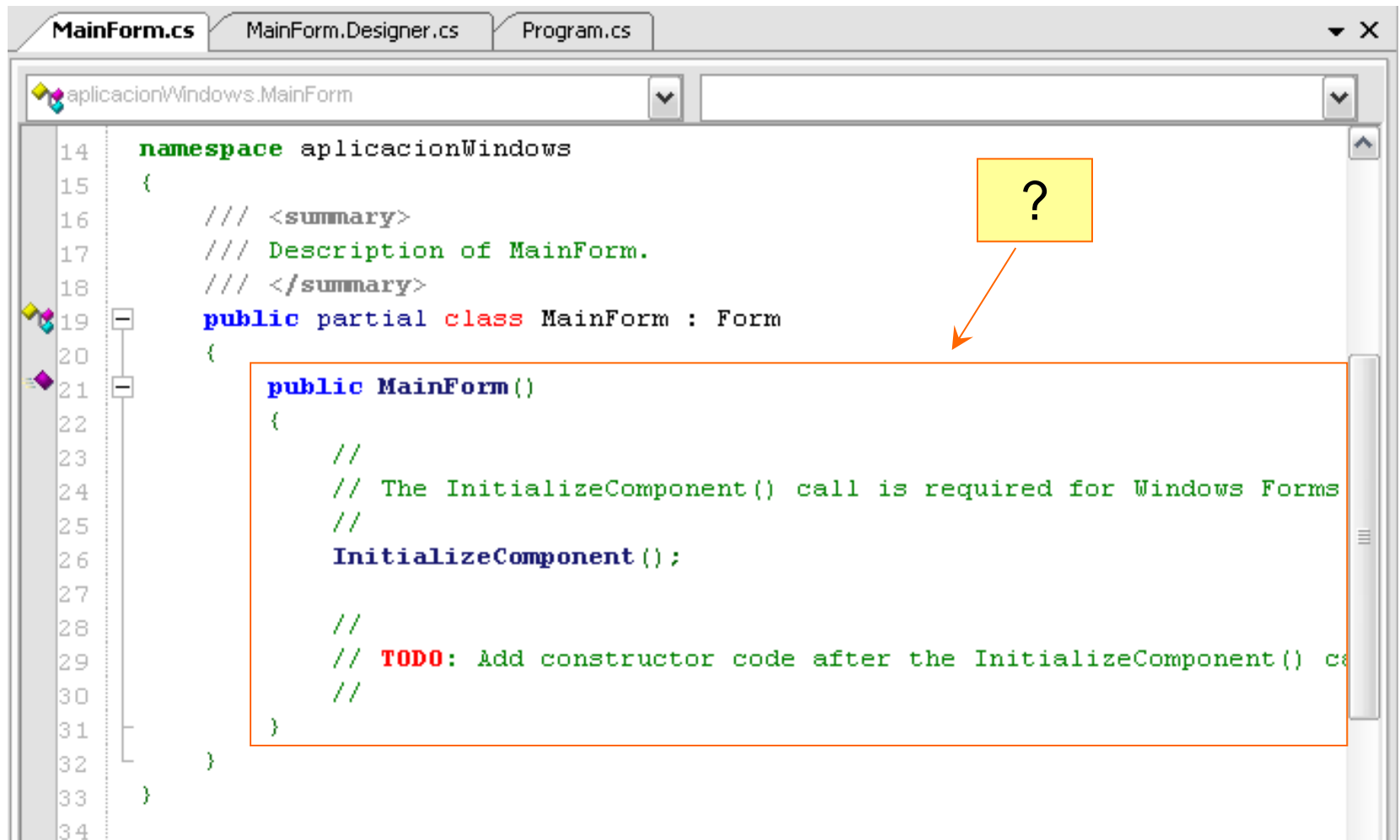
¿De qué clase hereda MainForm?

The screenshot shows a Visual Studio code editor with three tabs: **MainForm.cs**, **MainForm.Designer.cs**, and **Program.cs**. The **MainForm.cs** tab is active, showing the following code:

```
14 namespace aplicacionWindows
15 {
16     /// <summary>
17     /// Description of MainForm.
18     /// </summary>
19     public partial class MainForm : Form
20     {
21         public MainForm()
22         {
23             //
24             // The InitializeComponent() call is required for Windows Forms
25             //
26             InitializeComponent();
27
28             //
29             // TODO: Add constructor code after the InitializeComponent() call
30             //
31         }
32     }
33 }
```

