

The background features several large, stylized geometric shapes in the corners. In the top-left and bottom-right, there are dark gray shapes with a white outline, resembling thick chevrons or stylized 'L' shapes. In the top-right and bottom-left, there are lighter gray shapes, also stylized. The central text is positioned in the middle of the slide.

Clase 05

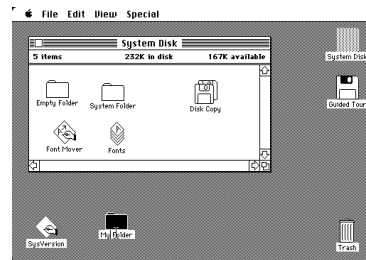
Windows Foms

Que es una interfaz gráfica – GUI

La **Graphical User Interface(GUI)** o interfaz gráfica de usuario fue desarrollada al final de la década de los 70 por el **PARC** (Palo Alto Research Center) y desplegada comercialmente en las Macintosh de Apple y en los sistemas operativos de Microsoft Windows.

Fue diseñada en respuesta al problema de la ineficiente inestabilidad en las antiguas interfaces basadas en líneas de comando para el **usuario promedio**.

Las interfaces gráficas se volverían el estándar para el **diseño centrado en usuarios** en el desarrollo de software, permitiéndole al usuario operar distintos sistemas a través de la manipulación de iconos gráficos como botones, barras de desplazamiento, ventanas, páginas, menús, cursores y el puntero del mouse.

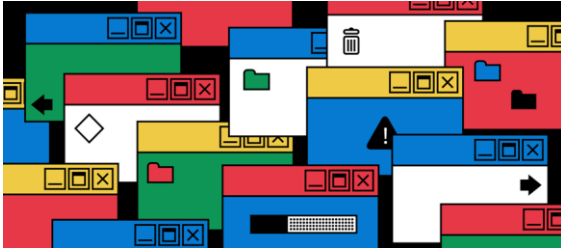


Mac System 1.0
(1984)

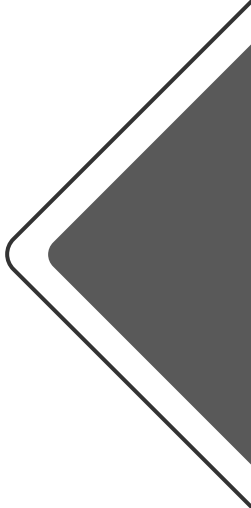
Beneficios de usar una interfaz gráfica –GUI

La gran ventaja de las **GUI** es mejorar la **usabilidad de un sistema para el usuario promedio**, las características de las **GUI** se aprovechan un sentimiento familiar, como el drag-and-drop para transferir archivos y usar iconos familiares como la papelera de reciclaje para eliminar archivos, creando un ambiente donde los sistemas son intuitivos y fácilmente dominados sin ninguna práctica o conocimiento previo de computación o lenguajes de programación

Las **GUI** son autodescriptivas,
las respuestas típicamente son inmediatas
y las ayudas visuales ayudan y **dirigen al usuario**



Windows Forms

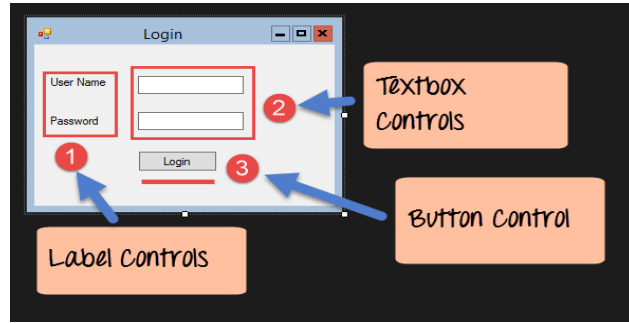


Qué son los Windows Forms

Windows Forms es una librería de clases que provee una Interfaz gráfica de usuario (**GUI**) por el mismo Sistema de .Net, su propósito es proveer a los usuarios una fácil interfaz para desarrollar aplicaciones de escritorio.

