ENVÍO DE DATOS ENTRE FORMULARIOS

Guía de laboratorio – Capítulo VII

Contenidos:

- ✓ Uso de constructores.
- ✓ Uso de propiedades de clases.
- ✓ Métodos Show, ShowDialog y Exit.
- ✓ Modificadores de acceso.
- ✓ Tipo Static.

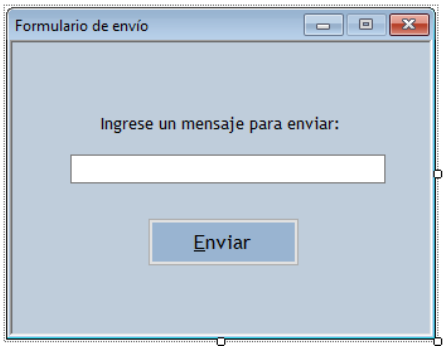
FORMAS DE COMUNICACIÓN ENTRE FORMULARIOS

Existen múltiples formas de enviar y recibir datos de diferentes variables entre formularios.

En esta guía presentaremos algunas de ellas, las cuales podrán realizar con los contenidos acordes a este segundo año de la carrera.

Cabe resaltar que no es lo mismo un formulario llamado mediante el método Show, que un formulario modal, llamado desde ShowDialog.

Para comenzar crearemos un nuevo proyecto, con una GUI similar a la siguiente:

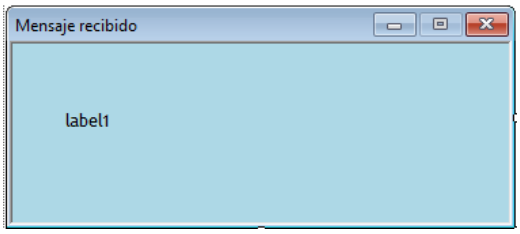


Para ello utilizaremos los siguientes controles y modificaremos algunas de sus propiedades:

Form1	
Name	frmInicio
Text	Pantalla de inicio
StartPosition	CenterScreen
FormBorderStyle	Fixed3D
BackColor	InactiveCaption
MaximizeBox	False
MinimizeBox	True

ControlBox	True
Font	Trebuchet MS; 9,75pt
ForeColor	Black
WindowState	Normal
Label1	
Name	Label1
Text	Ingrese un mensaje para enviar:
Button1	
Name	btnEnviar
Text	&Enviar
Font: Size	12
BackColor	ActiveCaption
TextBox1	
Name	txtMensaje
Text	""

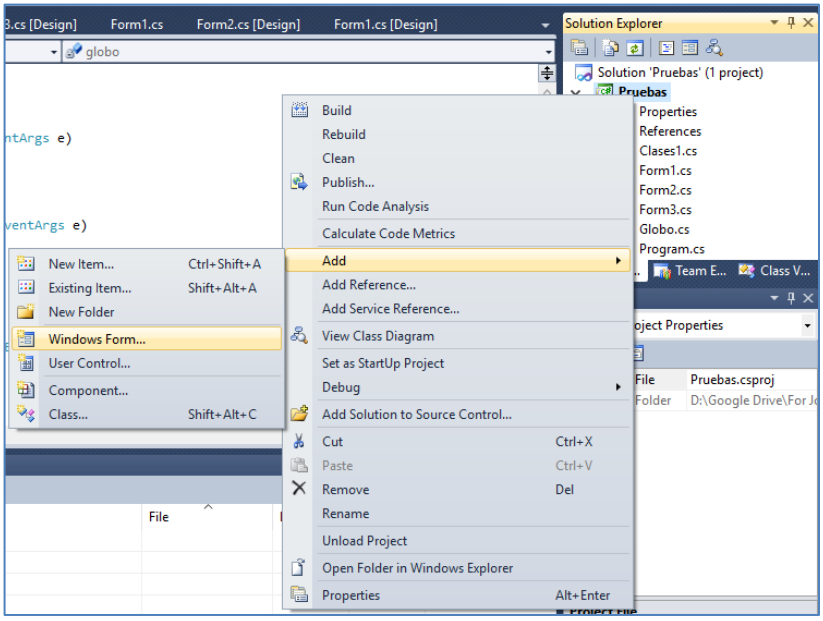
Crearemos también un segundo formulario, que se vea de la siguiente forma, con los controles y propiedades que establecemos a continuación:



