# APLICACIONES EN WINDOWS FORMS C#

# En esta clase hemos veremos los siguientes temas:

1. Controles de diálogo en C#: veremos los diferentes tipos de controles de diálogo en C# y cómo se pueden utilizar para interactuar con el usuario.
2. Persistencia de datos en archivos: Exploraremos la importancia de la persistencia de datos y cómo se pueden almacenar los datos en archivos de texto delimitados por comas (CSV) utilizando la clase StreamWriter y la clase StreamReader.
3. Almacenamiento de imágenes en archivos: hemos discutido los diferentes formatos de archivo de imagen (BMP, JPEG, PNG, GIF) y cómo almacenan la información de la imagen.

# Los controles de diálogo en C#

1. Son elementos de interfaz gráfica de usuario (GUI) que se utilizan para interactuar con el usuario a través de cuadros de diálogo.
2. Se utilizan comúnmente en aplicaciones para solicitar información al usuario, confirmar acciones o mostrar mensajes de error.
3. Al comprender cómo funcionan los controles de diálogo y cómo integrarlos en sus aplicaciones, los desarrolladores pueden crear una experiencia de usuario más intuitiva y mejorar la usabilidad de sus aplicaciones.

# Tipos de controles de diálogo:

1. Message Box: Este es un cuadro de diálogo que se utiliza para mostrar un mensaje al usuario. El usuario puede hacer clic en un botón para cerrar el cuadro de diálogo y continuar con la aplicación. Los Message Box se utilizan comúnmente para mostrar mensajes de bienvenida, mensajes de error o mensajes de confirmación.

2. Input Box: Este cuadro de diálogo se utiliza para solicitar al usuario que ingrese información en un campo de texto. El cuadro de diálogo generalmente incluye un mensaje que indica al usuario qué tipo de información se espera, como un nombre de usuario o una contraseña.

# MessageBox

- El MessageBox es un cuadro de diálogo estándar que se utiliza para mostrar mensajes al usuario y solicitar acciones o confirmaciones. Al utilizar el MessageBox, es importante considerar los siguientes criterios de usabilidad:

1. **Claridad del mensaje:** El mensaje mostrado en el MessageBox debe ser claro, conciso y fácil de entender para el usuario.
2. **Títulos descriptivos:** El título del MessageBox debe ser descriptivo y reflejar claramente el propósito del cuadro de diálogo.
3. **Tipos de botones apropiados:** El MessageBox permite mostrar diferentes tipos de botones, como "Aceptar", "Cancelar", "Sí/No" o "Aceptar/Cancelar".
4. **No abusar del MessageBox:** El MessageBox debe utilizarse con moderación y solo cuando sea necesario.

# MessageBox YesNo

```
1. if (MessageBox.Show("¿Seguro que desea borrar el registro?",
   "Confirmar borrado", MessageBoxButtons.YesNo) == DialogResult.Yes)
2. {
3.     // Acciones a realizar si el usuario hace clic en "Sí"
4. }
5. else
6. {
7.     // Acciones a realizar si el usuario hace clic en "No"
8. }
```

# El InputBox

1. El InputBox es un control que no está disponible de forma nativa en C#, pero se puede usar a través de la clase Microsoft.VisualBasic.Interaction.
2. En cuanto al mejor criterio para su uso, se recomienda utilizar el InputBox en casos donde se necesita una entrada de datos rápida y sencilla del usuario, como por ejemplo en pequeñas aplicaciones de consola o herramientas de uso interno.
3. Sin embargo, si se está desarrollando una aplicación más compleja con una interfaz gráfica de usuario, es recomendable utilizar controles de entrada personalizados que se adapten mejor a las necesidades de la aplicación.