Todos los formularios heredan de la clase Form del NameSpace **System.Windows.Forms** y además es posible crear nuestras propias clases a partir de estas.

Una aplicación de **Windows Forms** normalmente tendrá una colección de **controles** como los **labels**, **textboxes**, **ListBoxes**, **buttons**, etc



Esta es una simple aplicación de Windows Forms. Muestra una pantalla de login **accesible para un usuario**. El usuario ingresa las credenciales requeridas y luego le dará **click** al **botón** "login" para proceder



Diseñador De formularios

El Diseñador de formularios(**W**indows **F**orms **D**esigner) provee muchas herramientas para desarrollar nuestras Aplicaciones de escritorio.

Algunas de ellas son:

- Organizar los controles mediante guías de alineación.
- Establecer los márgenes y rellenos de los controles.
- Establecer los valores de propiedad con la ventana propiedades.
- Generación de eventos.
- Copiar/Cortar/Pegar/Eliminar.

Al momento de diseñar un formulario, el **diseñador** de Visual Studio escribe de forma automática el código que describe a cada uno de los controles y al propio formulario.

Clases parciales

El concepto de Clases parciales (**Partial Class**) permite separar el código autogenerated por el diseñador de la lógica de nuestra aplicación en archivos diferentes.

```
using System;
using System.Windows.Forms;

namespace EjemploDeFormulario
{
    3 referencias
    public partial class FrmEjemplo : Form
    {
        1 referencia
        public FrmEjemplo()
        {
            InitializeComponent();
        }

        1 referencia
        private void FrmEjemplo_Load(object sender, EventArgs e)
        {
        }
    }
}
```

← Nuestro formulario donde vamos a desarrollar la **lógica** de la aplicación.

Clases parciales

```
namespace EjemploDeFormulario
{
    3 referencias
    partial class FrmEjemplo
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        0 referencias
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        1 referencia
        private void InitializeComponent()
        #endregion
    }
}
```

← Clase parcial auto generada por el Diseñador donde va a escribir todo el código auto generado por nuestra aplicación.

Clases parciales

```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
1 referencia
private void InitializeComponent()
{
    this.SuspendLayout();
    //
    // FrmEjemplo
    //
    this.AutoScaleMode = new System.Drawing.SizeF(7F, 15F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(629, 279);
    this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
    this.IsMdiContainer = true;
    this.MinimizeBox = false;
    this.Name = "FrmEjemplo";
    this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
    this.Text = "Mi Formulario";
    this.Load += new System.EventHandler(this.FrmEjemplo_Load);
    this.ResumeLayout(false);
}

#endregion
```

← En esta región vamos a encontrar todas las configuraciones iniciales de nuestro objeto Form además de todas las modificaciones y adiciones que se van a generar mientras vayamos generando controles, modificando propiedades y asignando eventos en nuestro Form.



Objeto Form



Objeto Form

El objeto Form es el principal componente de una aplicación Windows.

Algunas de sus propiedades admiten valores de alguno de los tipos nativos (por valor) de .NET.


```
FrmHijo miFrm = new FrmHijo();  
miFrm.ShowInTaskbar = false; //Mostrar en barra de tareas  
miFrm.Text = "Formulario de Alejandro"; //Titulo del FRM  
miFrm.Opacity = 0.85F; //Opacidad para transparencia  
  
//Otras propiedades que requieren asignacion de objetos  
  
miFrm.Size = new System.Drawing.Size(100, 100); // Tamaño en ancho y alto  
miFrm.Location= new System.Drawing.Point(0,0); //Ubicacion en X e Y  
miFrm.Font = new System.Drawing.Font("Arial", 10); //Fuente y tamaño
```

```
miFrm.Font = new System.Drawing.Font("Arial", 10); //Fuente y tamaño  
miFrm.Location= new System.Drawing.Point(0,0); //Ubicacion en X e Y
```



