"La Ciencia sin Moral es Vana"

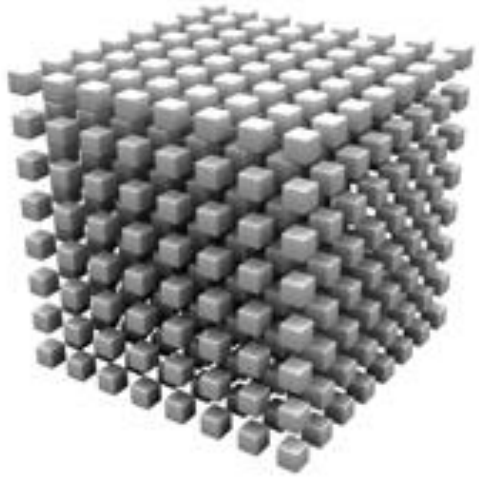
Universidad Católica de El Salvador

Facultad de Ingeniería y Arquitectura

Materia: Programación II

Docente: Master Giovanni Acosta.

Tema 6: Colecciones, estructuras y enumeraciones

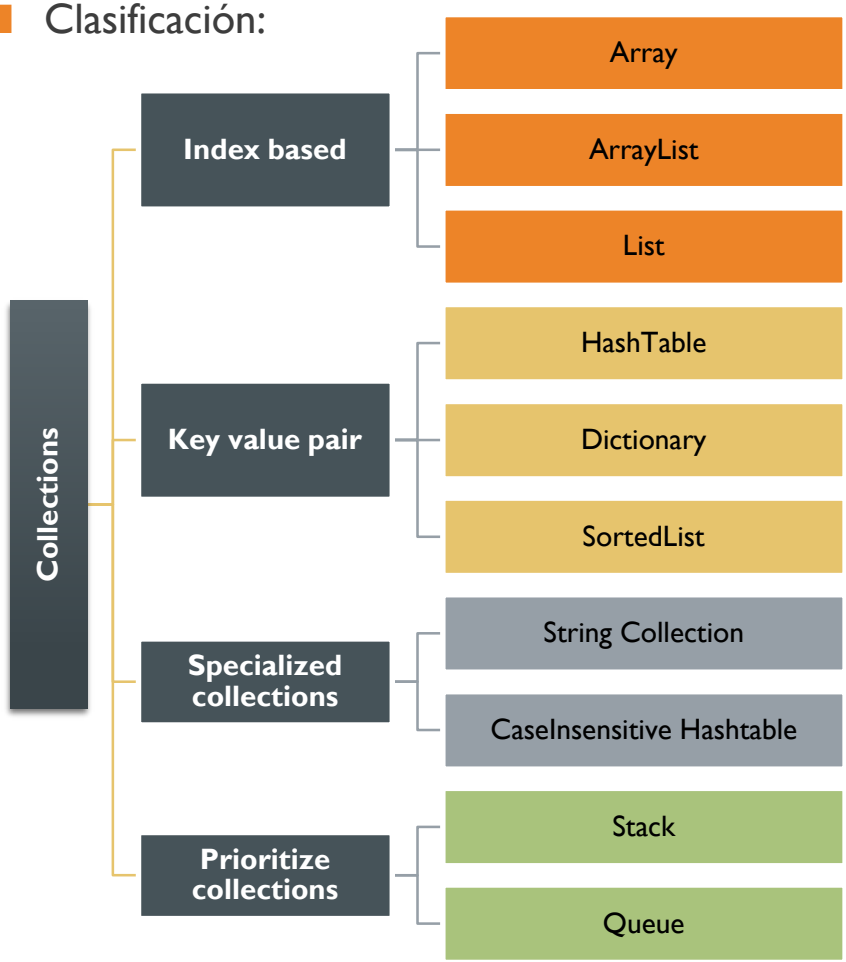


Objetivos:

- Almacenar datos en colecciones: ArrayList, List y Dictionary
- Crear estructuras de datos y enumeraciones
- Desarrollar aplicaciones utilizando colecciones, estructuras de datos y enumeraciones

¿QUÉ ES UNA COLECCIÓN?