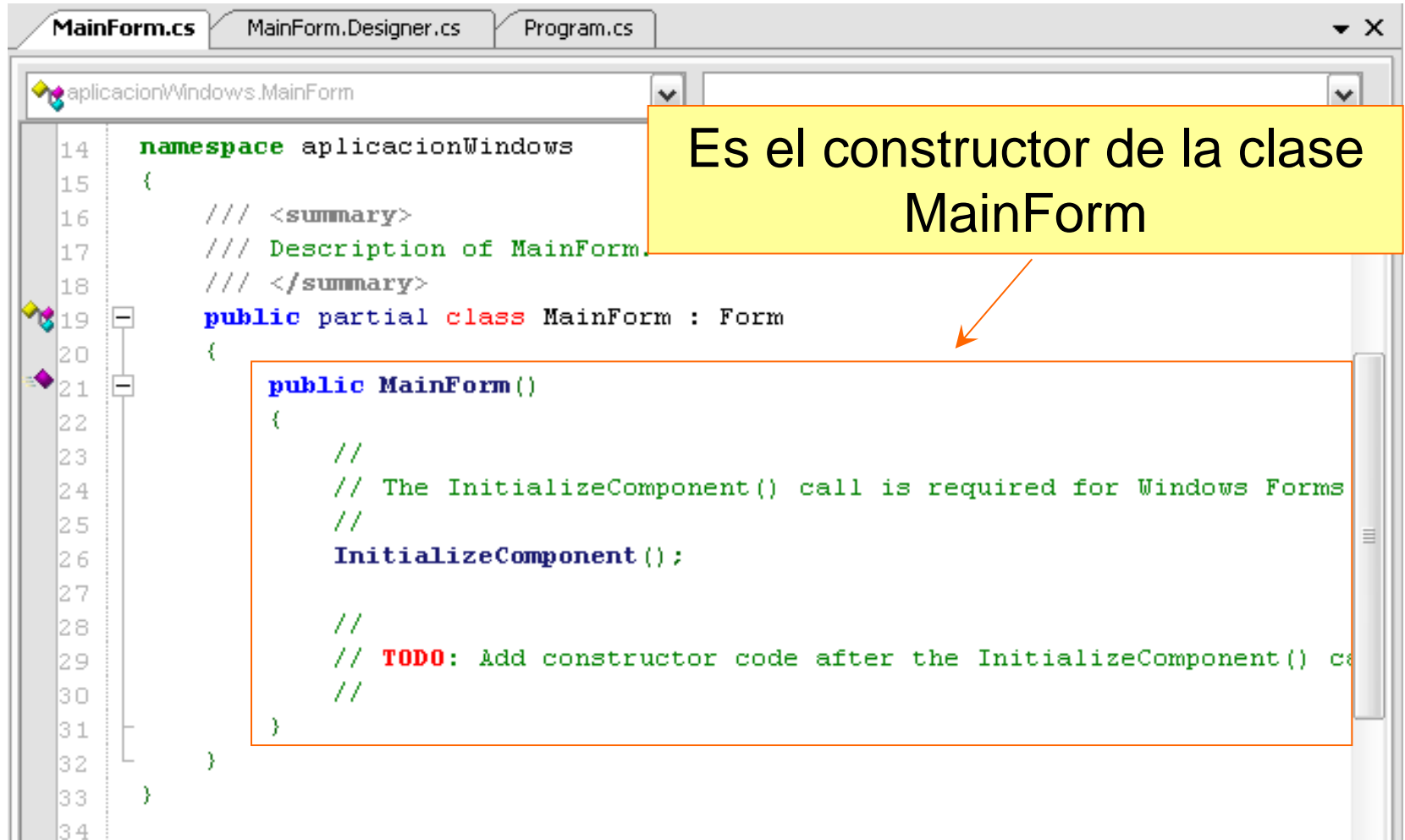
A yellow callout box with the text "MainForm es una subclase de Form" has an arrow pointing to the line `public partial class MainForm : Form` in the code.

¿Qué significa?

