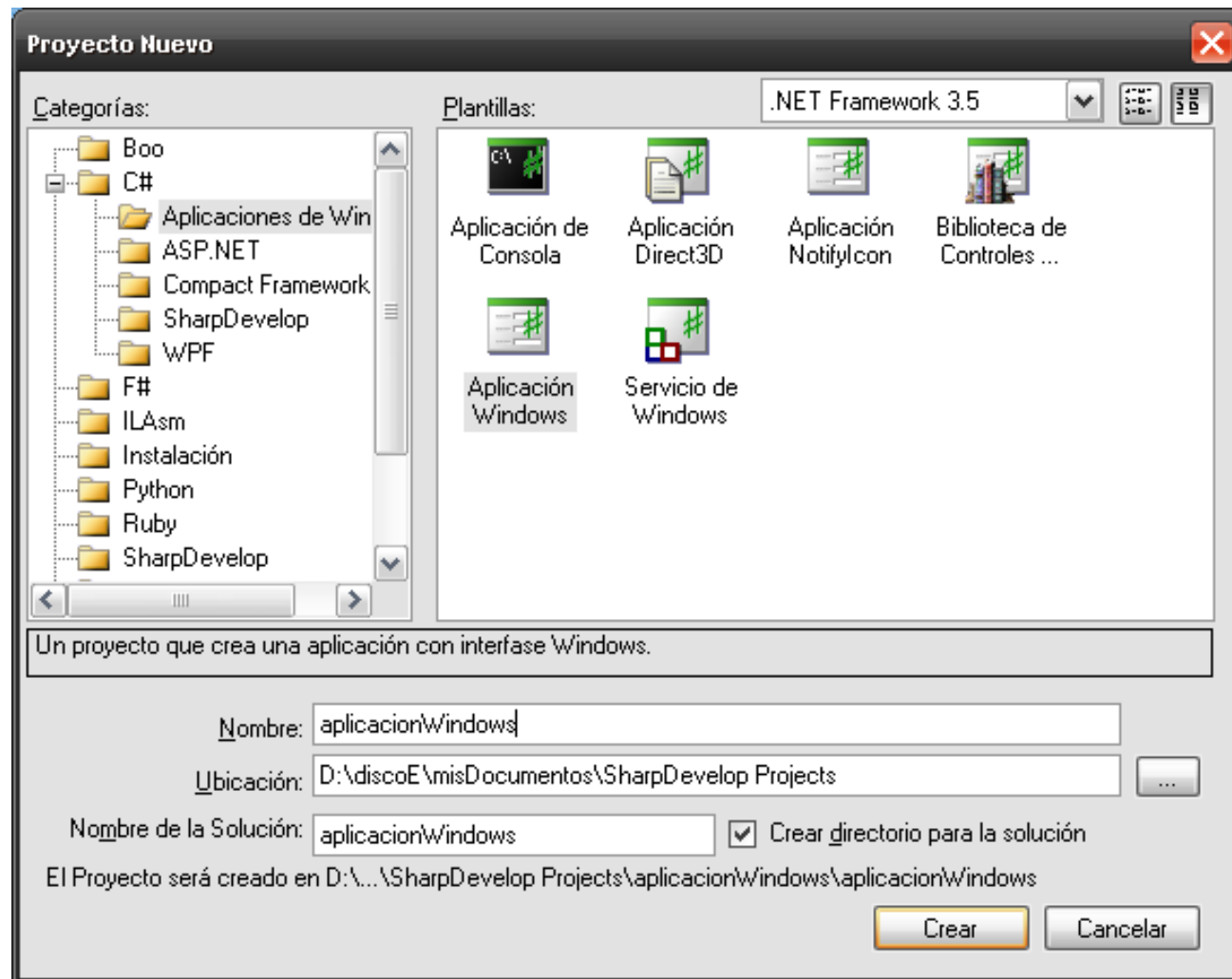


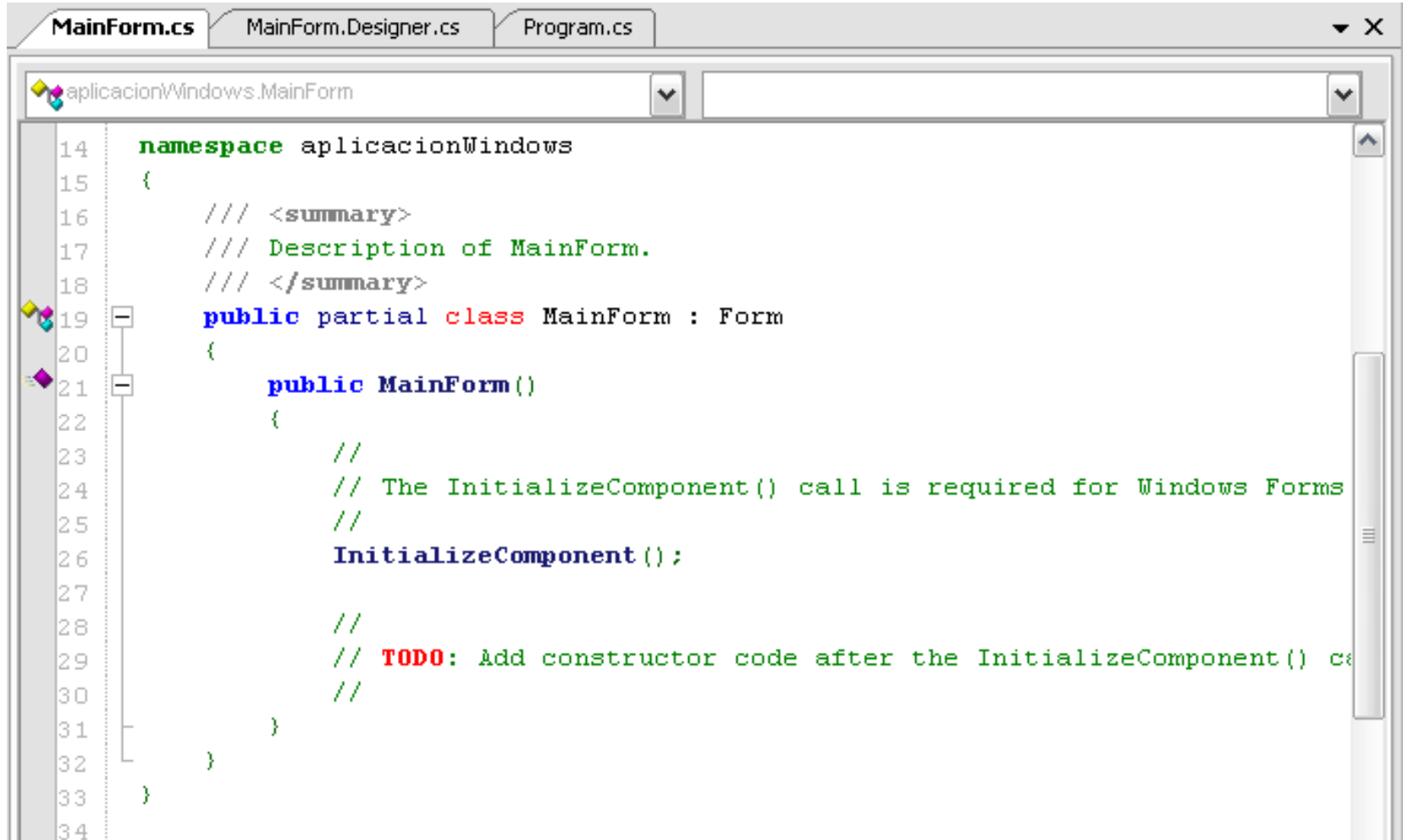
C#.Net

Aplicaciones Windows