- Una **colección** agrupa un conjunto de objetos relacionados.
- Clasificación:



Espacio de nombres para **colecciones:**

1. https://msdn.microsoft.com/es-es/library/system.collections(v=vs.110).aspx
2. https://msdn.microsoft.com/es-es/library/system.collections.generic(v=vs.110).aspx
3. https://msdn.microsoft.com/es-es/library/system.collections.specialized(v=vs.110).aspx

COLECCIÓN ARRAYLIST

- La colección **ArrayList** del .NET Framework imita la funcionalidad de los arreglos convencionales y proporciona la capacidad de modificar el tamaño de la colección en forma dinámica, a través de sus métodos.
- Un **ArrayList** puede almacenar objetos de cualquier tipo.

- ¿Cómo se usa ArrayList en C#?

1. Agregar el espacio de nombres **Collections**;

```
using System.Collections;
```

2. Sintaxis para declarar y crear la colección **ArrayList**:

```
ArrayList nombreLista = new ArrayList();
```

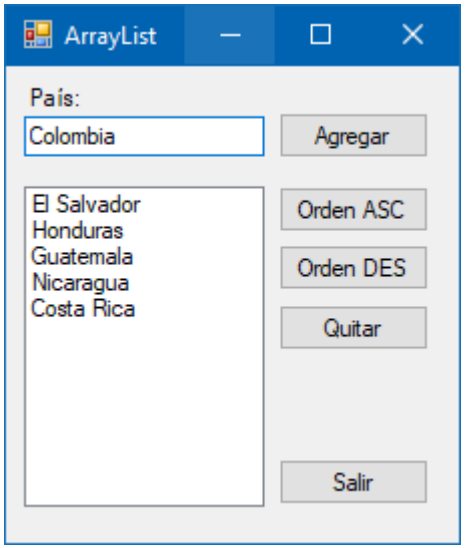
MÉTODOS Y PROPIEDADES DE ARRAYLIST

Método o propiedad	Descripción
Capacity	Obtiene y establece el número de elementos para los que se reserva espacio en un momento dado, dentro del objeto ArrayList
Add	Agrega un nuevo elemento al ArrayList
Clear	Elimina todos los elementos del objeto ArrayList
Contains	Devuelve true si el elemento especificado está en el objeto ArrayList
Count	Retorna el número de elementos almacenados en el objeto ArrayList
IndexOf	Devuelve el índice de la primera ocurrencia del elemento especificado en el objeto ArrayList
Insert	Inserta un elemento en el índice especificado
Remove	Elimina la primera ocurrencia del elemento especificado
RemoveAt	Elimina un objeto en el índice especificado
Sort	Ordena el objeto ArrayList

Para consultar otros métodos de **ArrayList**: https://docs.microsoft.com/en-us/dotnet/api/system.collections.arraylist?view=netframework-4.8

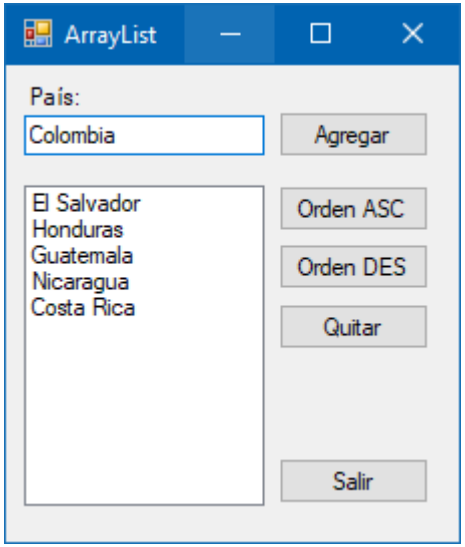
DEMO: ARRAYLIST

```
using System.Collections;


private static ArrayList paises = new ArrayList();


private void frmArrayList_Load(object sender, EventArgs e)
{
    paises.Add("El Salvador");
    paises.Add("Honduras");
    paises.Add("Guatemala");
    paises.Add("Nicaragua");
    paises.Add("Costa Rica");
    LlenarLista(lstPaises);
}


public static void LlenarLista(ListBox lista)
{
    lista.Items.Clear();
    foreach (string pais in paises)
    {
        lista.Items.Add(pais);
    }
}
```