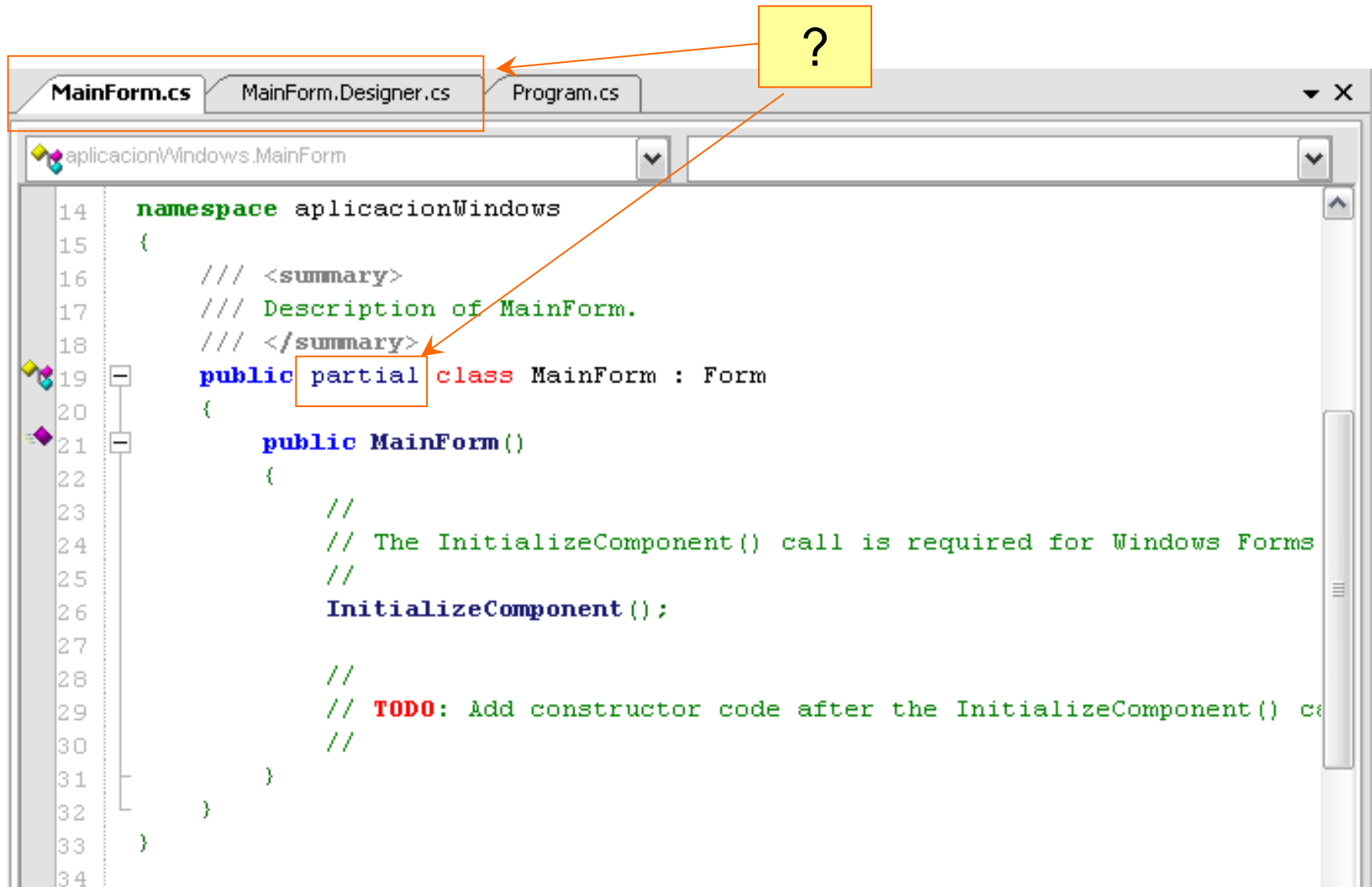


```
14 namespace aplicacionWindows
15 {
16     /// <summary>
17     /// Description of MainForm.
18     /// </summary>
19     public partial class MainForm : Form
20     {
21         public MainForm()
22         {
23             //
24             // The InitializeComponent() call is required for Windows Forms
25             //
26             InitializeComponent();
27
28             //
29             // TODO: Add constructor code after the InitializeComponent() call
30             //
31         }
32     }
33 }
```

¿Qué significa?



¿Qué significa?



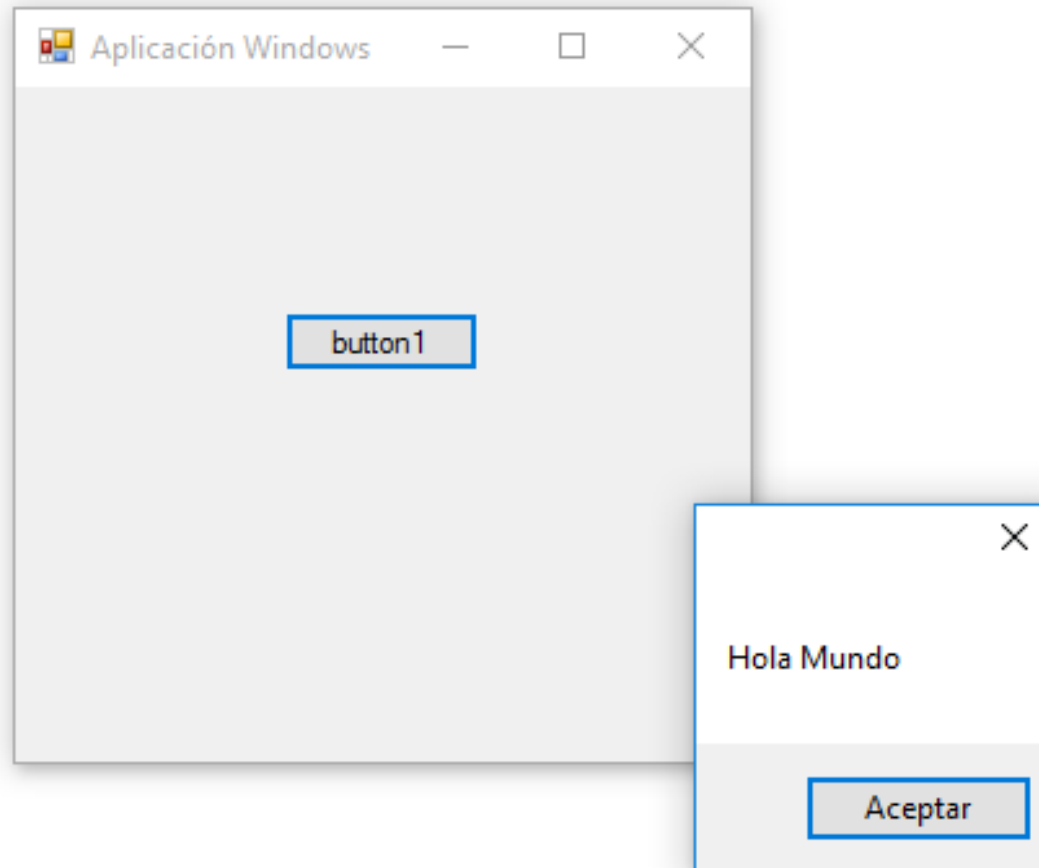
¿Qué significa?