Utilizando SharpDevelop cree una nueva aplicación windows

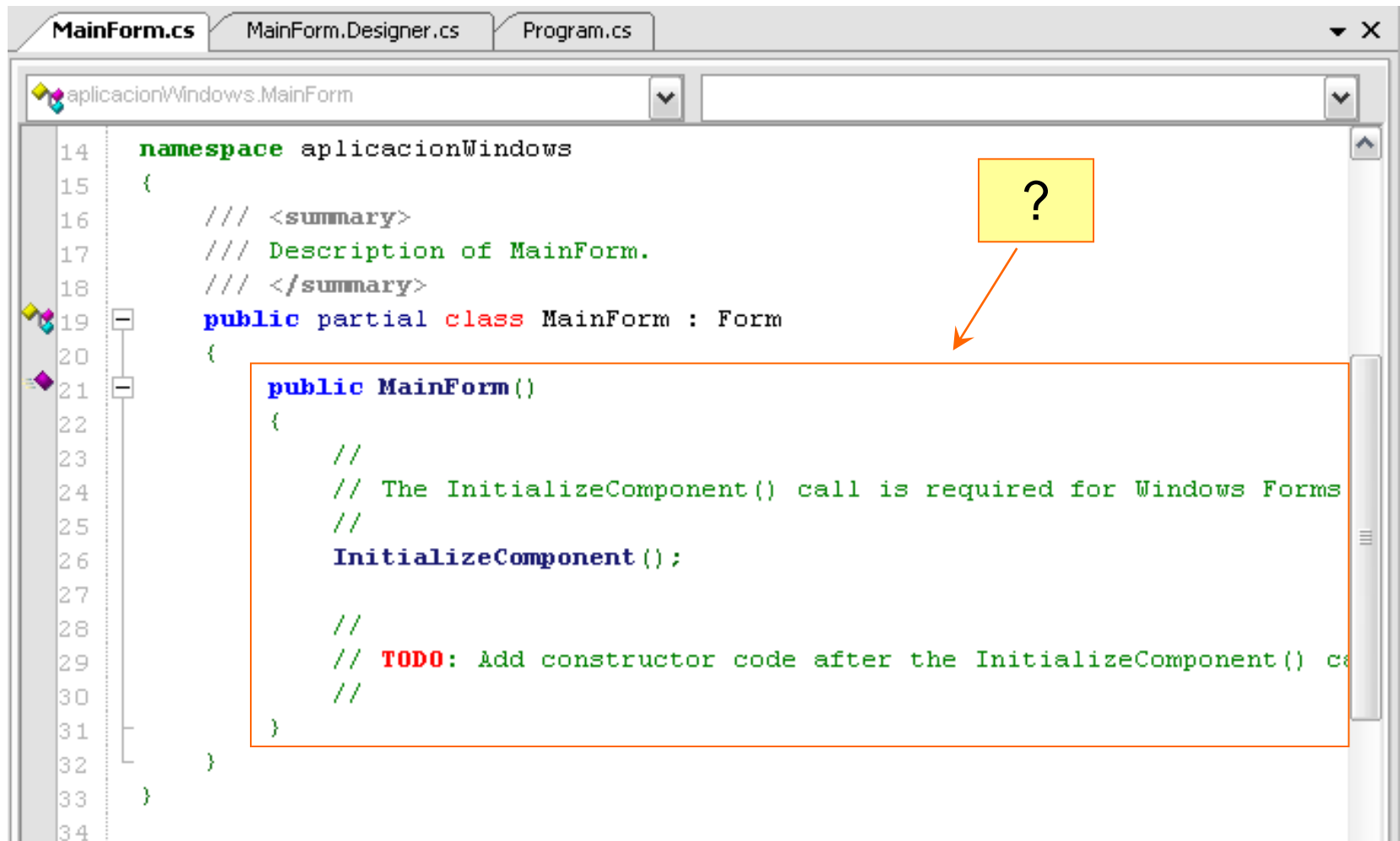


¿De qué clase hereda MainForm?