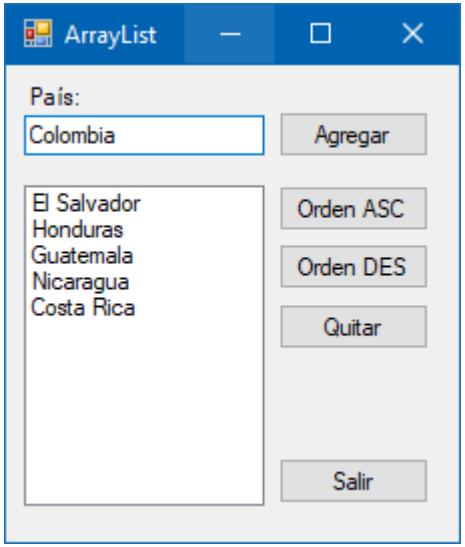
DEMO: ARRAYLIST (CONT..)

```
private void btnAgregar_Click(object sender, EventArgs e)
{
    if (txtPais.Text != String.Empty)
    {
        paises.Add(txtPais.Text);
        LlenarLista(lstPaises);
        txtPais.Clear();
        txtPais.Focus();
    }
}

private void btnOrdenAsc_Click(object sender, EventArgs e)
{
    if (lstPaises.Items.Count>1)
    {
        paises.Sort();
        LlenarLista(lstPaises);
    }
}
```



DEMO: ARRAYLIST (CONT..)

```
private void btnOrdenDes_Click(object sender, EventArgs e)
{
    if (lstPaises.Items.Count > 1)
    {
        paises.Sort();
        paises.Reverse();
        LlenarLista(lstPaises);
    }
}
```




```
private void btnQuitar_Click(object sender, EventArgs e)
{
    if (lstPaises.SelectedIndex >= 0)
    {
        lstPaises.Items.Remove(lstPaises.SelectedIndex);
        paises.RemoveAt(lstPaises.SelectedIndex);
        LlenarLista(lstPaises);
    }
}
```

COLECCIÓN LIST

- **List** es una colección genérica, disponible dentro del espacio de nombres System.Collections.Generic. Una colección genérica es útil cuando todos los elementos de la colección tienen el mismo tipo.
- Representa una lista de objetos **fuertemente tipados** a la que se puede tener acceso por índice.

- ¿Cómo se usa List en C#?

1. Agregar el espacio de nombres **Collections.Generic**;

```
using System.Collections.Generic;
```

2. Sintaxis para declarar y crear la colección **List**:

```
List<Tipo> nombreLista = new List<Tipo>();
```

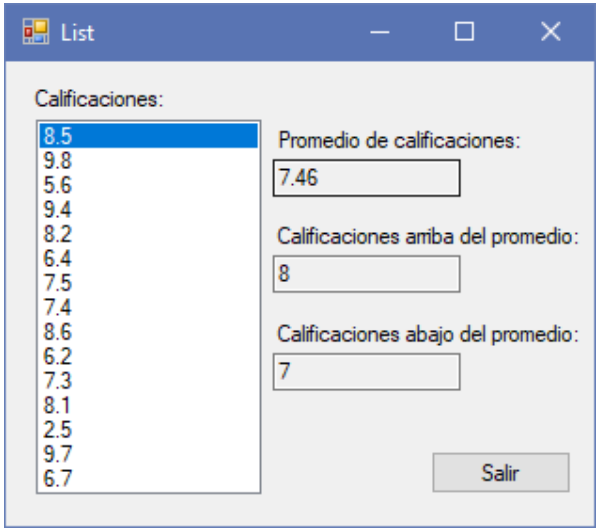
MÉTODOS Y PROPIEDADES DE LIST

Método o propiedad	Descripción
Capacity	Obtiene y establece el número de elementos para los que se reserva espacio en un momento dado, dentro del objeto List
Add	Agrega un nuevo elemento al final del List
Clear	Elimina todos los elementos del objeto List
Contains	Devuelve true si el elemento especificado está en el objeto List
Count	Retorna el número de elementos almacenados en el objeto List
IndexOf	Devuelve el índice de la primera ocurrencia del elemento especificado en el objeto List
Insert	Inserta un elemento en el índice especificado
Remove	Elimina la primera ocurrencia del elemento especificado
RemoveAt	Elimina un objeto en el índice especificado
Sort	Ordena el objeto List

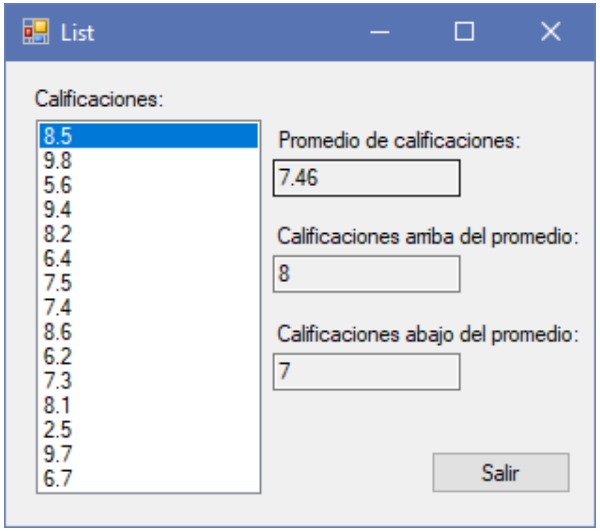
Para consultar otros métodos de **List**: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.8

DEMO: LIST

```
using System.Collections.Generic;


private double ArribaPromedio(List<double> notas, double promedio)
{
    int cant = 0;
    foreach (double nota in notas)
    {
        if (nota > promedio) cant++;
    }
    return cant;
}


private double AbajoPromedio(List<double> notas, double promedio)
{
    int cant = 0;
    foreach (double nota in notas)
    {
        if (nota < promedio) cant++;
    }
    return cant;
}
```



DEMO: LIST (CONT..)




```
private void frmList_Load(object sender, EventArgs e)
{
    List<double> calificaciones = new List<double> { 8.5, 9.8, 5.6,
      9.4, 8.2, 6.4, 7.5, 7.4, 8.6, 6.2, 7.3, 8.1, 2.5, 9.7, 6.7 };
    lstCalificaciones.DataSource = calificaciones;
    double prom = calificaciones.Average();
    txtPromedio.Text = prom.ToString("N2");
    txtArriba.Text = ArribaPromedio(calificaciones, prom).ToString();
    txtAbajo.Text = AbajoPromedio(calificaciones, prom).ToString();
}
```

COLECCIÓN DICTIONARY

- Es una estructura de datos que representa una colección de pares de datos de **clave y valor**. La clave es única en el diccionario, pero un valor puede asociarse con muchas claves diferentes.

- ¿Cómo se usa Dictionary en C#?

1. Agregar el espacio de nombres **Collections.Generic**;

```
using System.Collections.Generic;
```

2. Sintaxis para declarar y crear la colección **Dictionary**:

```
Dictionary<Clave, Valor> nombreDiccionario = new Dictionary<Clave, Valor>();
```

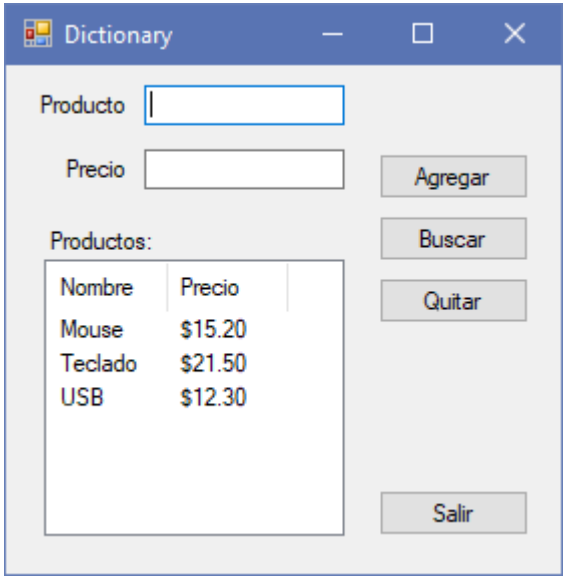
PROPIEDADES Y MÉTODOS DE DICTIONARY

Método o Propiedad	Descripción
Add	Agregar una pareja de clave – valor al diccionario
Key	Obtiene la clave en el par clave / valor
Value	Obtiene el valor en el par clave / valor
ContainsKey	Determina si el diccionario contiene la clave especificada
ContainsValue	Determina si el diccionario contiene un valor específico
Remove	Remueve el valor con la clave especificada del diccionario
Clear	Remueve todas las claves y valores del diccionario
Count	Obtiene el número de pares clave / valor contenidos en el diccionario
Equals	Determina si el objeto especificado es igual al objeto actual

Para consultar otros métodos de **Dictionary:** https://docs.microsoft.com/en-us/dotnet/api/system.collections.dictionarybase?view=netframework-4.8

DEMO: DICTIONARY

```
using System.Collections.Generic;
```




```
private void LlenarLista()
{
    lstProductos.Items.Clear();
    foreach (KeyValuePair<string, double> producto in productos)
    {
        string[] data = { producto.Key, producto.Value.ToString("C2") };
        ListViewItem item = new ListViewItem(data);
        lstProductos.Items.Add(item);
    }
}
```

DEMO: DICTIONARY (CONT..)

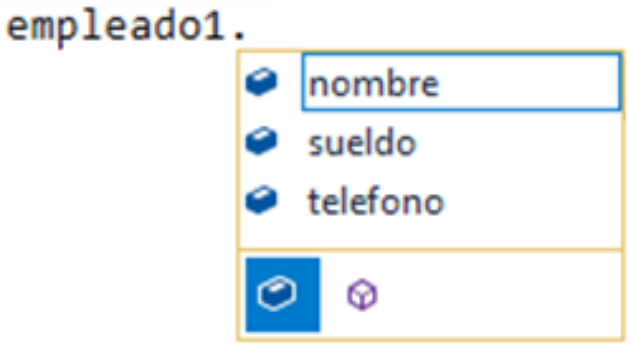
```
private void btnAgregar_Click(object sender, EventArgs e)
{
    if (txtClave.TextLength > 0 && txtValor.TextLength > 0)
    {
        if (productos.ContainsKey(txtClave.Text))
        {
            MessageBox.Show("El diccionar ya contiene ese producto.");
        }
        else
        {
            productos.Add(txtClave.Text, Convert.ToDouble(txtValor.Text));
            txtClave.Clear();
            txtValor.Clear();
            txtClave.Focus();
            LlenarLista();
        }
    }
    else
    {
        MessageBox.Show("Ingrese el nombre y precio del producto.");
        txtClave.Focus();
    }
}
```

DEMO: DICTIONARY (CONT..)

```
private void btnQuitar_Click(object sender, EventArgs e)
{
    if (txtClave.TextLength > 0)
    {
        if (!productos.ContainsKey(txtClave.Text))
        {
            MessageBox.Show("Ese producto no existe en el diccionario.");
            txtValor.Clear();
            txtClave.Focus();
            txtClave.SelectAll();
        }
        else
        {
            productos.Remove(txtClave.Text);
            LlenarLista();
        }
    }
}
```

¿QUÉ ES UNA ESTRUCTURA DE DATOS?

- Una estructura consiste en un conjunto de datos que se unen para formar un tipo de dato compuesto, conocido como **UDT** (User Defined Type), y permite agrupar bajo un único identificador, una serie de elementos relacionados: campos, constantes, propiedades y métodos.



SINTAXIS PARA CREAR UNA ESTRUCTURA

- Para crear una estructura, se debe utilizar la palabra clave **struct** junto al nombre de la estructura, situando entre llaves { } los elementos de la estructura.

- Sintaxis:

```
modificadordeAcceso struct NombreEstructura
{
   Elementos;  // campos, constantes, propiedades y métodos
}
```

- Modificadores de acceso: public, protected, private

EJEMPLO DE ESTRUCTURA DE DATOS

- Creación de la estructura:

```
public struct Empleado
{
    public string nombre;
    public string telefono;
    public decimal sueldo;
}
```

- Uso de la estructura:

```
Empleado empleado1 = new Empleado();
empleado1.nombre = txtNombre.Text;
empleado1.telefono = txtTelefono.Text;
empleado1.sueldo = Convert.ToDecimal(txtSueldo.Text);
```

USO DE ELEMENTOS PRIVADOS

- Creación de la estructura:

```
public struct Empleado
{
    public string nombre;
    public string telefono;
    private decimal sueldo;
}
```

```
Empleado empleado1 = new Empleado();
empleado1.nombre = txtNombre.Text;
empleado1.telefono = txtTelefono.Text;
empleado1.sueldo = Convert.ToDecimal(txtSueldo.Text);
```

Error
No se puede acceder directamente a un elemento privado

AGREGAR MÉTODOS GET Y SET PARA CAMPOS PRIVADOS

```
public struct Empleado
{
    public string nombre;
    public string telefono;
    private decimal sueldo;
    1 reference
    public void setSueldo(decimal sueldo)
    {
        if (sueldo >= 0)
        {
            this.sueldo = sueldo;
        }
    }
    1 reference
    public decimal getSueldo()
    {
        return sueldo;
    }
}
```

Método de modificación (**SET**) del campo sueldo

Método de acceso (**GET**) al campo sueldo

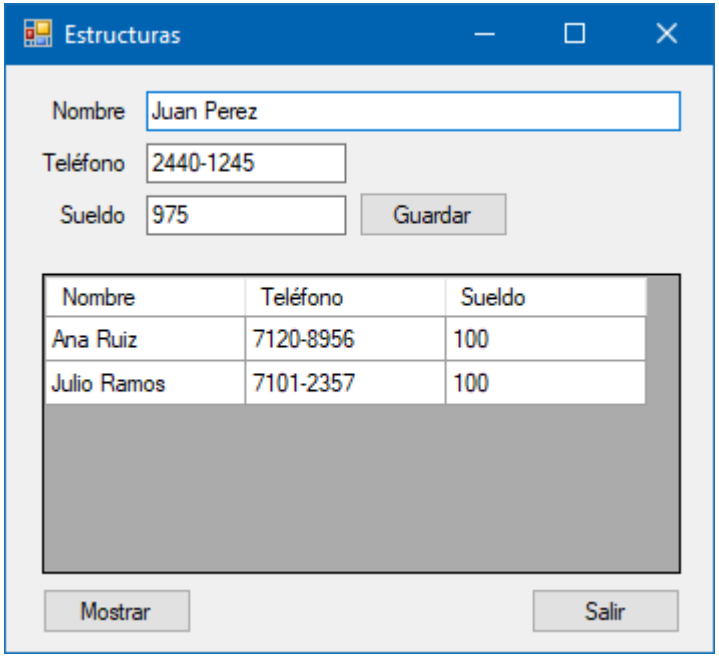
```
Empleado empleado1 = new Empleado();
empleado1.nombre = txtNombre.Text;
empleado1.telefono = txtTelefono.Text;
empleado1.setSueldo(Convert.ToDecimal(txtSueldo.Text));
```

USO DE CONSTANTES EN LA ESTRUCTURA

```
public struct Empleado
{
    public string nombre;
    public string telefono;
    private decimal sueldo;
    private const decimal bono = 100;
    1 reference
    public void setSueldo(decimal sueldo)
    {
        if (sueldo >= 0)
        {
            this.sueldo = sueldo;
        }
    }
    1 reference
    public decimal getSueldo()
    {
        return sueldo + bono;
    }
}
```

DEMO: STRUCT Y ARRAYLIST

```
public struct Empleado...
public static List<Empleado> listaEmpleados = new List<Empleado>();
private void btnGuardar_Click(object sender, EventArgs e)
{
    try
    {
        Empleado empleado = new Empleado();
        empleado.nombre = txtNombre.Text;
        empleado.telefono = txtTelefono.Text;
        empleado.setSueldo(Convert.ToDecimal(txtSueldo.Text));
        listaEmpleados.Add(empleado);
        txtNombre.Clear();
        txtTelefono.Clear();
        txtSueldo.Clear();
        txtNombre.Focus();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
private void btnMostrar_Click(object sender, EventArgs e)
{
    dgvEmpleados.Rows.Clear();
    foreach (Empleado empleado in listaEmpleados)
    {
        dgvEmpleados.Rows.Add(empleado.nombre, empleado.telefono, empleado.getSueldo());
    }
    dgvEmpleados.ClearSelection();
}
```



¿QUÉ ES UNA ENUMERACIÓN?

- Una enumeración consiste en un **conjunto de constantes relacionadas**. A cada constante se le asigna un nombre, mientras que la agrupación de tales constantes, es decir, la propia enumeración recibe también un nombre identificativo.

- Ejemplos de enumeraciones:
  - **Días de la semana**: Domingo, Lunes, Martes, Miércoles, Jueves, Viernes, Sábado
  - **Colores**: Negro, Blanco, Rojo, Verde, Azul, Amarillo, Morado, Plateado, Púrpura, Naranja
  - **Meses del año**: Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre.
  - **Coordenadas**: Este, Oeste, Sur, Norte, Noreste, Noroeste, Sureste, Suroeste.

¿CÓMO DECLARAR UNA ENUMERACIÓN EN C#?

- La construcción sintáctica (palabra reservada) para la declaración de enumeraciones en C# es **enum**

- Sintaxis:

```
modificadordeAcceso enum nombreEnumeracion
{
    Constante[= valor inicialización, otras constantes]
}
```

- Modificadores de acceso: public, protected, private
- Tipos de datos para enum: byte, sbyte, short, ushort, int, uint, long, ulong

EJEMPLO DE ENUMERACIONES

- Enumeración sin inicialización implícita y explicita (**asignar valores únicos**)

```
public enum Meses
{
    Enero,
    Febrero,
    Marzo,
    Abril,
    Mayo,
    Junio,
    Julio,
    Augosto,
    Septiembre,
    Octubre,
    Noviembre,
    Diciembre
}
```

```
public enum Meses
{
    Enero = 1,
    Febrero = 2,
    Marzo = 3,
    Abril = 4,
    Mayo = 5,
    Junio = 6,
    Julio = 7,
    Augosto = 8,
    Septiembre = 9,
    Octubre = 10,
    Noviembre = 11,
    Diciembre = 12
}
```

COMO RECORRER UNA ENUMERACIÓN

- Usando foreach

```
public enum Meses
{
    Enero,
    Febrero,
    Marzo,
    Abril,
    Mayo,
    Junio,
    Julio,
    Augosto,
    Septiembre,
    Octubre,
    Noviembre,
    Diciembre
}
```

```
foreach (var mes in Enum.GetValues(typeof(Meses)))
{
    listBox1.Items.Add(mes);
}
```

COMO RECUPERAR VALORES DE UNA ENUMERACIÓN

- Se puede recorrer la enumeración o recuperar un valor especifico

```
public enum Meses
{
    Enero = 1,
    Febrero = 2,
    Marzo = 3,
    Abril = 4,
    Mayo = 5,
    Junio = 6,
    Julio = 7,
    Augosto = 8,
    Septiembre = 9,
    Octubre = 10,
    Noviembre = 11,
    Diciembre = 12
}
```

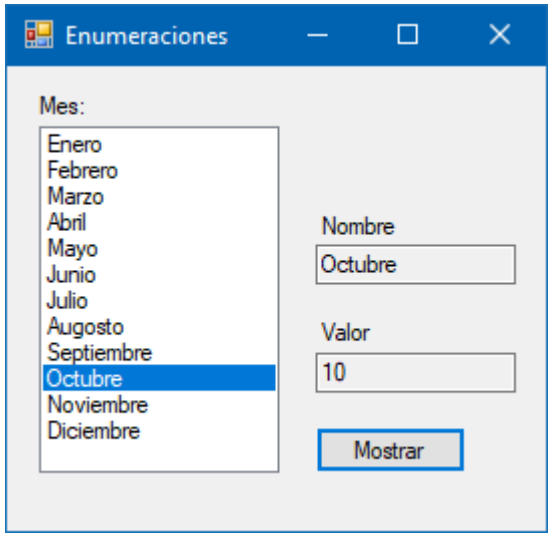
```
foreach (var mes in Enum.GetValues(typeof(Meses)))
{
    MessageBox.Show(mes.ToString());
    MessageBox.Show(((int)mes).ToString());
}
```

```
Meses mes = Meses.Octubre;
int valor = (int)Meses.Octubre;
MessageBox.Show(mes.ToString());
MessageBox.Show(valor.ToString());
```

DEMO: ENUMERACIÓN

```
public enum Meses
{
    Enero = 1,
    Febrero = 2,
    Marzo = 3,
    Abril = 4,
    Mayo = 5,
    Junio = 6,
    Julio = 7,
    Augosto = 8,
    Septiembre = 9,
    Octubre = 10,
    Noviembre = 11,
    Diciembre = 12
}
```

```
private void frmEnum_Load(object sender, EventArgs e)
{
    foreach (var mes in Enum.GetValues(typeof(Meses)))
    {
        lstMeses.Items.Add(mes);
    }
}
```




```
private void btnMostrar_Click(object sender, EventArgs e)
{
    if (lstMeses.SelectedIndex>=0)
    {
        Meses mes = (Meses)Enum.ToObject(typeof(Meses), lstMeses.SelectedIndex + 1);
        int valor = (int)mes;
        txtNombre.Text = mes.ToString();
        txtValor.Text = valor.ToString();
    }
}
```

BIBLIOGRAFÍA

1. Asad, A. (2017). The C# Programmer's Study Guide (MCSD) Exam: 70-483. Pakistan: Apress.
2. Ceballos, F. (2013). Microsoft C# Curso de programación. Segunda edición. México: Alfaomega.
3. Putier, S. (2015). C# 6 y Visual Studio 2015 Los fundamentos del lenguaje. España: ENI.