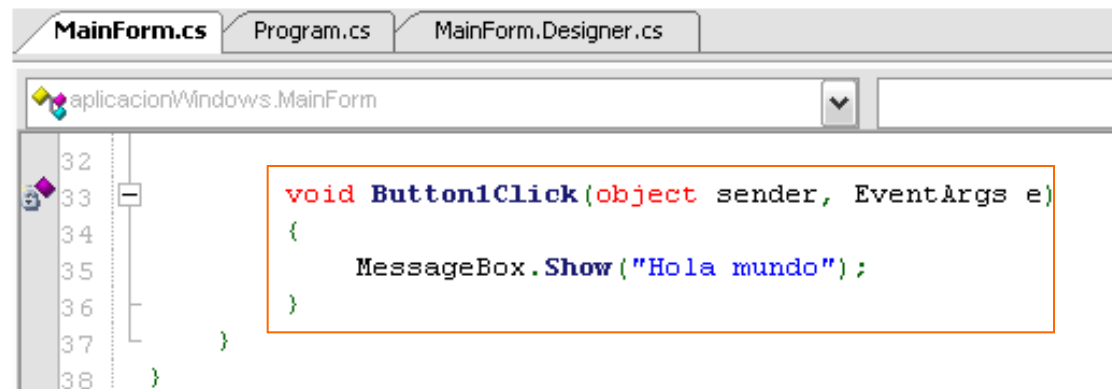
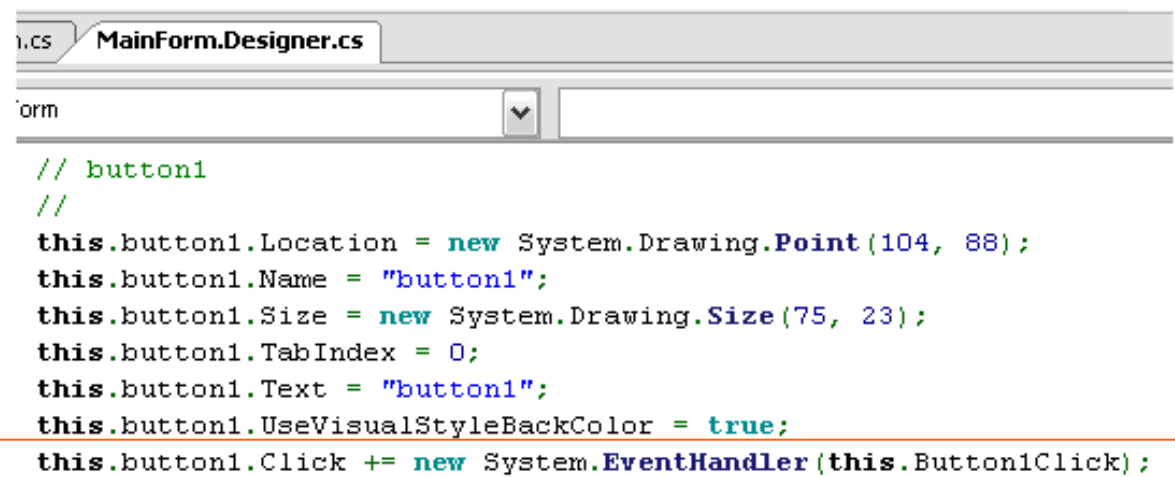
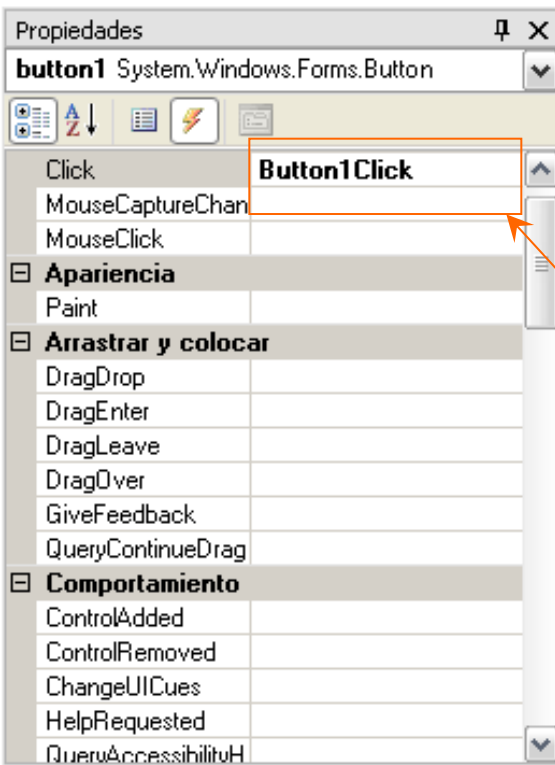
La definición de la clase se encuentra repartida en dos archivos



```
14 namespace aplicacionWindows
15 {
16     /// <summary>
17     /// Description of MainForm.
18     /// </summary>
19     public partial class MainForm : Form
20     {
21         public MainForm()
22         {
23             //
24             // The InitializeComponent() call is required for Windows Forms
25             //
26             InitializeComponent();
27
28             //
29             // TODO: Add constructor code after the InitializeComponent() call
30             //
31         }
32     }
33 }
```


Agregue un botón y codifique el manejador para el evento click mostrando un mensaje con el clásico “Hola mundo”





Propiedad Controls

- Los contenedores poseen una colección de controles accesible desde la propiedad Controls.
- Por contenedores entiéndase controles que pueden contener a otros controles en su interior (Form, Panel, GroupBox, etc.)