Métodos de mi objeto Form

```
miFrm.Show(); //Visualiza el formulario. Puede especificarse su formulario Owner (dueño o propietario).  
miFrm.ShowDialog(); //Puede utilizar este método para mostrar un cuadro de diálogo modal en la aplicación.  
miFrm.Close(); //Cierra el formulario.  
miFrm.Hide(); //Oculta el formulario del usuario
```



Propiedades

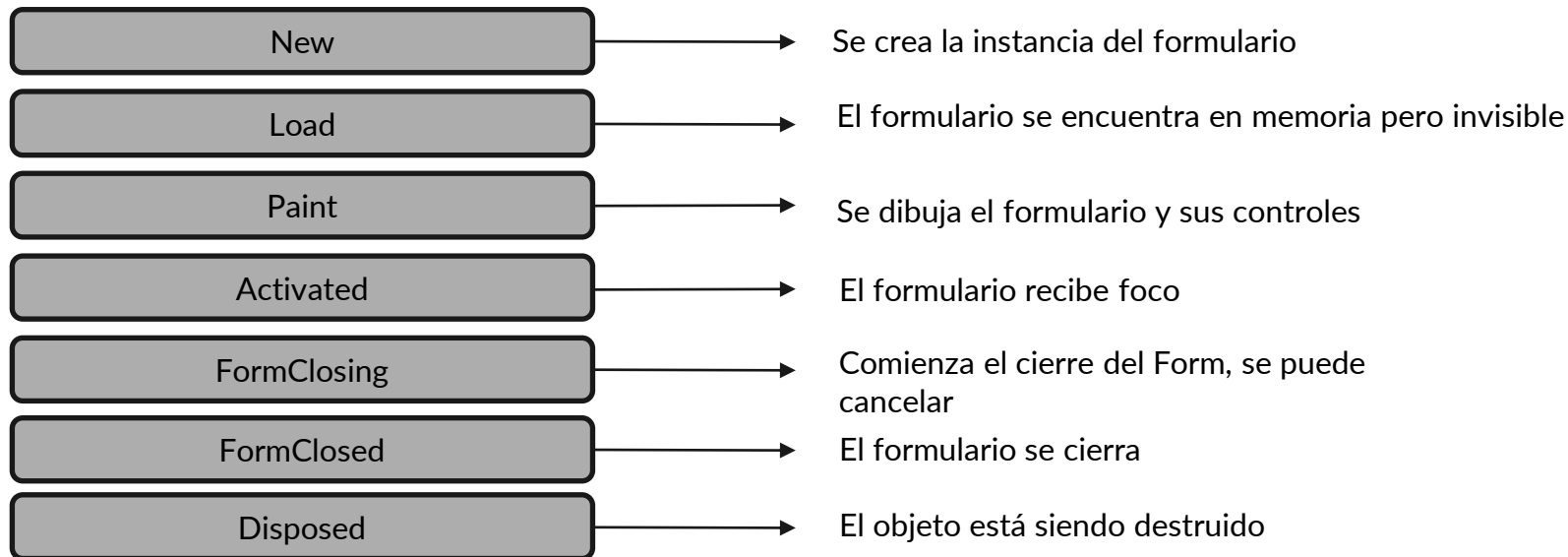
Las propiedades de nuestros formularios nos permiten modificar los valores de nuestros atributos o acceder a ellos. Un control de Windows Forms hereda muchas propiedades de la clase base **System.Windows.Forms.Control**.