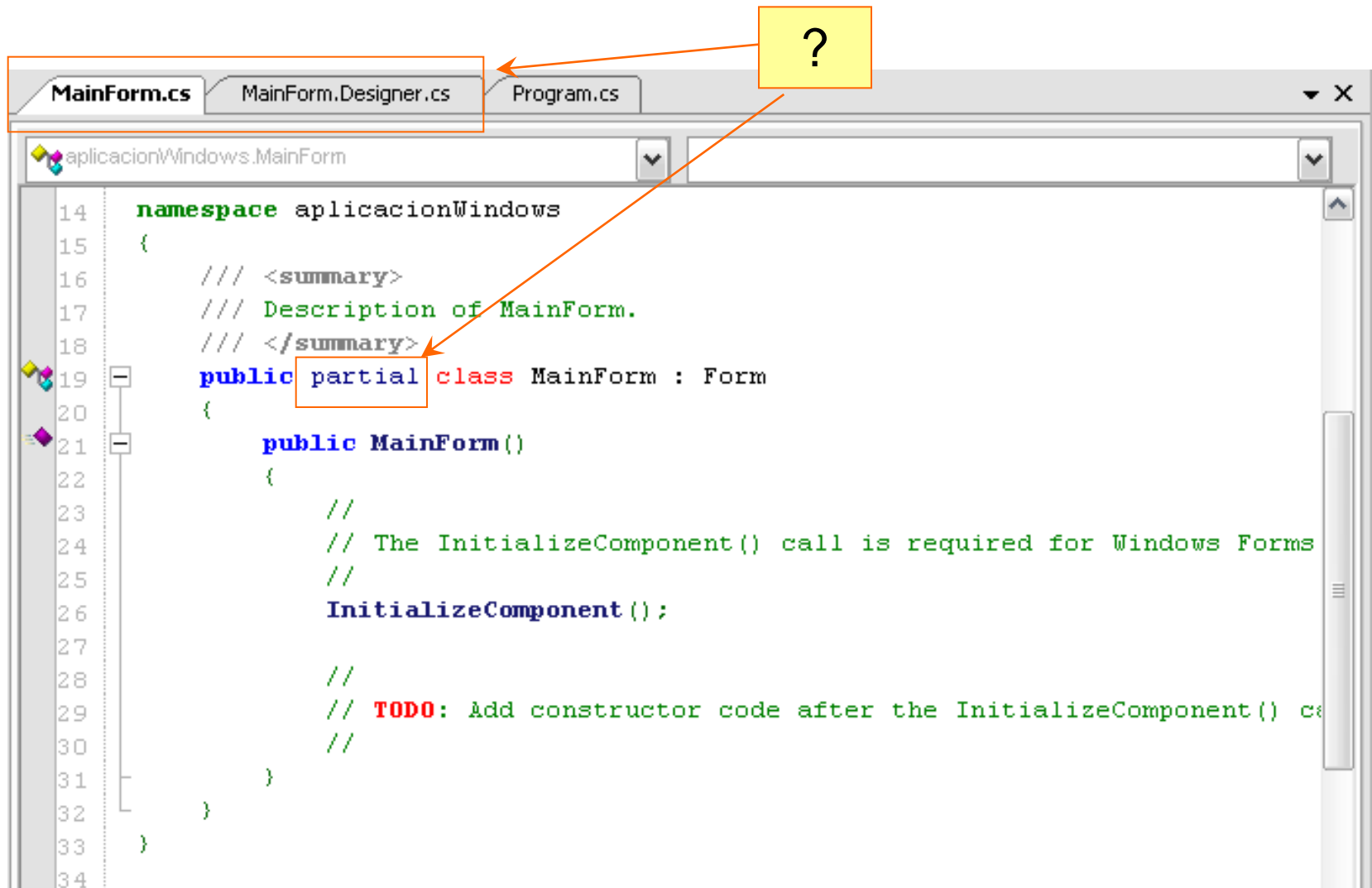
```
14 namespace aplicacionWindows
15 {
16     /// <summary>
17     /// Description of MainForm.
18     /// </summary>
19     public partial class MainForm : Form
20     {
21         public MainForm()
22         {
23             //
24             // The InitializeComponent() call is required for Windows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

¿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 }
34
```

¿Qué significa?



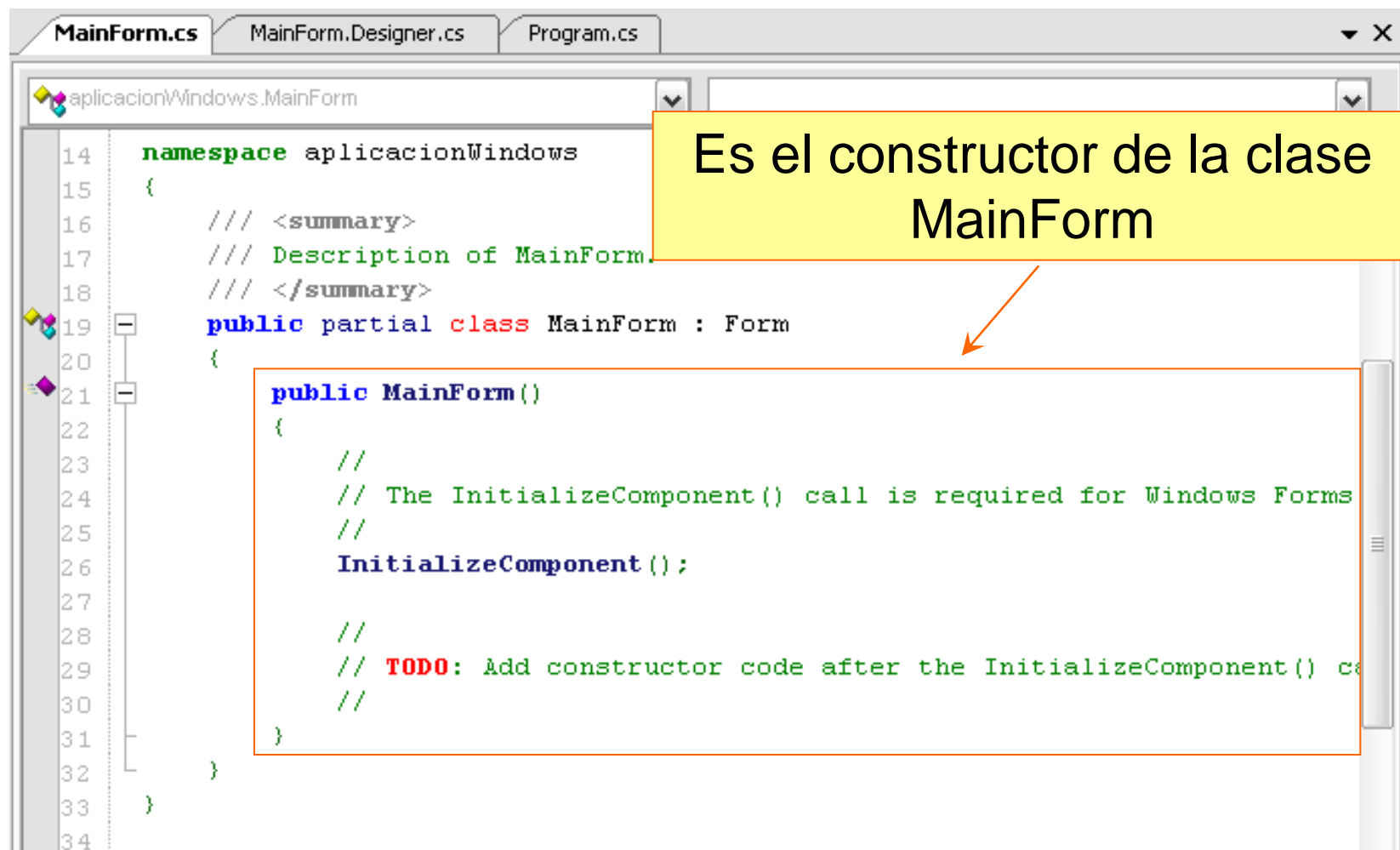
¿De qué clase hereda MainForm?

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

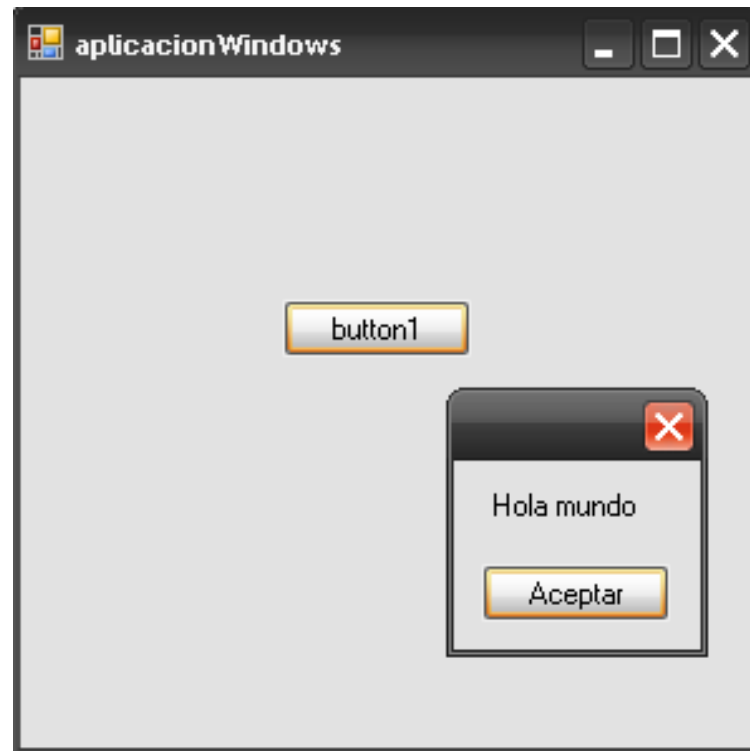


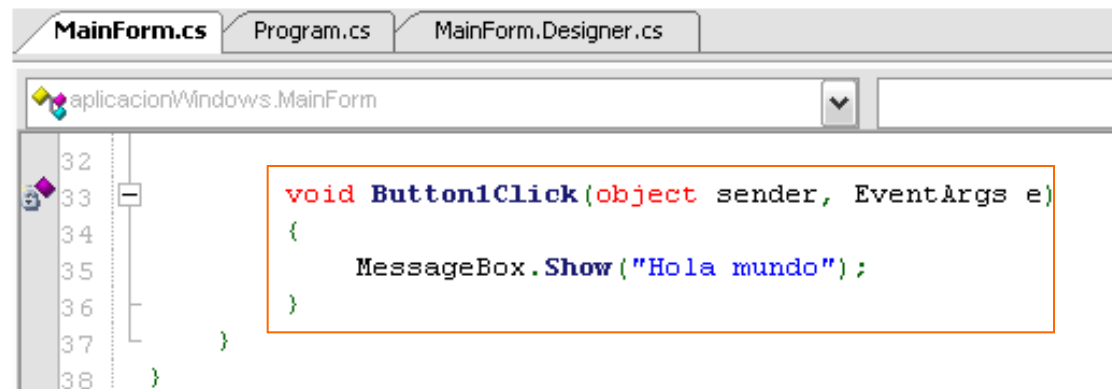
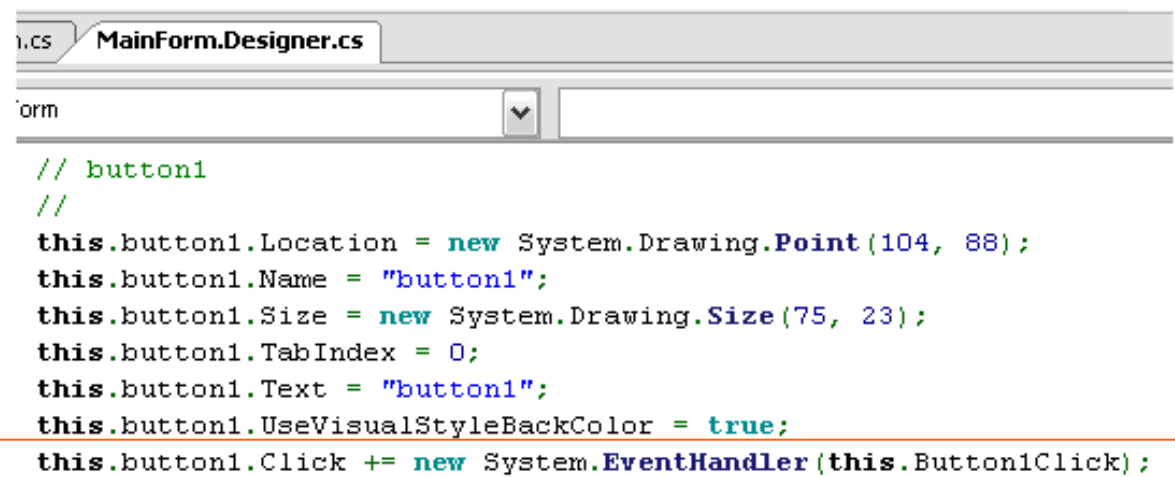
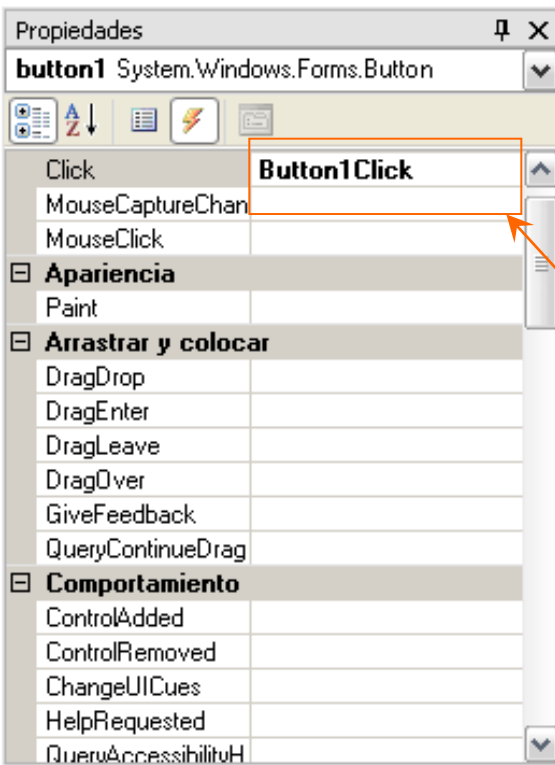
¿Qué significa?

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)
{
    for (int i= 1;i<=10;i++ ){
        Label l=new Label();l.Text="Hola Mundo";
        Button b= new Button();
        l.Top=i*25; b.Top=l.Top; b.Left=70;
        this.Controls.Add(b);
        this.Controls.Add(l);
    }
}
```

Ejecute y pruebe

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:

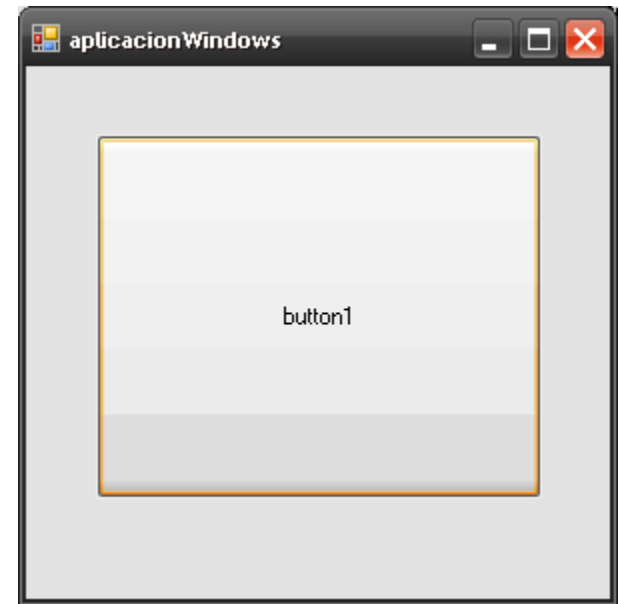
- Agregue un tercer botón para borrar sólo los controles que sean botones. Para preguntar por el tipo de un control utilice el operador **is**, por ejemplo (**c is Button**) devuelve **true** si el control **c** es un botón.

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;
    button1.Controls.Add(t);
}
```

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:

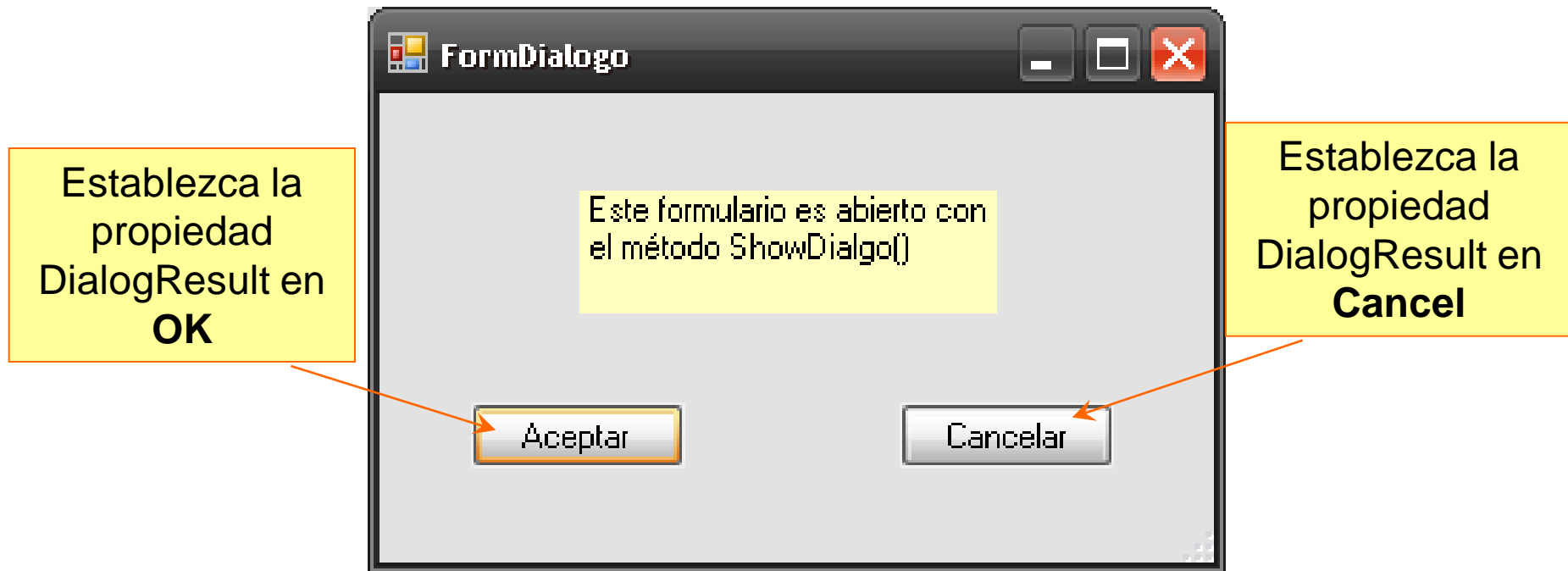
```
class panelNuevo:Panel{
    Button boton1=new Button();
    Button boton2=new Button();
    public panelNuevo(){
        this.BackColor=Color.LightBlue;
        boton1.Width=30;boton2.Width=30;
        boton1.Height=30;boton2.Height=30;
        boton2.Left =30;
        this.Controls.Add(boton1);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

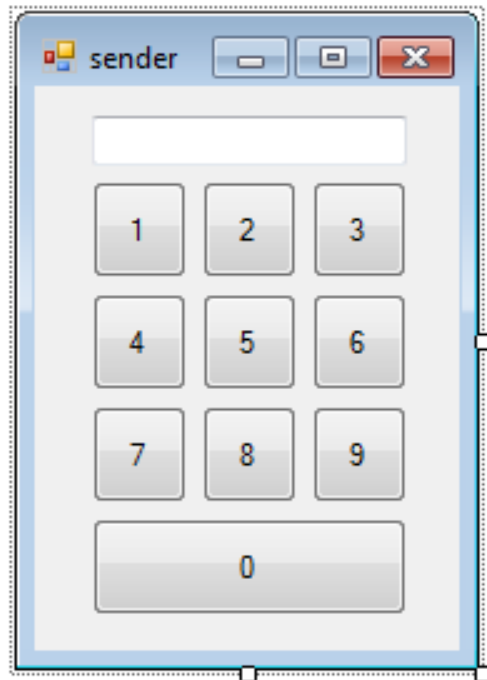
```
void Button1Click(object sender, EventArgs e)
{
    Form2 f=new Form2();