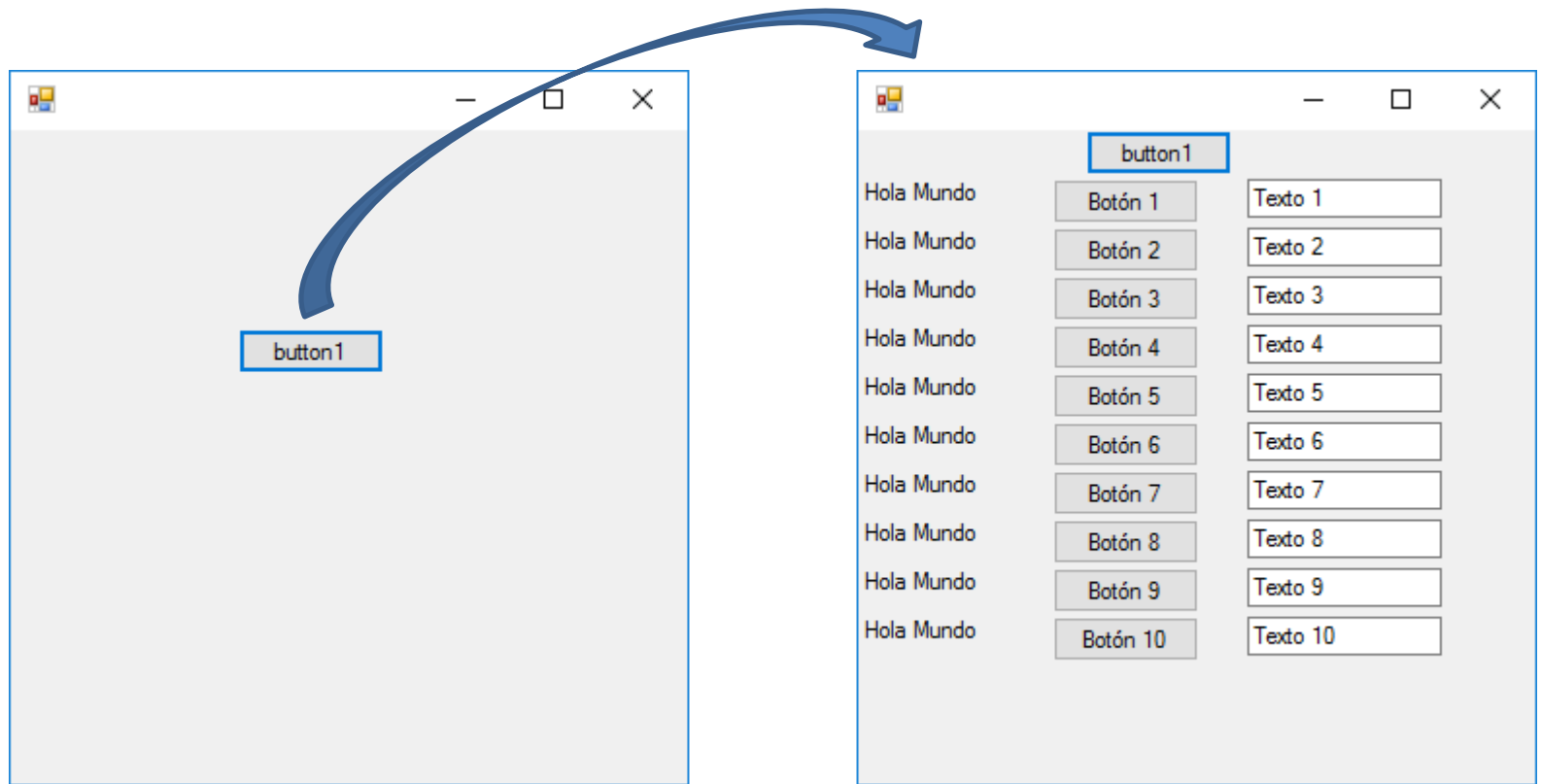
Propiedad Controls

- Modifique la aplicación anterior codificando el manejador para el evento click del botón de la siguiente manera

```
void Button1Click(object sender, EventArgs e)
{
    button1.Top = 0;
    for(int i=1; i<=10; i++)
    {
        Label lab = new Label(); lab.Text = "Hola Mundo";
        Button bot = new Button(); bot.Text = "Botón "+i;
        TextBox texto = new TextBox(); texto.Text = "Texto "+i;
        lab.Top = i*25; bot.Top = lab.Top; bot.Left = 100;
        texto.Top = lab.Top; texto.Left = 200;
        this.Controls.Add(lab);
        this.Controls.Add(bot);
        this.Controls.Add(texto);
    }
}
```