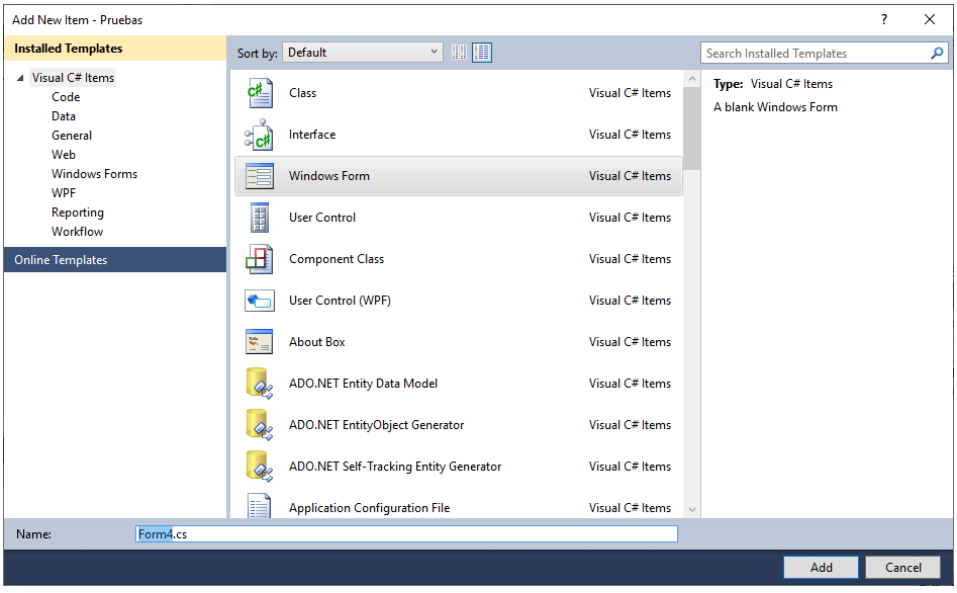
Form2	
Name	frmSecundario
Text	Mensaje recibido
StartPosition	CenterScreen
FormBorderStyle	Fixed3D
BackColor	LightBlue
MaximizeBox	False

Font	Trebuchet MS; 9,75pt
Label1	
Name	lblMensaje
Text	(default)

Para crear un nuevo formulario debemos hacer clic derecho sobre el nombre del proyecto en el explorador de soluciones, luego en “Agregar” (Add) y seleccionamos “Formulario de Windows” (Windows Form…):



Reemplazan el nombre por defecto (“Form4” en este caso) y presionan “Agregar” (Add):



PRIMERA FORMA: FORMULARIO MODAL (ShowDialog). USO DE CONSTRUCTORES

Teniendo en cuenta que para "mostrar" un formulario desde otro, necesitamos crear una instancia del mismo y mediante el método Show o ShowDialog aparecerá en pantalla.

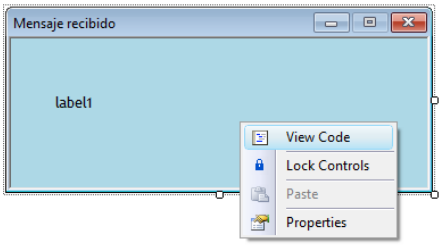
Al crear una instancia, instantáneamente invocamos al **constructor** de la clase. Este es un método que se ejecuta inicialmente y en forma automática. El constructor tiene las siguientes características:

- Tiene el mismo nombre de la clase.
- Es el primer método que se ejecuta.
- Se ejecuta en forma automática.
- No puede retornar datos.
- Se ejecuta una única vez.
- Un constructor tiene por objetivo inicializar atributos.

Sintaxis de un Constructor en C#

```
Modificador NombredelaClase (Parámetros)
{
  Instrucciones
}
```

Para comenzar, nos situamos en el **frmSecundario** y hacemos click derecho sobre él para acceder al código:




```
namespace ComunicacionEntreForms
{
    public partial class frmSecundario : Form
    {
        public frmSecundario()//constructor
        {
            InitializeComponent();
        }
    }
}
```

Lo que haremos es modificar el constructor, para que, al momento de invocarlo cuando creemos una instancia del **frmSecundario**, podamos enviar mediante un parámetro, el valor deseado. El constructor quedaría así:

```
namespace ComunicacionEntreForms
{
    public partial class frmSecundario : Form
    {
        public frmSecundario(string pMensaje)//constructor
        {
            InitializeComponent();
            lblMensaje.Text = pMensaje; //se asigna el valor del parámetro al label
            que mostrará el mensaje recibido
        }
    }
}
```

Lo que hicimos fue modificar las líneas de código que inicializan el formulario frmSecundario de forma tal que el mismo desde el inicio reciba un valor string y el mismo se muestre automáticamente en el lblMensaje.

Es importante tener en cuenta que, al usar esta técnica, toda asignación de variables, propiedades o cualquier línea de código a agregar dentro del constructor **debe hacerse debajo de la línea "InitializeComponent();"**, ya que esta función inicializa todos los controles y sus propiedades, por lo que si intentamos utilizar alguno de estos antes de inicializarlos el programa generará errores.

Ahora bien, teniendo el constructor modificado de la forma deseada, vamos nuevamente al **frmInicio** y en el evento Click del **btnEnviar**, agregaremos las siguientes líneas de código:

```
        private void btnEnviar_Click(object sender, EventArgs e)
        {
            //creamos la instancia del frmSecundario y enviamos el mensaje como
            parámetro
            frmSecundario Form2 = new frmSecundario(txtMensaje.Text);

            //mostramos de forma modal el frmSecundario
            Form2.ShowDialog();
        }
```

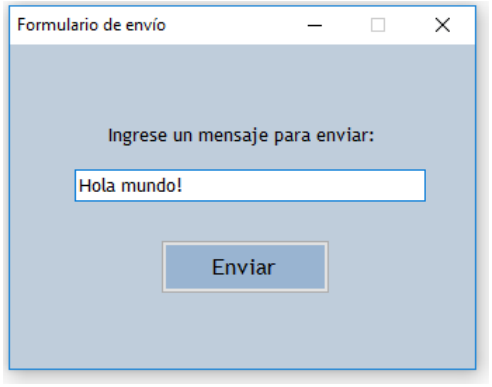
Notarán que como parámetro usaremos el texto que se encuentre dentro de la propiedad Text del **txtMensaje**.

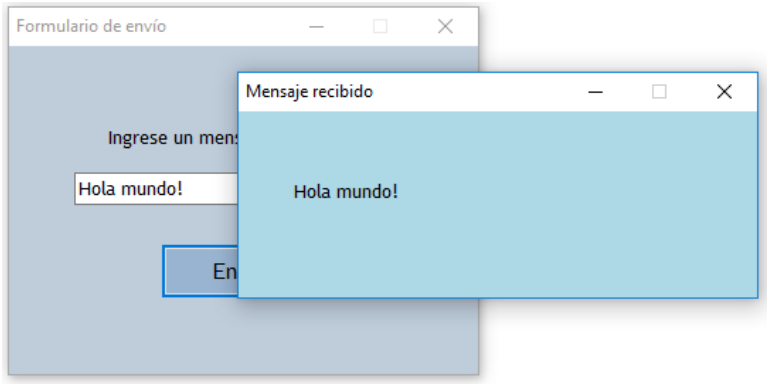
La ventaja de plantear un constructor en lugar de definir un método con cualquier nombre es que se llamará de forma automática cuando se crea un objeto de esta clase:

```
frmSecundario Form2 = new frmSecundario(txtMensaje.Text);
```

Cuando se crea el objeto Form2 se llama al método constructor, el cual recibe el mensaje escrito en txtMensaje.

Como resultado, en la ejecución del programa obtendremos lo siguiente:





SEGUNDA FORMA: FORMULARIO MODAL (ShowDialog). USO DE PROPIEDADES I (get, set).

Otra forma de enviar valores a otro formulario es usando propiedades.

Una **propiedad** es un miembro que pertenece a una clase y que nos permite obtener (get) o asignar un valor (set). Tanto **get** como **set** se denominan descriptores de acceso.

Las propiedades se declaran en el bloque de clase especificando el nivel de acceso del campo, seguido del tipo de propiedad, seguido del nombre de la propiedad y seguido de un bloque de código que declara un descriptor de acceso get, más el descriptor de acceso set, o sólo alguno de los dos.

A veces es conveniente excluir uno de los descriptores de acceso. Omitir el descriptor de acceso set, por ejemplo, hace que la propiedad sea de solo lectura. En el caso de omitir el descriptor de acceso get, sería de sólo escritura.

Es este caso vamos a utilizar una propiedad de solo escritura:

Procederemos a acceder al código del **frmSecundario** y descartaremos lo hecho con la forma anterior, de modo que el resultado sea el siguiente:

```
public partial class frmSecundario : Form
{
    public frmSecundario()//constructor por default sin parámetros
    {
        InitializeComponent();
    }

    private string mensaje;//variable privada de tipo string
```

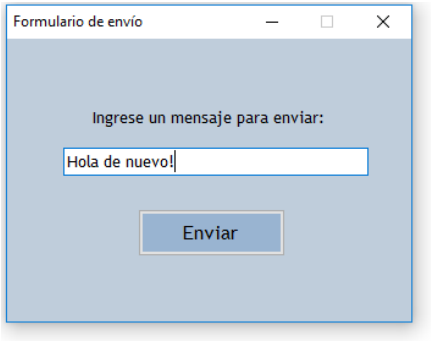
```
public string Mensaje //propiedad pública de sólo escritura
{
    set
    {
        //asigno a la variable el valor obtenido por el método set
        mensaje = value;
        //asigno el valor al lblMensaje
        lblMensaje.Text = mensaje;
    }
}
```

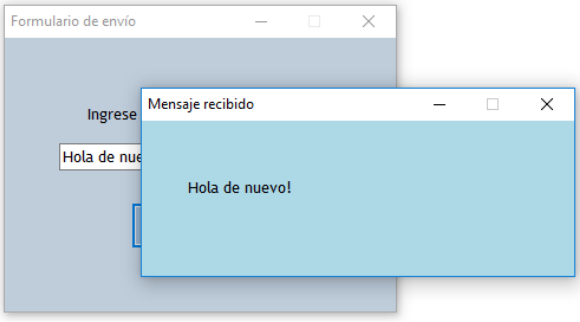
Luego nos situamos en el **frmInicio** y el código a implementar sería:

```
private void btnEnviar_Click(object sender, EventArgs e)
{
    //creamos la instancia del frmSecundario de forma normal
    frmSecundario Form2 = new frmSecundario();

    //asignamos el "value" del método set
    Form2.Mensaje = txtMensaje.Text;

    //mostramos de forma modal el frmSecundario
    Form2.ShowDialog();
}
```

Lo que estamos haciendo con estas líneas de código, es crear una propiedad del objeto Form2 que reciba el valor que se encuentra dentro de la propiedad Text del **txtMensaje** del **frmInicio** y que se muestre en el **lblMensaje** del **frmSecundario**, por lo que no es necesario devolver el valor de la propiedad al formulario inicial. Es por eso que la propiedad es de sólo escritura.

El resultado obtenido sería el siguiente:





En el caso de necesitar utilizar el valor de dicha propiedad del Form2 en el formulario inicial, la propiedad se convierte en lecto-escritura agregándole el método **get**:

```
public string Mensaje //propiedad de lecto-escritura
{
    set
    {
        //asigno a la variable el valor obtenido por el método set
        mensaje = value;
        //asigno el valor al lblMensaje
        lblMensaje.Text = mensaje;
    }
    //método que devuelve el valor actual de la propiedad
    get { return mensaje; }
}
```

Entonces desde donde se está trabajando con la instancia del Form2, podremos utilizar dicha propiedad de la siguiente forma:

```
label2.Text = Form2.Mensaje; //por ejemplo
```

En el caso de no existir el método get declarado en la propiedad, esta línea de código generará un error similar a “La propiedad o el indexador 'ComunicacionEntreForms.frmSecundario.Mensaje' no se puede usar en este contexto porque carece de acceso al descriptor de acceso get”.

TERCERA FORMA: METODO SHOW.

Utilizando el método Show en vez de ShowDialog, crearemos una instancia no modal del frmSecundario con la que podremos interactuar al mismo tiempo que con el frmInicio, al contrario de ShowDialog que sólo permite la interacción del formulario modal y requiere cerrar el mismo para volver al formulario de inicio.

Para ejemplificar esta forma, agregaremos al frmInicio un nuevo button con las siguientes propiedades:

Button2	
Name	btnActualizar
Text	&Actualizar
Font: Size	12
BackColor	ActiveCaption
Enabled	False

En el evento Click del btnEnviar, agregaremos algunas líneas de código y modificaremos otras. El código completo del evento sería el siguiente:

```
        private void btnEnviar_Click(object sender, EventArgs e)
        {
            //creamos la instancia del frmSecundario
            frmSecundario Form2 = new frmSecundario();

            //asignamos el "value" del método set
            Form2.Mensaje = txtMensaje.Text;

            //deshabilitamos el botón enviar para no abrir nuevas instancias del
            frmSecundario
            btnEnviar.Enabled = false;

            //habilitamos el botón actualizar para renovar los mensajes
            btnActualizar.Enabled = true;

            //mostramos el frmSecundario
            Form2.Show();
        }
```

Lo que hacemos, además de crear la instancia del frmSecundario y cargar el valor de la propiedad Mensaje, es deshabilitar el mismo btnEnviar ya que el método utilizado para desplegar la instancia Form2 es el método Show, por lo que cada vez que se presione dicho button, se estaría creando y mostrando una nueva instancia del formulario y sólo necesitamos uno. *(Pueden probar qué ocurre si no lo deshabilitamos.)*

Para poder continuar trabajando con el programa y actualizar el mensaje, también en este evento se habilita el bynActualizar (previamente deshabilitado mediante las propiedades).

Posteriormente necesitaremos codificar el evento Click del btnActualizar, pero como necesitamos trabajar con la misma instancia del frmSecundario, vamos a cambiar de lugar la línea de código que crea el objeto Form2, lo ubicaremos sobre, o debajo, del constructor del frmInicio:

```
    public partial class frmInicio : Form
    {
        //creamos la instancia del frmSecundario
        frmSecundario Form2 = new frmSecundario();

        public frmInicio()
        {
            InitializeComponent();
        }
```

[...]

De esta forma, dentro de ambos eventos Click, así como dentro de cualquier otra función, podremos utilizar dicha instancia.

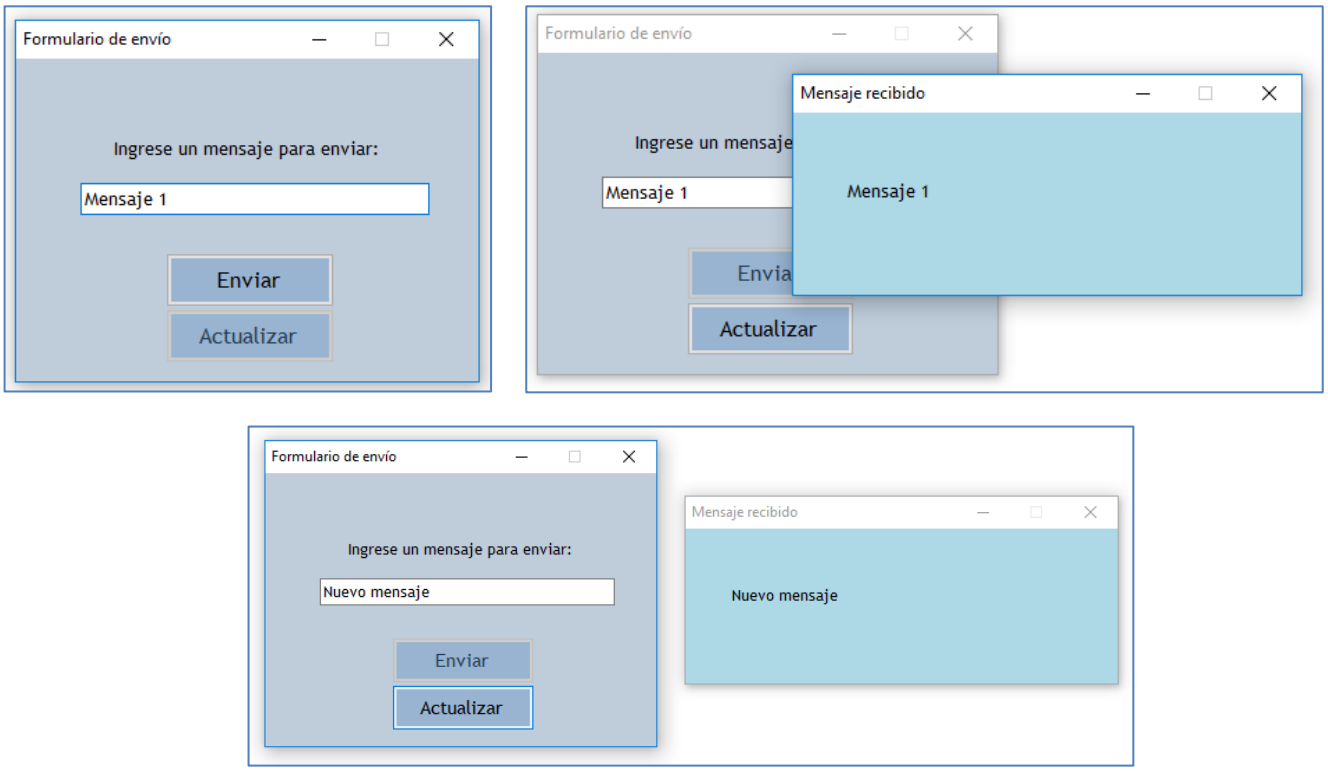
Ahora sí, en el evento Click de btnActualizar crearemos la siguiente línea de código:

```
        private void btnActualizar_Click(object sender, EventArgs e)
        {
            //asignamos el "value" del método set
            Form2.Mensaje = txtMensaje.Text;
        }
```

Y para que al cerrar cualquiera de ambos formularios se cierre todo el programa, agregaremos al evento FormClosing del frmSecundario la siguiente línea de código:

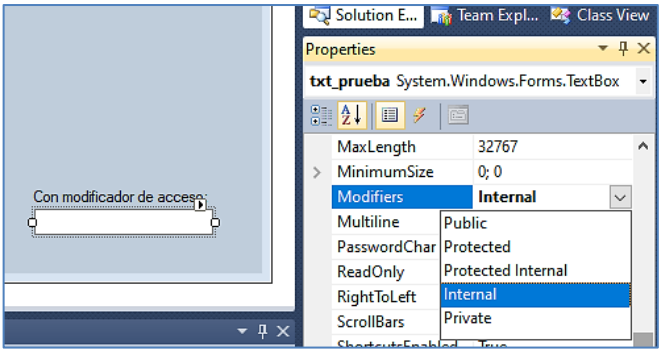
```
private void frmSecundario_FormClosing(object sender, FormClosingEventArgs e)
    {
        Application.Exit();
    }
```

El resultado de nuestro programa será el siguiente:



CUARTA FORMA: MODIFICADORES DE ACCESO.

Una de las formas más sencillas de poder intercambiar información entre diferentes clases, es utilizando la propiedad "**Modifiers**" que se encuentra en el panel derecho de Propiedades, al hacer clic sobre cualquier objeto del formulario en edición:

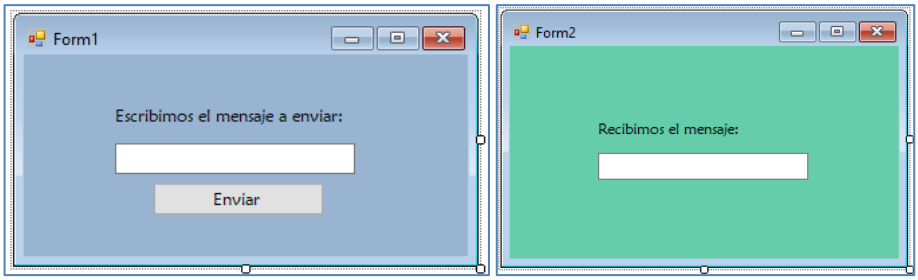


Esta propiedad habilita o no la accesibilidad a dicho componente desde los diferentes espacios en nuestro programa.