# Ejemplo InputBox:

```
1. using Microsoft.VisualBasic;
2. string respuesta = Interaction.InputBox("Ingrese un
   valor:", "Entrada de usuario", "");
3. // El primer argumento es el mensaje que se muestra
   en la ventana, el segundo argumento es el título de
   la ventana, y el tercer argumento es el valor
   predeterminado que se muestra en el cuadro de texto
```

# Otros controles muy usados...

- OpenFileDialog: Este es un cuadro de diálogo que se utiliza para permitir al usuario seleccionar un archivo para abrir en la aplicación.
- SaveFileDialog: Este cuadro de diálogo se utiliza para permitir al usuario seleccionar la ubicación y el nombre de un archivo que se guardará desde la aplicación.
- ColorDialog: Este cuadro de diálogo se utiliza para permitir al usuario seleccionar un color para utilizar en la aplicación.
- FontDialog: Este cuadro de diálogo se utiliza para permitir al usuario seleccionar una fuente y un tamaño de fuente para utilizar en la aplicación.

# Cómo usar los controles de diálogo...

- Agrega un control de diálogo a tu proyecto: para agregar un control de diálogo a tu proyecto, puedes arrastrar y soltar el control desde la caja de herramientas en el diseñador de formularios.
- Configura las propiedades del control de diálogo: después de agregar un control de diálogo a tu proyecto, es posible que necesites configurar sus propiedades para que funcione correctamente en tu aplicación. Por ejemplo, puedes establecer el texto del mensaje en un Message Box o especificar el título del cuadro de diálogo en un SaveFileDialog.
- Muestra el control de diálogo: una vez que hayas configurado las propiedades del control de diálogo, puedes mostrarlo en tu aplicación. Para hacerlo, puedes usar el método ShowDialog() del control de diálogo.
- Procesa la respuesta del usuario: después de que el usuario haya interactuado con el control de diálogo, es importante procesar su respuesta en tu aplicación. Por ejemplo, si el usuario hizo clic en el botón OK en un Message Box, puedes realizar una acción específica en tu aplicación en función de su respuesta.

# Persistencia en archivos

# Archivos...

- La persistencia de datos en archivos es una técnica muy utilizada en la programación para almacenar información de forma permanente y poder recuperarla en un momento posterior. En C# existen diversas clases y métodos que permiten trabajar con archivos de distintos tipos y formatos, lo que hace que esta técnica sea muy versátil y adaptable a distintas necesidades.

# Caracteristicas...

- **Flexibilidad:** C# permite trabajar con archivos de distintos tipos y formatos, desde archivos de texto plano hasta archivos binarios y documentos de Office.
- **Permanencia de la información:** Al utilizar archivos para almacenar información, ésta se mantiene disponible incluso después de cerrar la aplicación o reiniciar el sistema.

# Caracteristicas...

- **Facilidad de acceso:** C# proporciona una gran cantidad de métodos y clases que facilitan la lectura y escritura de archivos, lo que permite acceder a la información almacenada de forma rápida y sencilla.
- **Seguridad:** Al utilizar archivos para almacenar información, se pueden aplicar técnicas de seguridad como la encriptación o el uso de contraseñas para proteger la información almacenada.
- **Escalabilidad:** Al utilizar archivos para almacenar información, es posible trabajar con grandes cantidades de datos sin necesidad de contar con recursos adicionales como bases de datos o servidores.

# Archivos CSV...

- Al almacenar la información en un archivo CSV, se puede exportar fácilmente y compartir con otros sistemas o aplicaciones.
- Además, el formato CSV es fácil de leer y editar por humanos, lo que lo hace ideal para trabajar con datos tabulares en hojas de cálculo o editores de texto. Esto significa que los archivos CSV pueden ser utilizados no solo para intercambio de datos entre sistemas, sino también para análisis y manipulación de datos en aplicaciones de usuario final.
- En resumen, el uso de archivos CSV es una forma eficiente y conveniente de intercambiar información entre aplicaciones y sistemas, lo que lo convierte en un formato de archivo popular y útil para el intercambio de datos en una variedad de aplicaciones y entornos.

# Guardar datos en un archivo CSV:

```
using System.IO;

string rutaArchivo = "alumnos.csv"; // Ruta del archivo a guardar
using (StreamWriter writer = new StreamWriter(rutaArchivo))
{
    // Escribir la cabecera del archivo
    writer.WriteLine("Nombre,Apellido,Edad");

    // Escribir cada fila del archivo
    foreach (Alumno alumno in listaAlumnos)
    {
        writer.WriteLine($"{alumno.Nombre},{alumno.Apellido},{alumno.Edad}");
    }
}
```

# Usando file dialog con el archivo CSV…

- En este ejemplo, se crea un objeto StreamWriter para escribir en un archivo llamado "alumnos.csv" (puedes cambiar la ruta y el nombre del archivo según tus necesidades).
- Se escribe la cabecera del archivo (que corresponde a los nombres de las propiedades de la clase Alumno) y luego se escribe cada fila del archivo utilizando la sintaxis de interpolación de cadenas ($"{variable1},{variable2},{variable3}").