Ejecute y pruebe

Propiedad Controls



Propiedad Controls

- Agregue un nuevo botón y codifique el manejador para el evento click del botón de la siguiente manera

```
void Button2Click(object sender, EventArgs e)
{
    while (Controls.Count > 0)
    {
        Controls.RemoveAt(0);
    }
}
```

Ejecute y pruebe

Propiedad Controls

EJERCICIO:

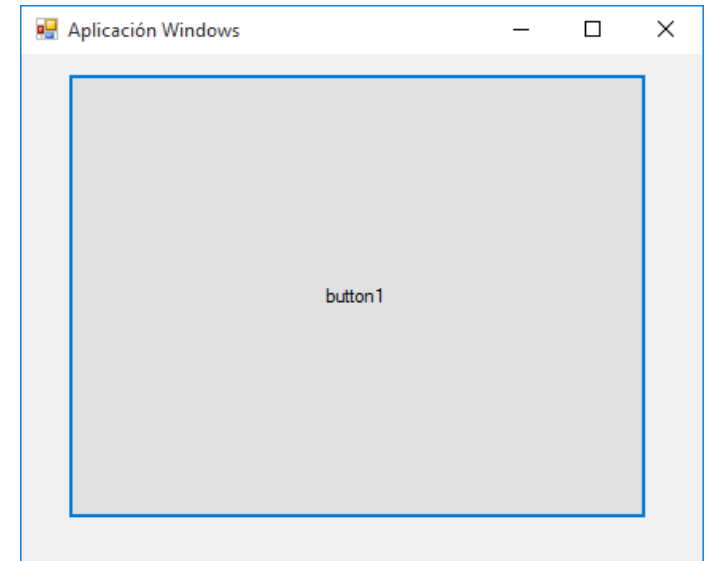
- Agregue un tercer botón para borrar sólo los controles que sean botones. Para preguntar por el tipo de un control utilice el operador **is**, por ejemplo (**c is Button**) devuelve **true** si el control **c** es un botón.

Propiedad Controls

```
void Button3Click(object sender, EventArgs e)
{
    int i = 0;
    while (i < Controls.Count )
    {
        if (Controls[i] is Button)
        {
            Controls.RemoveAt(i);
        }
        else
        {
            i++;
        }
    }
}
```


Propiedad Controls

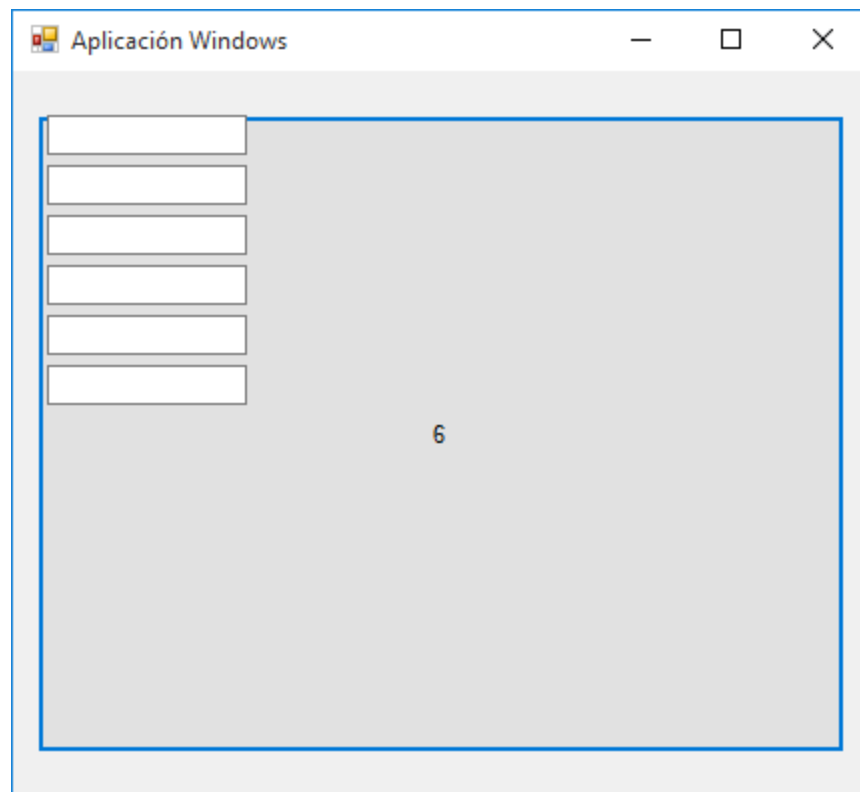
- En realidad no sólo los contenedores poseen la propiedad Controls.
- **EJERCICIO:** Cada vez que el usuario haga clic sobre el botón debe agregarse un nuevo TextBox dentro del botón



Propiedad Controls

```
void Button1Click(object sender, EventArgs e)
{
    TextBox t = new TextBox();
    t.Top = button1.Controls.Count * (t.Height + 5);
    t.Left = 5;
    button1.Controls.Add(t);
    button1.Text = button1.Controls.Count.ToString();
}
```