    if (f.ShowDialog()==DialogResult.OK)
        MessageBox.Show("Aceptaron");
}
```

Ejecute y pruebe

Parámetro Sender

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

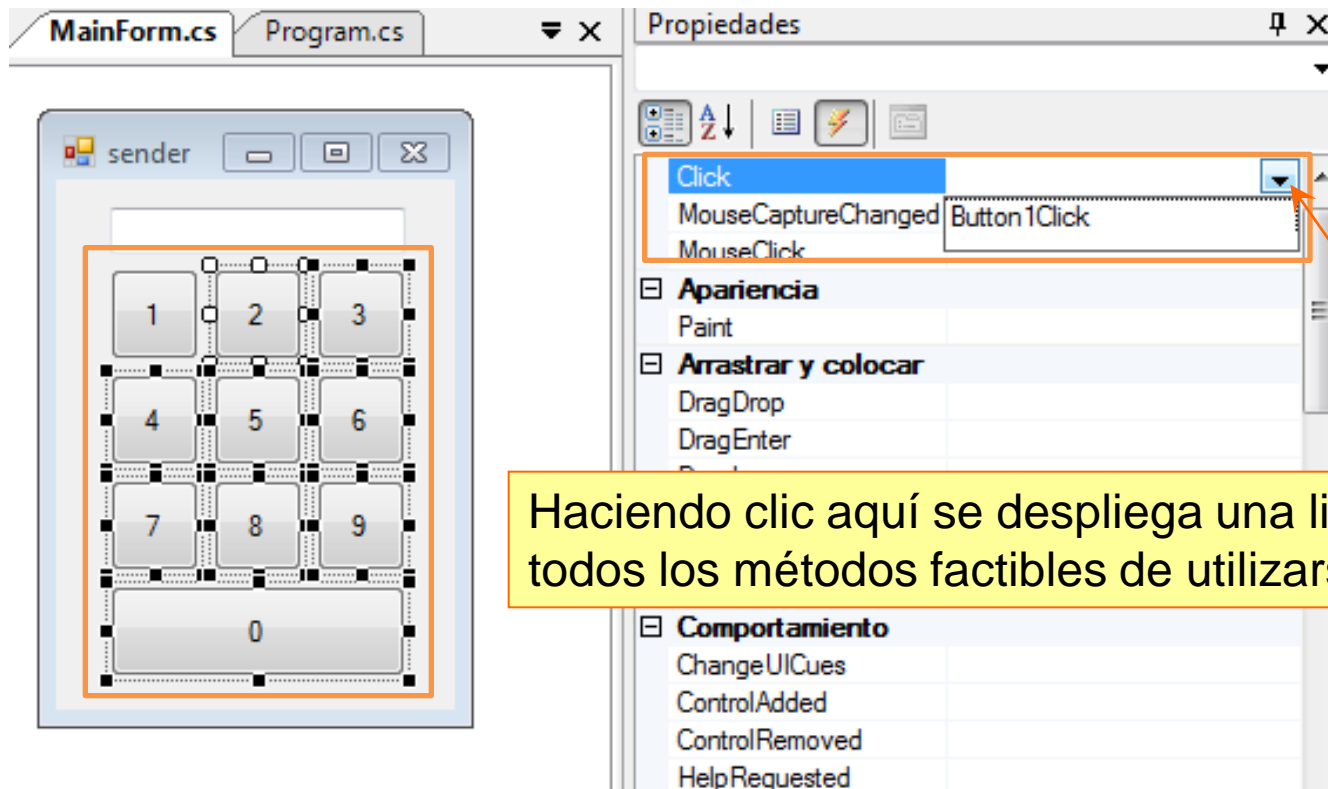
MainForm.cs Program.cs MainI



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;
}
```

An orange arrow points from the text box in the first callout to the 'sender' parameter in the code. Another orange arrow points from the text box in the second callout to the '(sender as Button)' casting expression in the code.

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