Por defecto, Visual Studio siempre inicia la propiedad **Modifiers** con el valor ***Private***. A continuación, una breve descripción de los diferentes tipos de "Modificadores de acceso" que podemos seleccionar o aplicar a cualquier objeto de nuestro proyecto:

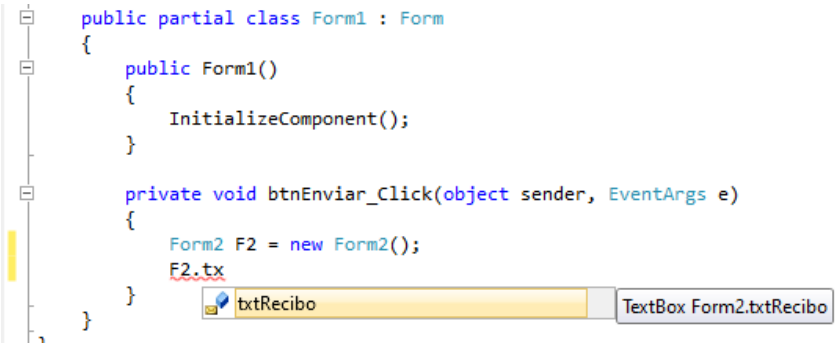
Public: Cualquier clase y método puede utilizar el elemento
Private: Solo la clase que lo declara lo puede usar (todos los objetos tienen este modificador de acceso por defecto)
Protected: La clase que lo define y las subclases que hereden de esa clase, pueden acceder a el
Internal: La clase que lo define y las otras clases en el mismo ensamblado (en el mismo proyecto) pueden usarlo
Protected Internal: Las clases que heredan sea el ensamblado que sea, y las otras clases en el mismo ensamblado pueden acceder.

Probemos esto con un ejemplo: vamos a crear dos formularios:



Seleccionamos el textBox (txtEnvio) del Form2 y cambiamos su propiedad **Modifiers** a "**Internal**" ya que con esto es suficiente para acceder desde cualquier clase de nuestro proyecto.

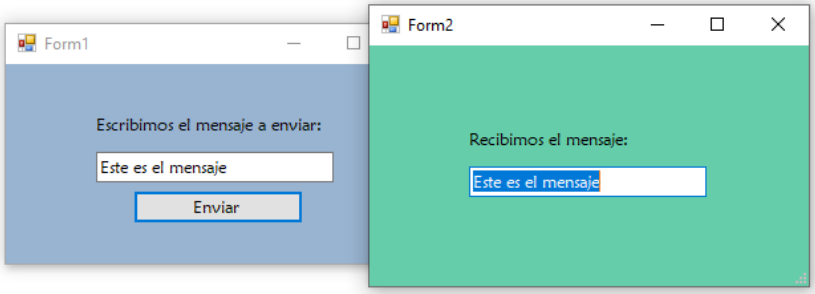
Luego accedemos a la clase del Form1 y en el evento Click del button creamos la instancia del Form2:



Podemos ver como se nos habilita el acceso a dicho objeto, para terminar, asignándole el texto del textBox anterior (del primer formulario) y finalmente muestro el segundo Form:

```
private void btnEnviar_Click(object sender, EventArgs e)
{
    Form2 F2 = new Form2();
    F2.txtRecibo.Text = txtEscribo.Text;
    F2.ShowDialog();
}
```

El resultado es el siguiente:

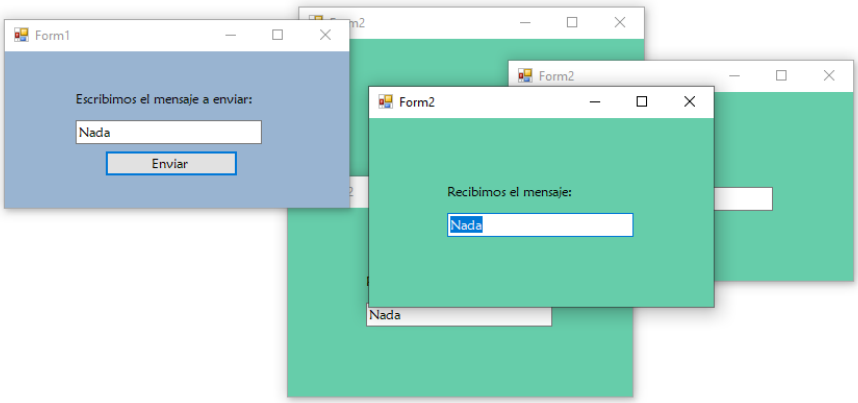


QUINTA FORMA: TIPO STATIC.

Otra forma de poder accede a valores de otras clases o de objetos durante la ejecución del programa es con el tipo de dato “Static”, el cual permite, como bien lo indica su nombre, establecer el valor de dicho objeto o clase de forma estática.

Retrocedamos un poco. Como bien sabemos, cuando utilizamos esta línea de código: `Form2 F2 = new Form2();` estamos creando una instancia NUEVA del Form2. Lo que implica que cada componente, cada variable y método que contenga, se inician nuevamente desde cero, porque de hecho, se están creando desde cero.

Si utilizamos el método Show() con un botón para mostrar el Form2 y lo presionamos varias veces, tendríamos varias instancias del Form2 en la pantalla:



Para evitar esto, utilizamos el método ShowDialog() o al menos controlamos que no exista una instancia ya abierta del Form2.

Pero como bien dijimos, cuando queremos abrir nuevamente el Form2, ya no contamos con ningún valor trabajado en otra instancia de Form2. Por lo que, si creamos una variable de tipo pública en el Form2 y le asignamos un valor, al momento de abrir dicha instancia vamos a verlo, el problema surge cuando esta instancia se cierra u oculta, pasamos utilizar un Form3 y queremos acceder al valor de dicha variable, para eso debemos crear nuevamente una instancia en la clase del Form3, lo cual descarta por completo el valor de la variable a la que queremos acceder.

Si a esta variable le indicamos que es de tipo static, no importa cuántas instancias del Form2 creemos en la ejecución, el valor no se eliminará en ningún momento si no lo indicamos nosotros.

Modificaremos el programa anterior para ejemplificar este tipo de dato:

```
public Form2()
{
    InitializeComponent();
}

public static string variableEstatica;

private void Form2_Load(object sender, EventArgs e)
{
    txtRecibo.Text = variableEstatica;
}
```

De esta forma estamos declarando la variable de tipo Static y en el evento Load del Form2, le asignamos su valor al textBox.

Desde el Form1 accedemos a ella de la siguiente forma:

```
private void btnEnviar_Click(object sender, EventArgs e)
{
    Form2 F2 = new Form2();
    Form2.variableEstatica = txtEscribo.Text;
    F2.Show();
}
```

Parece todo conocido, pero en la línea 2 del código, al llamar a dicha variable, no estamos usando la instancia creada (F2), sino que directamente estamos nombrando al Form2 y como si fuera una propiedad, nos va a permitir acceder a dicha variable directamente. Esto es por su condición de static, no depende de ninguna instancia creada.

7.1. **Ejercicio:** Crear un programa tal que, con el uso de 3 formularios, el usuario pueda ir avanzando en la compra de entradas de cine. Se podrá volver al formulario 1 y 2 las veces que lo desee para modificar sus elecciones.

Primer formulario: Selección de **película**, tipo de **sala**, **horario**, **fecha** y **asientos**.

- ✓ En este formulario deberán diseñar una interfaz intuitiva que permita seleccionar las opciones mencionadas. Las mismas las deberán pre-cargar en etapa diseño o por código.

Segundo formulario: Datos personales.

- ✓ En este formulario deberán permitir la carga de los datos normalmente solicitados a la hora de comprar online (ver ejemplos, agregar todas las validaciones que requiera el programa). Esto incluye la forma de pago como los datos de la tarjeta, opción MercadoPago, etc.

Tercer formulario: Compra.

- ✓ En este formulario deberán mostrar un resumen DETALLADO de la compra. Que incluirá todos los datos del primer formulario y los del segundo para finalmente poder confirmar la compra. De no confirmar, se vuelve al Form1 para una nueva selección.

En lo formularios 2 y 3 deberá aparecer un encabezado que indique el nombre de la película, el día y horario que seleccionó en el Form1.

Es requerimiento excluyente para la aprobación de este ejercicio, que utilicen **al menos 1 (una) propiedad y 1 (un) constructor** para el envío de datos. Pueden usar todas las formas que deseen, pero al menos esas dos deben estar.

Pueden entrar a la web y analizar los pasos para comprar tickets para el cine para evaluar la necesidad de algún dato más que sea necesario, para tener una idea del diseño, etc.