Propiedad Controls



Heredando de un control

- Pruebe lo siguiente

```
class panelNuevo:Panel{
    Button boton1=new Button();
    Button boton2=new Button();
    public panelNuevo(){
        this.BackColor=Color.LightBlue;
        boton1.Width=30;boton2.Width=30;
        boton1.Height=30;boton2.Height=30;
        boton2.Left =30;
        this.Controls.Add(boton1);
        this.Controls.Add(boton2);
    }
}
```

Heredando de un control

- Codifique el manejador para el evento Load del formulario

```
void MainFormLoad(object sender, EventArgs e)
{
    this.Controls.Add(new panelNuevo());
}
```

Heredando de un control

- En la clase panelNuevo agregue:

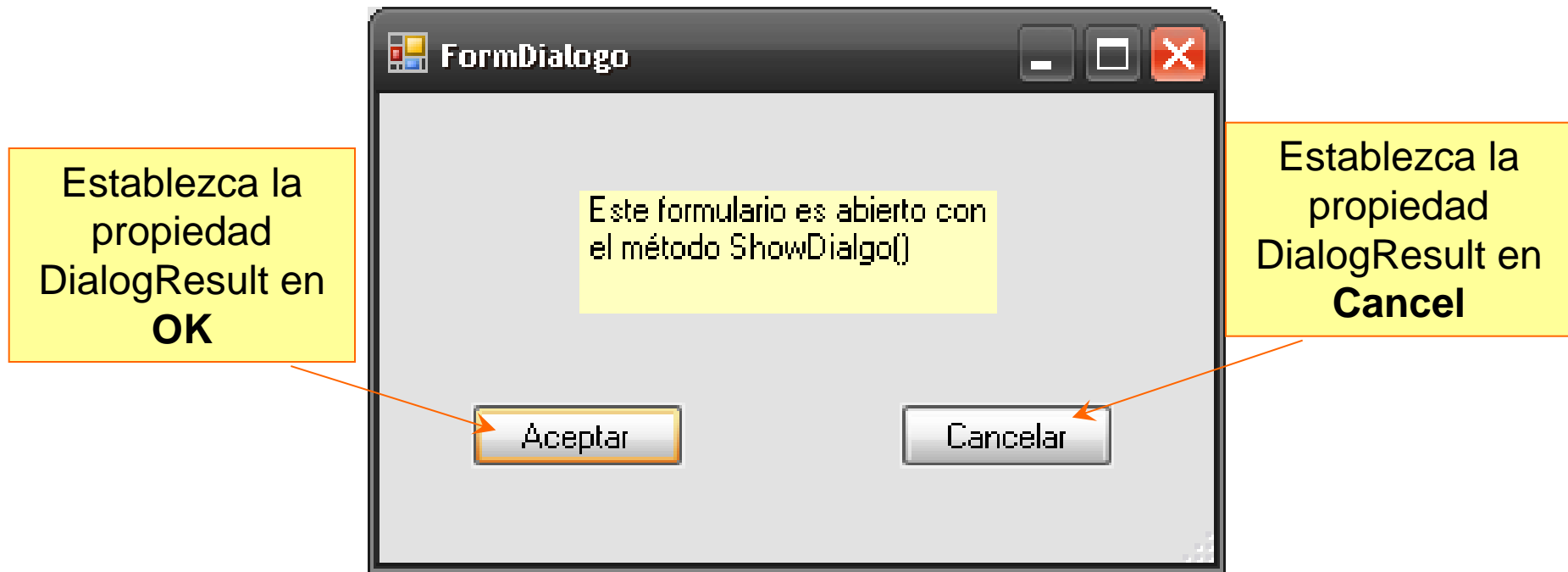
```
class panelNuevo:Panel{
    Button boton1=new Button();
    Button boton2=new Button();
    public panelNuevo(){
        this.BackColor=Color.LightBlue;
        boton1.Width=30;boton2.Width=30;
        boton1.Height=30;boton2.Height=30;
        boton2.Left =30;
        this.Controls.Add(boton1);
        this.Controls.Add(boton2);
        boton1.Click+= new EventHandler(boton1_Click);
        boton2.Click+= new EventHandler(boton2_Click);
    }
    void boton1_Click(object sender, EventArgs e) {
        this.Width-=10;
    }
    void boton2_Click(object sender, EventArgs e){
        this.Width +=10;
    }
}
```

Cuadros de diálogos

- Cuando se abre un formulario con el método **ShowDialog()**, el control sólo vuelve a la aplicación cuando ese formulario se cierra.
- Suelen utilizarse cuando es necesario que el usuario tome alguna decisión
- Muchas veces será necesario saber cuál ha sido la respuesta del usuario.

Cuadros de diálogos

- Cree una aplicación windows. Agregue un segundo formulario y diseñelo de la siguiente manera.



Cuadros de diálogos

- Agregue un botón al formulario principal y codifique el manejador para el evento click de la siguiente manera

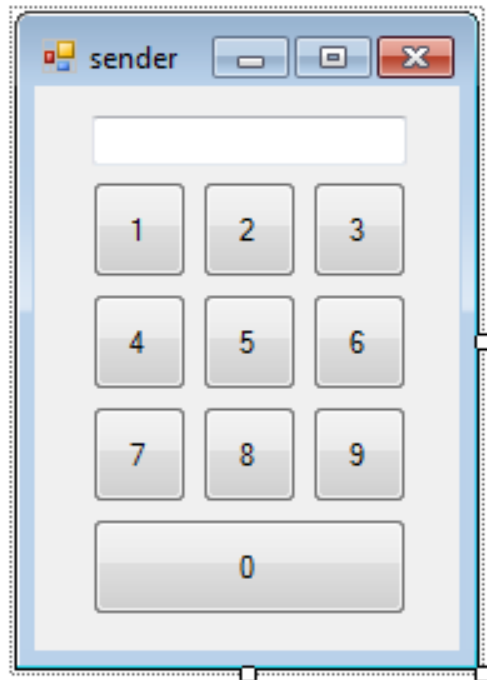
```
void Button1Click(object sender, EventArgs e)
{
    Form2 f=new Form2();
    if (f.ShowDialog()==DialogResult.OK)
        MessageBox.Show("Aceptaron");
}
```

Ejecute y pruebe

Parámetro Sender

- A veces resulta útil compartir un único manejador de evento entre muchos controles

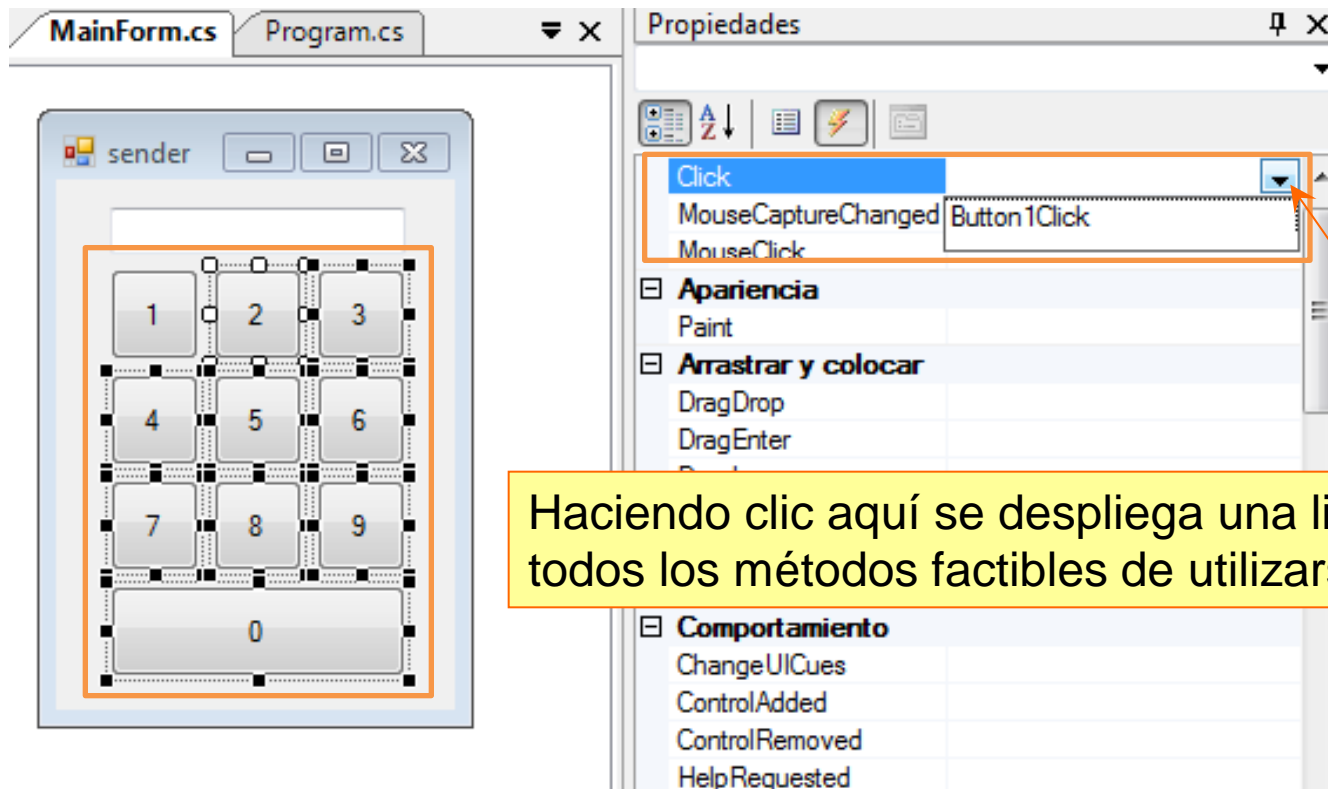
MainForm.cs Program.cs MainI



A medida que el usuario presiona los botones, deben ir apareciendo en el textbox superior los números correspondientes

Parámetro Sender

- Codifique el manejador para el evento Clic del primer botón y asigne el mismo a todos los otros botones



Parámetro Sender

En el manejador del evento debe utilizar el parámetro sender para identificar al botón presionado

```
void Button1Click(object sender, EventArgs e)
{
    textBox1.Text += (sender as Button).Text;
}
```

Es necesario hacer un casting porque sender es de tipo object, y la propiedad Text no está definida en object