Algunas de estas propiedades son:

- ✓ **Name:** Indica el nombre utilizado en el código para identificar el objeto.
- ✓ **BackColor:** Indica el color de fondo del componente.
- ✓ **Cursor:** Indica el cursor que aparece al pasar el puntero por el control.
- ✓ **Enabled:** Indica si el control está habilitado.
- ✓ **Font:** Fuente utilizada para mostrar el texto en el control.
- ✓ **ForeColor:** Color utilizado para mostrar texto.
- ✓ **Locked:** Determina si se puede mover o cambiar el tamaño del control.
- ✓ **Modifiers:** Indica el nivel de visibilidad del objeto.
- ✓ **TabIndex:** Determina el índice de tabulación que ocupará este control.
- ✓ **Text:** Texto asociado al control.
- ✓ **Visible:** Determina si el control está visible u oculto.
- ✓ **BackgroundImage:** Nos permite asignarle una imagen de fondo a nuestro control
- ✓ **StartPosition:** Determina la posición del formulario cuando aparece por primera vez
- ✓ **Icon:** Indica el icono del formulario

Ciclo de vida de un formulario

El objeto Form está compuesto por una gran cantidad de eventos que fueron heredados de la clase **System.Windows.Forms**, Muchos de estos eventos corresponden al ciclo de vida del Form



Eventos de ciclo de vida

1 referencia

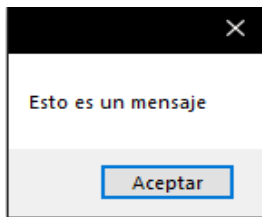
```
private void FrmEjemplo_FormClosing(object sender, FormClosingEventArgs e)
{
    if(e.CloseReason == System.Windows.Forms.CloseReason.UserClosing)
    {
        e.Cancel = true;
    }
}
```

MessageBox

Para mostrar información o pedir una intervención del usuario, podemos utilizar la clase **MessageBox**. Esta clase contiene métodos estáticos que permiten mostrarle un cuadro de mensaje al usuario para interactuar con la aplicación.

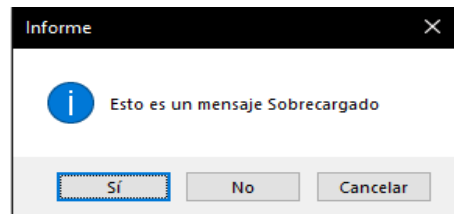
```
MessageBox.Show("Esto es un mensaje");
```

El método Show de la clase MessageBox nos va a permitir mostrarle una ventana de diálogo simple al usuario.



```
MessageBox.Show("Mensaje", "Informe", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Information);
```

El método Show también cuenta con múltiples sobrecargas para darle formatos distintos a sus ventanas emergentes.





Controles



¿Que es un control?

Los **controles** de Windows Forms son componentes reutilizables que encapsulan la funcionalidad de una interfaz gráfica.

Windows Forms provee una gran cantidad de **controles listos para trabajar**, pero también provee la infraestructura para desarrollar controles propios, **extender controles** existentes o combinar controles existentes

Como se añaden controles?

Los controles son añadidos a través de diseñador del visual studio, con el diseñador se pueden alinear modificar su tamaño y moverlos.

Alternativamente los controles pueden ser añadidos a través del código.

```
TextBox txtNuevo = new TextBox()
{
    Location = new System.Drawing.Point(5, 59),
    TabIndex = 0
};
this.Controls.Add(txtNuevo);
```

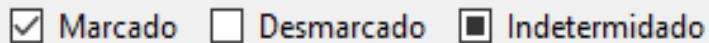
El siguiente código añade y posiciona un control de tipo TextBox

CheckBox

CheckBox

Este **control** muestra una casilla de verificación, que podemos marcar para establecer un estado.

Generalmente el estado de un CheckBox es **marcado** (verdadero) o **desmarcado** (falso), sin embargo, podemos configurar el control para que sea detectado un tercer estado, que se denomina **indeterminado**, en el cual, el control se muestra con la marca en la casilla pero en un color de tono gris.

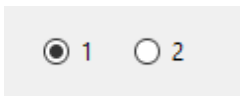


Los tres posibles estados del control.

RadioButton, Groupbox

RadioButton

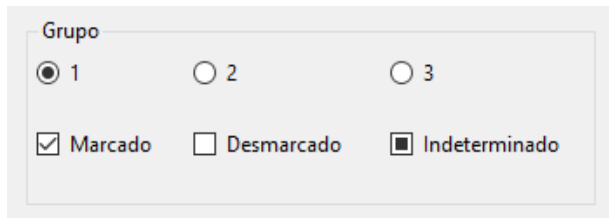
Los **controles** RadioButton nos permiten definir conjuntos de opciones **auto excluyentes**, de modo que situando varios controles de este tipo en un formulario, sólo podremos tener seleccionado **uno en cada ocasión**.



← Ejemplo de dos RadioButton, solo uno de ellos puede estar marcado.

GroupBox

Permite **agrupar** controles en su interior, tanto RadioButton como de otro tipo, ya que se trata de un control contenedor.

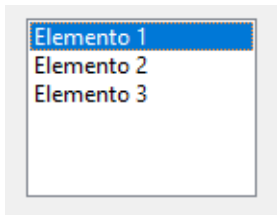


← Ejemplo de distintos controles agrupados en el control GroupBox.

ListBox

ListBox

Un control ListBox contiene una lista de valores, de los cuales, el usuario puede seleccionar uno o varios simultáneamente.



← Ejemplo de un ListBox cargado con 3 elementos de tipo String.

Algunas de sus propiedades más importantes son:

Items. Contiene la lista de valores que visualiza el control. Se trata de un tipo `ListBox.ObjectCollection`, de manera que el contenido de la lista puede ser tanto tipo cadena, numéricos u objetos de distintas clases.

SelectionMode. Establece el modo en el que se pueden seleccionar los elementos de la lista.

Si se guarda un objeto en un ListBox tener en cuenta que el control lo único que mostrará es el método `ToString` de ese objeto.

ComboBox

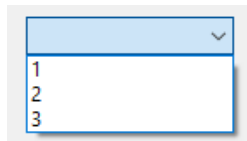
ComboBox

es un control basado en la combinación (de ahí su nombre) de dos controles: TextBox y ListBox.

Un control ComboBox dispone de una zona de edición de texto y una lista de valores, que se pueden desplegar desde el cuadro de edición.

El estilo de visualización por defecto de este control, muestra el cuadro de texto y la lista oculta, aunque mediante la propiedad **DropDownStyle** se puede cambiar.

La propiedad **DropDownStyle** también influye en una diferencia importante de comportamiento entre el estilo **DropDownList** y los demás, dado que cuando se crea un comboBox con el mencionado estilo, el cuadro de texto sólo podrá mostrar información, no permitiendo que esta sea modificada.



Control de tipo
ComboBox cargado con
tres valores iniciales.

Button, TextBox, Label

Button

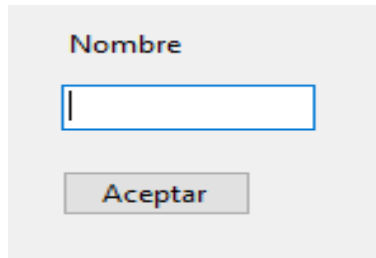
El control permite al usuario hacer un click para crear una acción, cuando el botón es clickeado simula que está siendo apretado y soltado.

TextBox

Este control es usado para ingresar parámetros de entrada por el usuario o mostrar algún texto, el texto puede ser editable como solo lectura.

Label

Este control se utiliza para mostrar información al usuario en un formato de texto que no se puede editar, se usan para identificar objetos en un formulario o proveer una breve descripción de que hace un control.



A screenshot of a web form. At the top, the word "Nombre" is written in a dark blue font. Below it is a white text box with a blue border and a vertical cursor. At the bottom is a button with the word "Aceptar" in a dark blue font.



Ejemplo combinando los 3 controles para pedirle a un usuario que ingrese un nombre.