- Para utilizar este código, debes tener una lista de objetos Alumno llamada listaAlumnos como la que se creó en el ejemplo anterior.

# Abriendo el archivo CSV…

- En este ejemplo, se utiliza un control DataGridView en el formulario para mostrar los datos del archivo de texto delimitado por comas que se carga con el control OpenFileDialog.
- En la primera línea del archivo se encuentran los encabezados de las columnas que se utilizan para crear las columnas del control DataGridView.
- Luego, se leen las líneas restantes del archivo y se separan los campos utilizando el método Split. Los datos se agregan al DataGridView mediante el método Rows.Add.

# Abriendo el archivo CSV:

```
using System.IO;

 private void btnOpen_Click(object sender, EventArgs e)
        {
            // Configurar el control Open File Dialog
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "Archivos de texto (*.txt)|*.txt|Todos los archivos (*.*)|*.*";
            openFileDialog.Title = "Abrir archivo de texto";
            openFileDialog.ShowDialog();

            // Leer el contenido del archivo seleccionado y cargarlo en el DataGridView
             if (openFileDialog.FileName != "")
             {
                 using (StreamReader sr = new StreamReader(openFileDialog.FileName))
                 {
                     // Leer la primera línea del archivo para obtener los encabezados de las columnas
                     string headerLine = sr.ReadLine();
                     string[] headers = headerLine.Split(',');

                    // Crear las columnas del DataGridView
                     foreach (string header in headers)
                     {
                         dataGridView1.Columns.Add(header, header);
```

# Abriendo el archivo CSV:

```
21:                 }
22:
23:                 // Leer las líneas restantes del archivo y agregarlas al DataGridView
24:                  string line;
25:                  while ((line = sr.ReadLine()) != null)
26:                  {
27:                      string[] fields = line.Split(',');
28:                      dataGridView1.Rows.Add(fields);
29:                  }
30:              }
31:          }
```

# Grabando el archivo CSV…

- En el botón "Guardar archivo" se utiliza el control SaveFileDialog para obtener la ubicación y el nombre del archivo de texto delimitado por comas donde se guardarán los cambios realizados en el DataGridView.
- Luego, se escriben los encabezados de las columnas en la primera línea del archivo y los datos de cada fila en las líneas subsiguientes, utilizando el método Write para escribir los datos y el método

# Abriendo el archivo CSV:

```
private void btnSave_Click(object sender, EventArgs e)
      {
          // Configurar el control Save File Dialog
          SaveFileDialog saveFileDialog = new SaveFileDialog();
          saveFileDialog.Filter = "Archivos de texto (*.txt)|*.txt|Todos los archivos (*.*)|*.*";
          saveFileDialog.Title = "Guardar archivo de texto";
          saveFileDialog.ShowDialog();

          // Escribir el contenido del DataGridView en un archivo de texto delimitado por comas
          if (saveFileDialog.FileName != "")
           {
              using (StreamWriter sw = new StreamWriter(saveFileDialog.FileName))
              {
                  // Escribir los encabezados de las columnas en la primera línea del archivo
                  foreach (DataGridViewColumn column in dataGridView1.Columns)
                  {
                      sw.Write(column.HeaderText + ",");
                  }
                  sw.WriteLine();

                 // Escribir los datos de cada fila en el archivo
                  foreach (DataGridViewRow row in dataGridView1.Rows)
                  {
```

# Abriendo el archivo CSV:

```
                    foreach (DataGridViewCell cell in row.Cells)
                        {
                            sw.Write(cell.Value.ToString() + ",");
                        }
                        sw.WriteLine();
                    }
                }
            }
        }
    }
}
```

# Preguntas?

# Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash