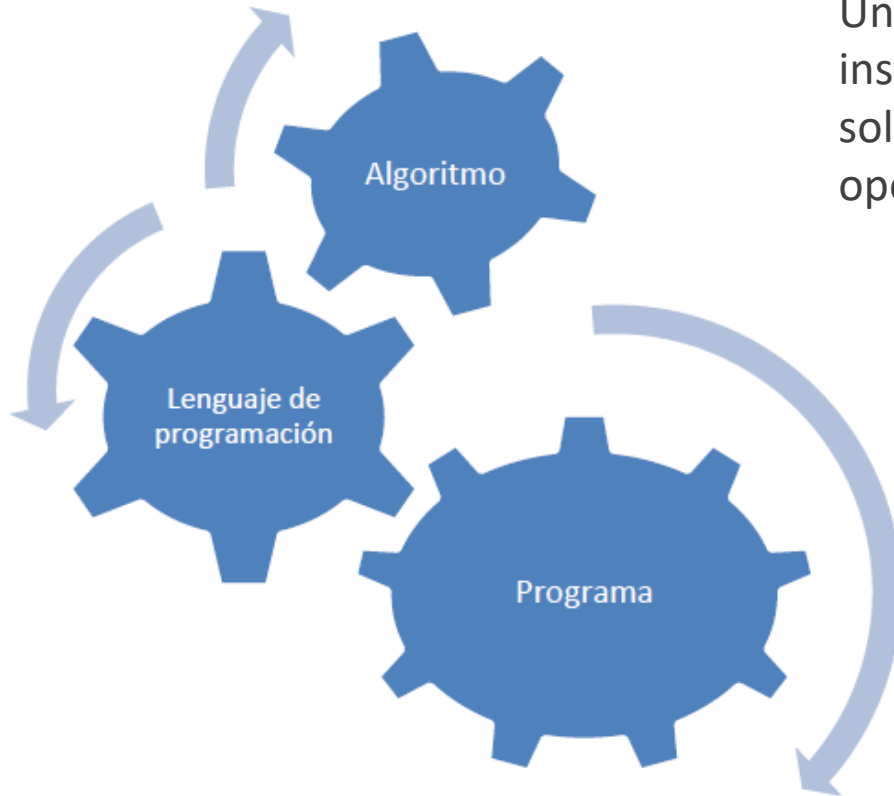


1.9 ANEXO C#.

1.9.1 Previo a programar. Recordamos...



Un **algoritmo** es una secuencia finita de instrucciones o reglas definidas, que permite solucionar un problema concreto o realizar una operación de computación.

Un **diagrama de flujo** es una representación gráfica de una serie de operaciones que se ejecutan en orden escalonado para dar como resultado final la solución a un problema o tarea específica.

Un **pseudocódigo** es un paso intermedio entre los diagramas de flujo, que se expresan mediante símbolos, y los lenguajes de programación, que están ligados a una sintaxis bien definida.

1.9 ANEXO C#.

1.9.2. Datos primitivos

C Sharp (C#) se un lenguaje de programación de alto nivel de propósito general y multi-paradigma desarrollado por Microsoft que sirve como lenguaje base del framework .NET.



C# includes some predefined value types and reference types. The following table lists predefined data types:

Type	Description	Range	Suffix
byte	8-bit unsigned integer		0 to 255
sbyte	8-bit signed integer		-128 to 127
short	16-bit signed integer		-32,768 to 32,767
ushort	16-bit unsigned integer		0 to 65,535
int	32-bit signed integer		-2,147,483,648 to 2,147,483,647
uint	32-bit unsigned integer		0 to 4,294,967,295 u
long	64-bit signed integer		-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 l
ulong	64-bit unsigned integer		0 to 18,446,744,073,709,551,615 ul
float	32-bit Single-precision floating point type		-3.402823e38 to 3.402823e38 f
double	64-bit double-precision floating point type		-1.79769313486232e308 to 1.79769313486232e308 d
decimal	128-bit decimal type for financial and monetary calculations		(+ or -)1.0 x 10e-28 to 7.9 x 10e28 m
char	16-bit single Unicode character	Any valid character, e.g. a,*, \x0058 (hex), or\u0058 (Unicode)	
bool	8-bit logical true/false value	True or False	
object	Base type of all other types.		
string	A sequence of Unicode characters		
DateTime	Represents date and time	0:00:00am 1/1/01 to 11:59:59pm 12/31/9999	

1.9 ANEXO C#.

Tipo de Dato Primitivo	Descripción	Tamaño	Rango	Ejemplo de Uso
int	Números enteros (integers)	32	-2^{31} hasta $2^{31} - 1$	int count; count = 42;
long	Números enteros (bigger range)	64	-2^{63} hasta $2^{63} - 1$	long wait; wait = 42L;
float	Números de punto flotante	32	-3.4×10^{-38} hasta 3.4×10^{38}	float away; away = 0.42F;
double	Números de punto flotante de Doble – Precisión (más precisos)	64	$\pm 5.0 \times 10^{-324}$ hasta $\pm 1.7 \times 10^{308}$	double trouble; trouble = 0.42;
decimal	Valores Monetarios	128	28 dígitos significativos	decimal coin; coin = 0.42M;
string	Secuencia de Caracteres	16 bits por carácter	No aplicable	string vest vest = "forty two";
char	Carácter único	16	Un único carácter	char grill; grill = 'x';
bool	Booleanos	8	Verdadero ó Falso	bool teeth; teeth = false;

1.9 ANEXO C#.

1.9.3. Instrucciones de declaración

Una instrucción de declaración declara una nueva variable local, constante local o variable local de referencia.

```
string greeting;  
int a, b, c;  
List<double> xs;
```

En una instrucción de declaración, también puede inicializar una variable con su valor inicial:

```
string greeting = "Hello";  
int a = 3, b = 2, c = a + b;  
List<double> xs = new();
```

1.9 ANEXO C#.

Los ejemplos anteriores especifican explícitamente el tipo de una variable. También puede permitir que el compilador infiera el tipo de una variable a partir de su expresión de inicialización. Para ello, use la palabra clave *var*.

Son **variables locales con asignación implícita de tipos**.

```
var greeting = "Hello";  
Console.WriteLine(greeting.GetType()); // output: System.String  
  
var a = 32;  
Console.WriteLine(a.GetType()); // output: System.Int32  
  
var xs = new List<double>();  
Console.WriteLine(xs.GetType()); // output: System.Collections.Generic.List`1[System.Double]
```

Para declarar una constante, la palabra clave *const*.

```
const string Greeting = "Hello";  
const double MinLimit = -10.0, MaxLimit = -MinLimit;
```

1.9 ANEXO C#.

Una **variable de referencia** es una variable que hace referencia a otra variable, que se denomina *referencia*. Es decir, una variable de referencia es un *alias* para su referente. Al asignar un valor a una variable de referencia, ese valor se asigna al referente. Cuando se lee el valor de una variable de referencia, se devuelve el valor del referente.

```
int a = 1;
ref int alias = ref a;
Console.WriteLine($"{a, alias} is ({a}, {alias})"); // output: (a, alias) is (1, 1)

a = 2;
Console.WriteLine($"{a, alias} is ({a}, {alias})"); // output: (a, alias) is (2, 2)

alias = 3;
Console.WriteLine($"{a, alias} is ({a}, {alias})"); // output: (a, alias) is (3, 3)
```

1.9 ANEXO C#.

1.9.3.1 Tipos anónimos

Los tipos anónimos son una manera cómoda de encapsular un conjunto de propiedades de solo lectura en un único objeto sin tener que definir primero un tipo explícitamente.

El compilador deduce el tipo de cada propiedad.

Para crear tipos anónimos, usa el operador *new* con un inicializador de objeto.

```
var v = new { Amount = 108, Message = "Hello" };

// Rest the mouse pointer over v.Amount and v.Message in the following
// statement to verify that their inferred types are int and string.
Console.WriteLine(v.Amount + v.Message);
```

Los tipos anónimos suelen usarse en la cláusula select de una expresión de consulta para devolver un subconjunto de las propiedades de cada objeto en la secuencia de origen.

1.9 ANEXO C#.

El escenario más habitual es inicializar un tipo anónimo con propiedades de otro tipo. En el siguiente ejemplo, se da por hecho que existe una clase con el nombre `Product`. La clase incluye propiedades `Color` y `Price` (y otras que no interesan).

La variable `products` es una colección de objetos `Product`. La declaración del tipo anónimo empieza con la palabra *new*. La declaración inicializa un nuevo tipo que solo usa 2 propiedades de `Product`. El uso de tipo anónimos hace que la consulta devuelva una cantidad menor de datos.

```
var productQuery =  
    from prod in products  
    select new { prod.Color, prod.Price };  
  
foreach (var v in productQuery)  
{  
    Console.WriteLine("Color={0}, Price={1}", v.Color, v.Price);  
}
```


1.9 ANEXO C#.

1.9.3.2 Tipos de valor

Un *struct* se usa como un contenedor para un pequeño conjunto de variables relacionadas

```
public struct Coords
{
    public int x, y;

    public Coords(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

Una enumeración, *enum*, define un conjunto de constantes integrales con nombre.

```
public enum FileMode
{
    CreateNew = 1,
    Create = 2,
    Open = 3,
    OpenOrCreate = 4,
    Truncate = 5,
    Append = 6,
}
```

1.9 ANEXO C#.

1.9.3.3 Tipos que admiten valor null

Los tipos de valor normales no pueden tener un valor *null*, pero se pueden crear *tipos de valor que aceptan valores NULL* mediante la adición de “?” después del tipo.

int? numero;

Son especialmente útiles cuando hay un intercambio de datos con bases de datos en las que los valores numéricos podrían ser *null*.

1.9.3.4 Tipos genéricos

El código de cliente proporciona el tipo concreto cuando crea una instancia del tipo. Estos tipos se denominan *tipos genéricos*. Las clases de colección genéricas se denominan *colecciones con establecimiento inflexible de tipos* porque el compilador conoce el tipo específico de los elementos de la colección y puede generar un error en tiempo de compilación.

```
List<string> stringList = new List<string>();  
stringList.Add("String example");  
// compile time error adding a type other than a string:  
stringList.Add(4);
```

1.9 ANEXO C#.

1.9.4. Clases

Un tipo definido como *class* es un tipo de referencia. Al declarar una variable de un tipo de referencia en tiempo de ejecución, esta contendrá el valor *null* hasta que se cree expresamente una instancia de la clase mediante el operador *new* o se le asigne un valor compatible.

```
//Declaring an object of type MyClass.  
MyClass mc = new MyClass();  
  
//Declaring another object of the same type, assigning it the value of the first object.  
MyClass mc2 = mc;
```

Declarar clases

```
//[access modifier] - [class] - [identifier]  
public class Customer  
{  
    // Fields, properties, methods and events go here...  
}
```

1.9 ANEXO C#.

Crear objeto a partir de clases

```
Customer object1 = new Customer();
```

Constructores e inicialización

```
public class Container
{
    // Initialize capacity field to a default value of 10:
    private int _capacity = 10;
}
```

```
public class Container
{
    private int _capacity;

    public Container(int capacity) => _capacity = capacity;
}
```

Required obliga a los llamadores a establecer esas propiedades como parte de una expresión *new*

```
public class Person
{
    public required string LastName { get; set; }
    public required string FirstName { get; set; }
}
```

```
var p1 = new Person(); // Error! Required properties not set
var p2 = new Person() { FirstName = "Grace", LastName = "Hopper" };
```

1.9 ANEXO C#.

Herencia de clases

Las clases admiten completamente la *herencia*, una característica fundamental de la programación orientada a objetos. Al crear una clase, puede heredar de cualquier otra clase que no esté definida como *sealed*.

Otras clases pueden heredar de la clase e invalidar los métodos virtuales de clase. Además, puede implementar una o varias interfaces (no así heredar de varias clases a la vez, la herencia múltiple no está soportada; puede ocurrir de manera indirecta por herencia sucesivas). **La herencia se consigue mediante una *derivación*, en la que se declara una clase mediante una *clase base*, desde la que hereda los datos y el comportamiento.**

```
public class Manager : Employee
{
    // Employee fields, properties, methods and events are inherited
    // New Manager fields, properties, methods and events go here...
}
```

Una clase *abstracta* contiene métodos abstractos que tienen una definición de firma, pero no tienen ninguna implementación. No se pueden crear instancias de las clases abstractas. Solo se pueden usar a través de las clases derivadas que implementan los métodos abstractos.

1.9 ANEXO C#.

Herencia de clases

Ejemplo

```
abstract class Motorcycle
{
    // Anyone can call this.
    public void StartEngine() { /* Method statements here */ }

    // Only derived classes can call this.
    protected void AddGas(int gallons) { /* Method statements here */ }

    // Derived classes can override the base class implementation.
    public virtual int Drive(int miles, int speed) { /* Method statements here */ return 1; }

    // Derived classes must implement this.
    public abstract double GetTopSpeed();
}
```

1.9 ANEXO C#.

Herencia de clases

Ejemplo

```
class TestMotorcycle : Motorcycle
{
    public override double GetTopSpeed()
    {
        return 108.4;
    }

    static void Main()
    {
        TestMotorcycle moto = new TestMotorcycle();

        moto.StartEngine();
        moto.AddGas(15);
        moto.Drive(5, 20);
        double speed = moto.GetTopSpeed();
        Console.WriteLine("My top speed is {0}", speed);
    }
}
```

1.9 ANEXO C#.

Getters/ Setters

Son funciones de acceso a las campos de la clase.

```
namespace GetSet
{
    class Persona
    {
        private string _nombre;
        private int _edad;
        public string nombre
        {
            get
            {
                return _nombre;
            }
            set
            {
                if (!string.IsNullOrEmpty(value))
                {
                    _nombre = value;
                }
            }
        }
        public int edad
        {
            get
            {
                return _edad;
            }
            set
            {
                if (value>0)
                {
                    _edad = value;
                }
            }
        }
    }
}
```

Pueden utilizarse para proteger datos de las clases instanciadas. Como en el ejemplo de la clase Persona.

La definición de los get y set en este ejemplo está extendida.

1.9 ANEXO C#.

Ejemplo

Explorar estos ejemplos puede ser de utilidad:

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties>

```
class Employee
{
    private string _name; // the name field
    public string Name => _name; // the Name property
}
```

```
var employee= new Employee();
//...
```

```
System.Console.Write(employee.Name); // the get accessor is invoked here
```

```
class Student
{
    private string _name; // the name field
    public string Name // the Name property
    {
        get => _name;
        set => _name = value;
    }
}
```

```
var student = new Student();
student.Name = "Joe"; // the set accessor is invoked here
```

```
System.Console.Write(student.Name); // the get accessor is invoked here
```

1.9 ANEXO C#.


1.9.5. Modificadores de acceso

Ubicación del autor de la llamada	public	protected internal	protected	internal	private protected	private
Dentro de la clase	✓	✓	✓	✓	✓	✓
Clase derivada (mismo ensamblado)	✓	✓	✓	✓	✓	✗
Clase no derivada (mismo ensamblado)	✓	✓	✗	✓	✗	✗
Clase derivada (otro ensamblado)	✓	✓	✓	✗	✗	✗
Clase no derivada (otro ensamblado)	✓	✗	✗	✗	✗	✗

1.9 ANEXO C#.

1.9.5. Modificadores de acceso

C#

 Copiar

```
public readonly struct Coords
{
    public Coords(double x, double y)
    {
        X = x;
        Y = y;
    }

    public double X { get; init; }
    public double Y { get; init; }

    public override string ToString() => $"({X}, {Y})";
}

public static void Main()
{
    var p1 = new Coords(0, 0);
    Console.WriteLine(p1); // output: (0, 0)

    var p2 = p1 with { X = 3 };
    Console.WriteLine(p2); // output: (3, 0)

    var p3 = p1 with { X = 1, Y = 4 };
    Console.WriteLine(p3); // output: (1, 4)
}
```

1.9 ANEXO C#.

1.9.6. Declaración de espacios de nombres para organizar

Los espacios de nombres se usan mucho en programación de C# de dos maneras. En primer lugar, .NET usa espacios de nombres para organizar sus clases de la siguiente manera:

```
System.Console.WriteLine("Hello World!");
```

System es un namespace y Console es una clase de ese namespace. La palabra clave *using* se puede emplear para que el nombre completo no sea necesario.

```
using System;
```

```
C#
```

```
Console.WriteLine("Hello World!");
```

1.9 ANEXO C#.

1.9.6. Declaración de espacios de nombres para organizar

En segundo lugar, declarar tus propios espacios de nombres puede ayudarte a controlar el ámbito de nombres de clase y métodos en proyectos de programación grandes. Se utiliza la palabra clave *namespace*.

Los espacios de nombres tienen las propiedades siguientes:

- Organizan proyectos de código de gran tamaño.
- Se delimitan mediante el operador `.`.
- La directiva `using` obvia la necesidad de especificar el nombre del espacio de nombres para cada clase.
- El espacio de nombres `global` es el espacio de nombres "raíz": `global::System` siempre hará referencia al espacio de nombres `System` de .NET.

1.9 ANEXO C#.

1.9.7. Colecciones

Clase	Descripción
Dictionary<TKey,TValue>	Representa una colección de pares de clave y valor que se organizan según la clave.
List<T>	Representa una lista de objetos a los que puede tener acceso el índice. Proporciona métodos para buscar, ordenar y modificar listas.
Queue<T>	Representa una colección de objetos de primeras entradas, primeras salidas (FIFO).
SortedList<TKey,TValue>	Representa una colección de pares clave-valor que se ordenan por claves según la implementación de IComparer<T> asociada.
Stack<T>	Representa una colección de objetos de últimas entradas, primeras salidas (LIFO).

1.9 ANEXO C#.

1.9.7. Colecciones

```
// Create a list of strings by using a  
// collection initializer.  
var salmons = new List<string> { "chinook", "coho", "pink", "sockeye" };
```

```
// Create a list of strings.  
var salmons = new List<string>();  
salmons.Add("chinook");  
salmons.Add("coho");  
salmons.Add("pink");  
salmons.Add("sockeye");  
  
// Iterate through the list.  
foreach (var salmon in salmons)  
{  
    Console.WriteLine(salmon + " ");  
}  
// Output: chinook coho pink sockeye
```

```
// Remove an element from the list by specifying  
// the object.  
salmons.Remove("coho");
```

1.9 ANEXO C#.

1.9.7. Colecciones

```
var numbers = new List<int> { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// Remove odd numbers.
for (var index = numbers.Count - 1; index >= 0; index--)
{
    if (numbers[index] % 2 == 1)
    {
        // Remove the element by specifying
        // the zero-based index in the list.
        numbers.RemoveAt(index);
    }
}

// Iterate through the list.
// A lambda expression is placed in the ForEach method
// of the List(T) object.
numbers.ForEach(
    number => Console.Write(number + " "));
// Output: 0 2 4 6 8
```


1.9 ANEXO C#.

1.9.7. Colecciones

```
private static void IterateThroughList()
{
    var theGalaxies = new List<Galaxy>
    {
        new Galaxy() { Name="Tadpole", MegaLightYears=400},
        new Galaxy() { Name="Pinwheel", MegaLightYears=25},
        new Galaxy() { Name="Milky Way", MegaLightYears=0},
        new Galaxy() { Name="Andromeda", MegaLightYears=3}
    };

    foreach (Galaxy theGalaxy in theGalaxies)
    {
        Console.WriteLine(theGalaxy.Name + " " + theGalaxy.MegaLightYears);
    }

    // Output:
    // Tadpole 400
    // Pinwheel 25
    // Milky Way 0
    // Andromeda 3
}

public class Galaxy
{
    public string Name { get; set; }
    public int MegaLightYears { get; set; }
}
```

1.9 ANEXO C#.

1.9.8. Interfaces

Una interfaz define un contrato. Cualquier class o struct que implemente ese contrato **debe proporcionar una implementación de los miembros definidos en la interfaz.**

```
interface ISampleInterface
{
    void SampleMethod();
}

class ImplementationClass : ISampleInterface
{
    // Explicit interface member implementation:
    void ISampleInterface.SampleMethod()
    {
        // Method implementation.
    }

    static void Main()
    {
        // Declare an interface instance.
        ISampleInterface obj = new ImplementationClass();

        // Call the member.
        obj.SampleMethod();
    }
}
```

1.9 ANEXO C#.

1.9.9. Control de excepciones

```
public class ExceptionTest
{
    static double SafeDivision(double x, double y)
    {
        if (y == 0)
            throw new DivideByZeroException();
        return x / y;
    }

    public static void Main()
    {
        // Input for test purposes. Change the values to see
        // exception handling behavior.
        double a = 98, b = 0;
        double result;

        try
        {
            result = SafeDivision(a, b);
            Console.WriteLine("{0} divided by {1} = {2}", a, b, result);
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Attempted divide by zero.");
        }
    }
}
```

Throw new ex

Try{}catch(ex)

1.9 ANEXO C#.

1.9.10. Arrays

<https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/arrays>

```
// Declare a single-dimensional array of 5 integers.
int[] array1 = new int[5];

// Declare and set array element values.
int[] array2 = { 1, 2, 3, 4, 5, 6 };

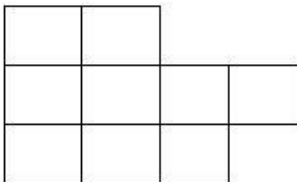
// Declare a two dimensional array.
int[,] multiDimensionalArray1 = new int[2, 3];

// Declare and set array element values.
int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

// Declare a jagged array.
int[][] jaggedArray = new int[6][];

// Set the values of the first array in the jagged array structure.
jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
```

Se dice que una matriz es irregular (jagged) si la cantidad de elementos de cada fila varía. Luego podemos imaginar una matriz irregular:



```
mat[0]=new int[2];
mat[1]=new int[4];
mat[2]=new int[3];
```

1.9 ANEXO C#.

1.9.11. Strings

<https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/strings/>

```
// Declare without initializing.
string message1;

// Initialize to null.
string message2 = null;

// Initialize as an empty string.
// Use the Empty constant instead of the literal "".
string message3 = System.String.Empty;

// Initialize with a regular string literal.
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";

// Initialize with a verbatim string literal.
string newPath = @"c:\Program Files\Microsoft Visual Studio 9.0";

// Use System.String if you prefer.
System.String greeting = "Hello World!";

// In local variables (i.e. within a method body)
// you can use implicit typing.
var temp = "I'm still a strongly-typed System.String!";

// Use a const string to prevent 'message4' from
// being used to store another string value.
const string message4 = "You can't get rid of me!";

// Use the String constructor only when creating
// a string from a char*, char[], or sbyte*. See
// System.String documentation for details.
char[] letters = { 'A', 'B', 'C' };
string alphabet = new string(letters);
```

1.9 ANEXO C#.

1.9.10. Instrucciones de control de flujo

1.9.10.1 Sentencia de bloque

1.9.10.2 Sentencias condicionales: If/ switch

1.9.10.3 Sentencias de iteración: while/ do while/ for/ for each

1.9.10.4 Clausura de iteraciones break/ continue/ throw/ return

1.9.10. Funciones

```
AccessMod output_type Nombre_Función (Argumentos){  
    return output type;  
}  
  
public void_type Nombre_Función (argumentos){  
  
}
```

1.10 Eventos, escuchadores y acciones a eventos

1.10.1. Evento

Un evento es una acción que resulta de la **interacción de un usuario** sobre un componente de la aplicación.

- Es necesario definir cada evento con la acción a la que va asociada. Esta acción es conocida como administrador de eventos.
- La idea es: cuando se ejecuta el programa, se realizan inicializaciones y después **el programa esperará a que ocurra cualquier evento**.
- Cuando se dispara un evento el programa ejecuta el código correspondiente del administrador de eventos (es decir, esa acción que hemos programado que haga). Ejemplo: un botón que cuando el usuario lo pulsa se reproduce un sonido.
- La programación dirigida a eventos es la base de la interfaz de usuario (ya que el usuario interactúa con ésta constantemente).

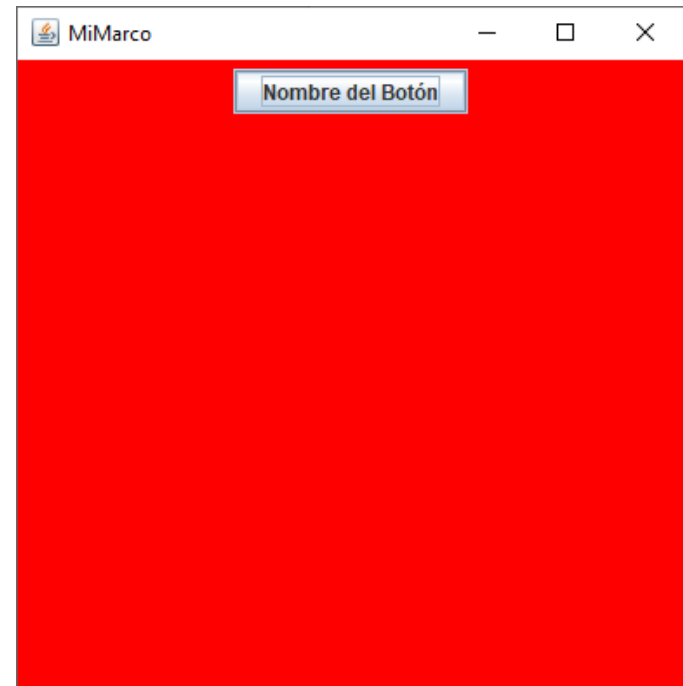
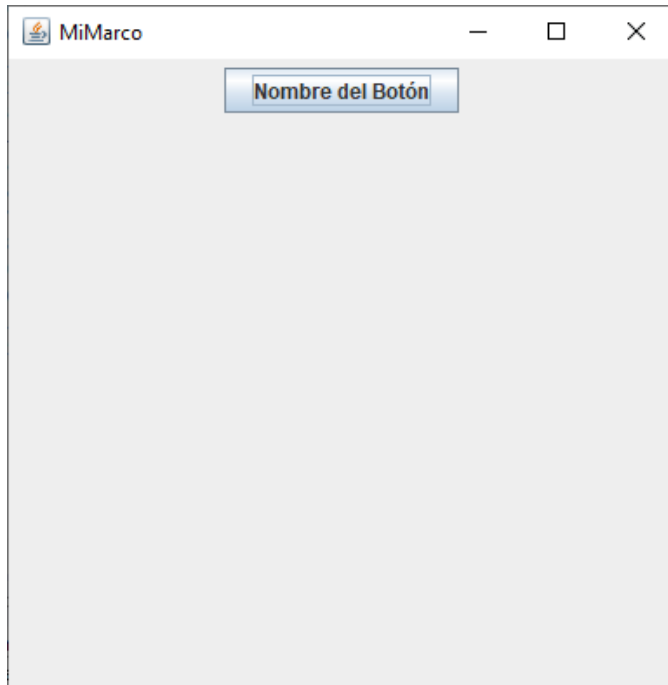
1.10 Eventos, escuchadores y acciones a eventos

1.10.2. Escuchador

- Un escuchador de eventos (**event listener**) es un mecanismo asíncrono pendiente de ciertas circunstancias que ocurren en clases diferentes a la nuestra. Se utiliza, por ejemplo, para detectar si un botón ha sido pulsado.
- Para usar un escuchador de eventos, se tienen que seguir tres pasos:
 - ❖ Implementar la interfaz del escuchador.
 - ❖ Registrar el escuchador en el objeto que genera el evento, indicándole el objeto que los recogerá.
 - ❖ Implementar los métodos **callback** correspondientes.

1.10 Eventos, escuchadores y acciones a eventos

- Ex1 – Realizar una interfaz con un botón que cambie el color del fondo de la aplicación.



1.10 Eventos, escuchadores y acciones a eventos

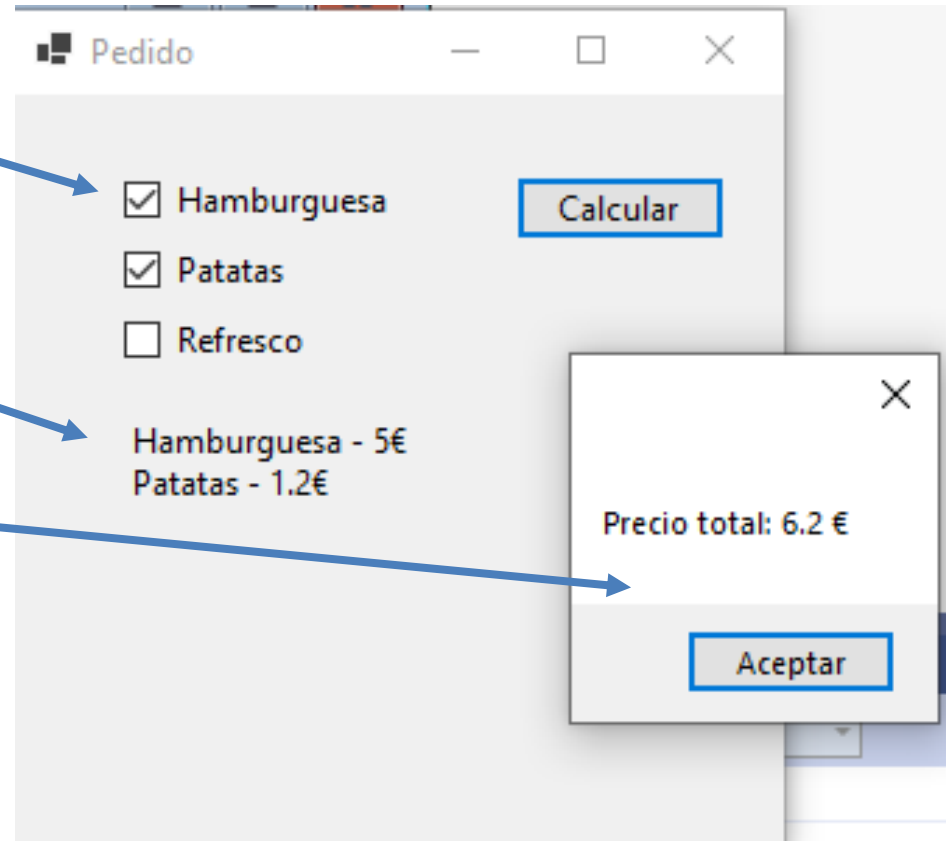
- Ex1 – Realizar una interfaz con un botón que cambie el color del fondo de la aplicación.
- Guardar trabajos en carpeta AD22-UD1-Apellido_Nombre
- Análisis de elementos de la aplicación.
- Realizar interfaz con botón en Eclipse estructurando el programa con 3 clases.
- Añadir 2 botones más para cambiar a amarillo, azul y rojo.
- Añadir botón para borrar color.
- Añadir cuadro de texto para seleccionar color + botón de confirmación.
- Mover botones a la parte inferior. Determinar coordenadas.

1.11 Componentes

Check Box

Labels

Message Box



Comportamiento Modal: Impide la ejecución del programa si no se da respuesta. Se emplea para mostrar información crítica.

Comportamiento no modal: El usuario puede ignorar la información y continuar con el programa.

1.11 Componentes

Group Box

The image shows a Windows form titled "Form1" with a light gray background. It contains the following elements:

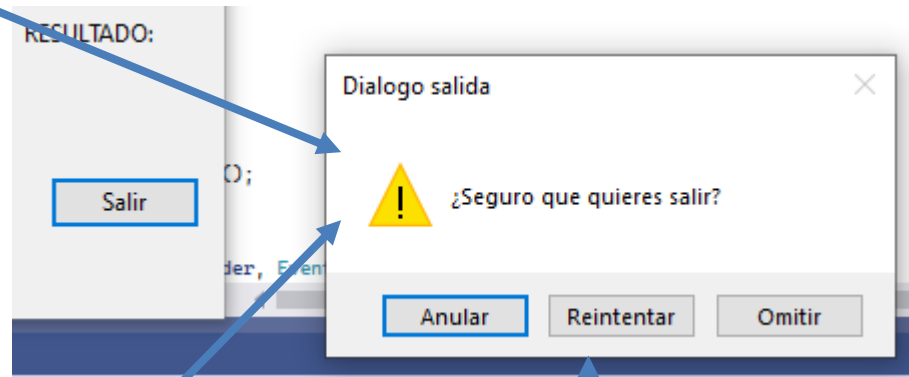
- Two text input fields: the first is labeled "A" and contains the value "10"; the second is labeled "B" and contains the value "5".
- A button labeled "Calcular" with a blue border.
- A "Group Box" titled "Operaciones" containing two radio buttons: "Suma" (which is selected) and "Resta".
- A label "RESULTADO:" followed by the value "15".

Blue arrows point from the text labels "Group Box" and "Radio buttons" to the "Operaciones" group box and the radio buttons, respectively.

Radio buttons

1.11 Componentes

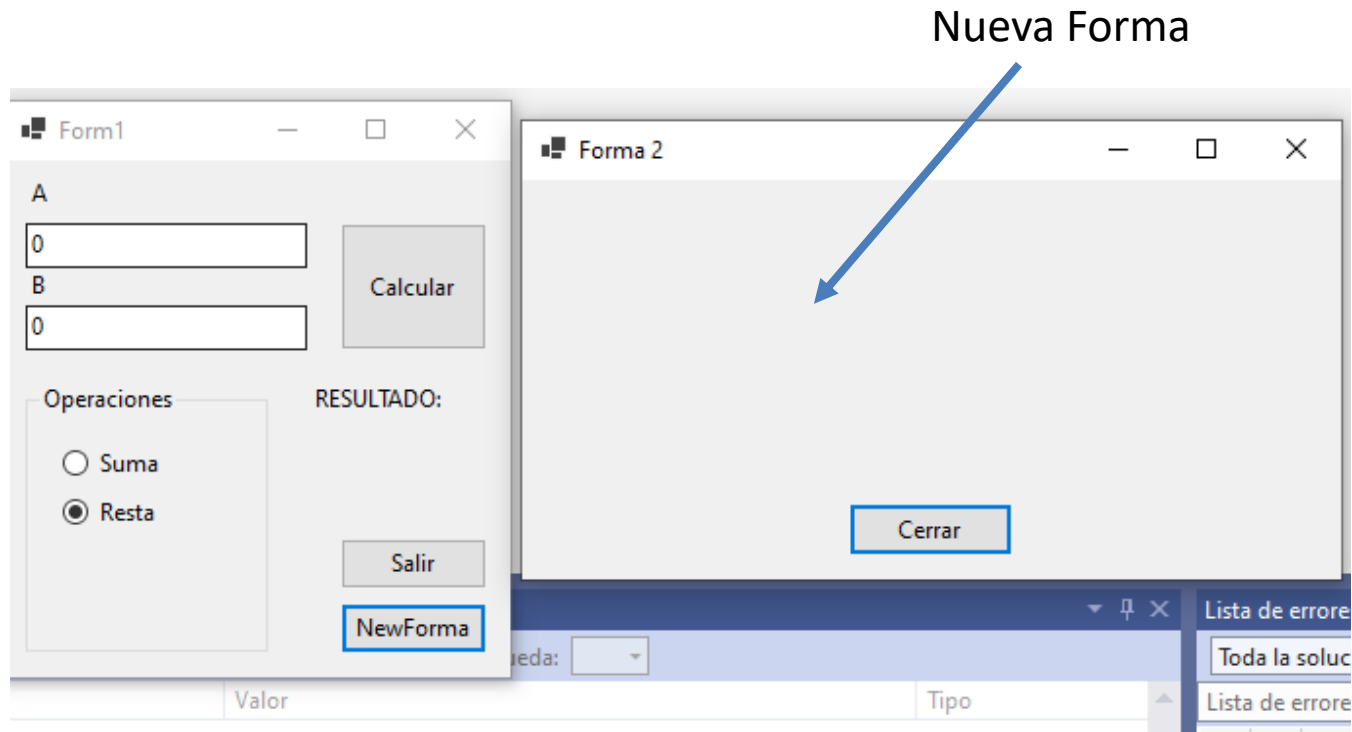
Message Box



Mssge icon

MessageButtonsBox

1.11 Componentes

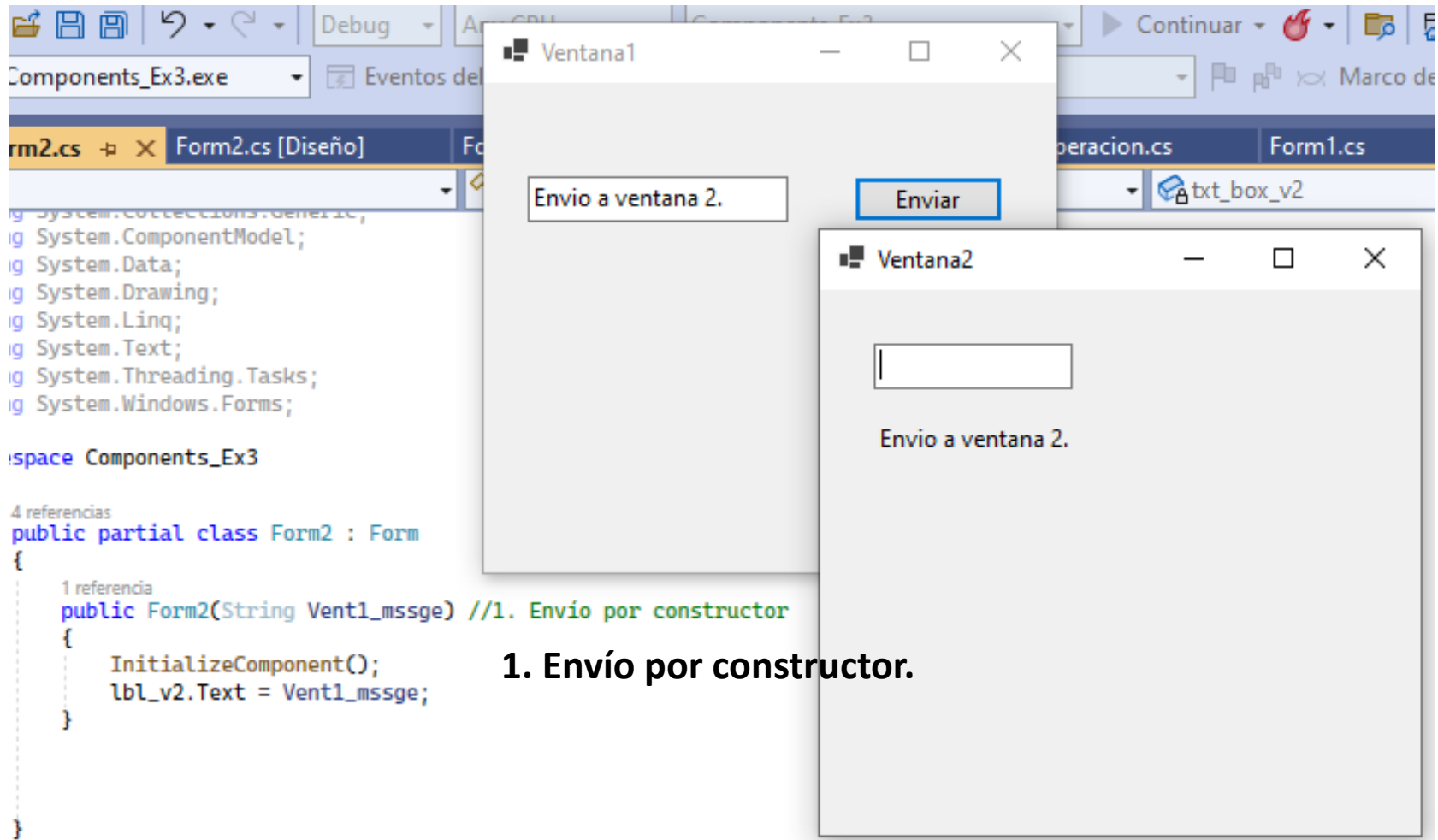


Show

ShowDialog -> Forma modal

1.11 Componentes

Mandar información entre Windows forms.

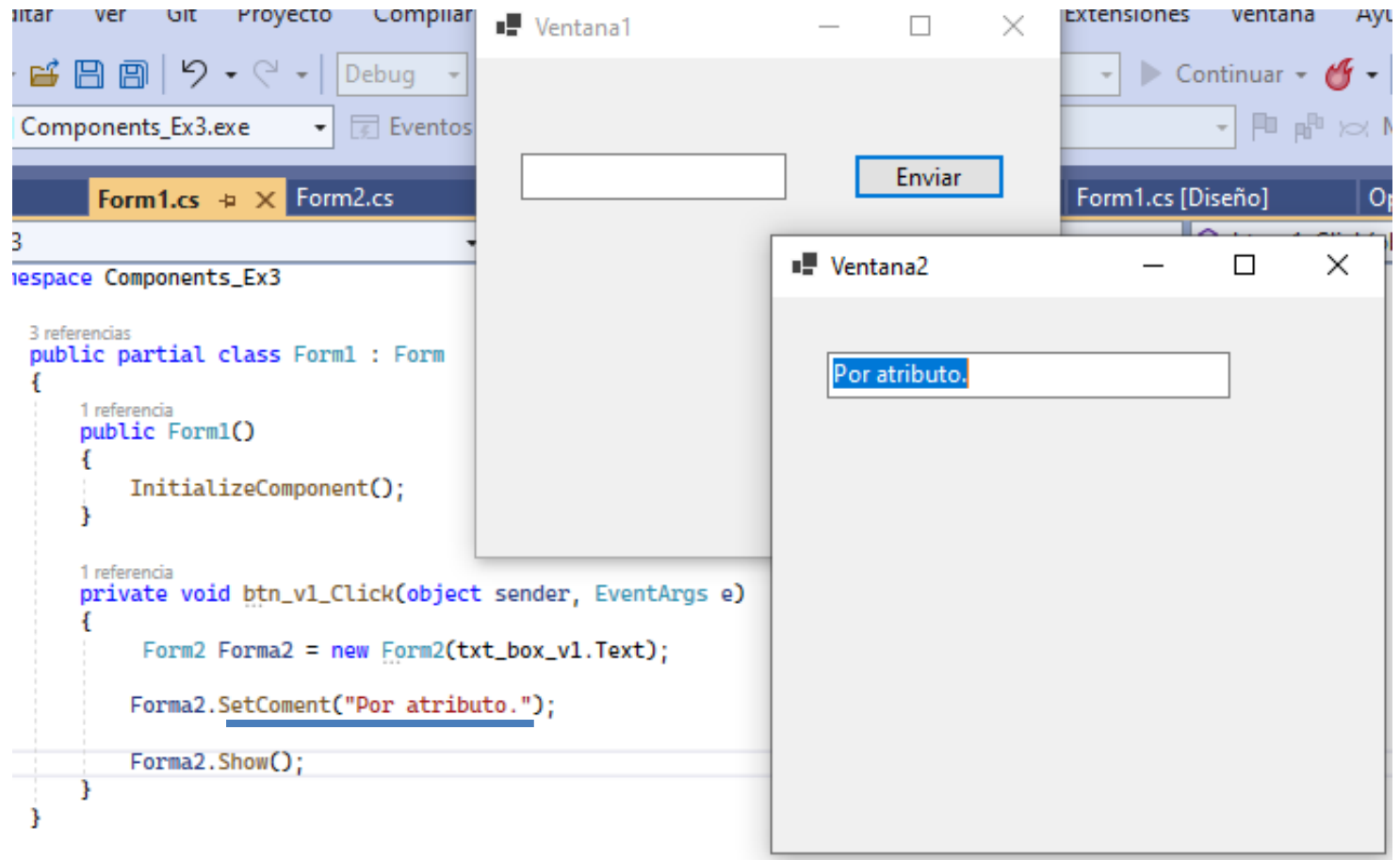


1. Envío por constructor.

1.11 Componentes

Mandar información entre Windows forms.

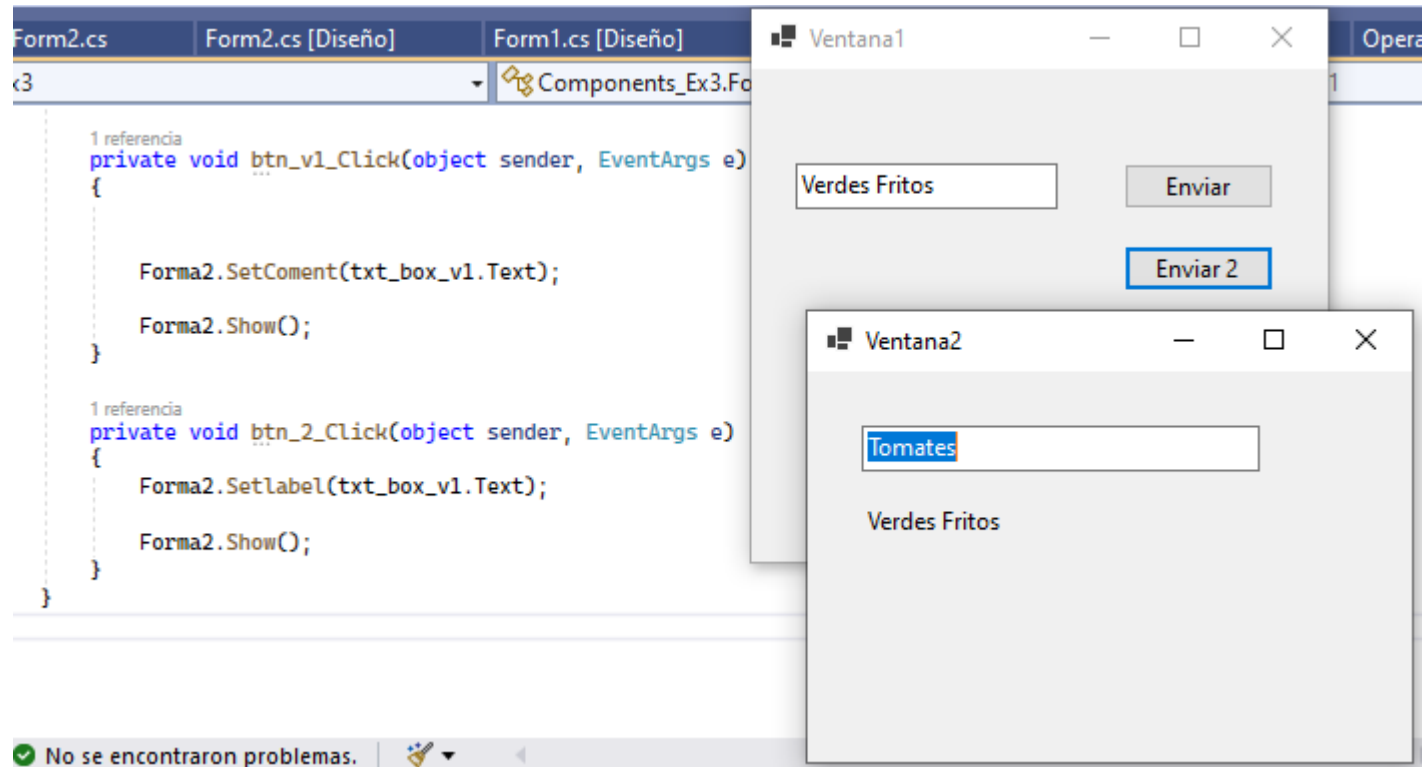
2. Envío setter (atributo)



1.11 Componentes

Mandar información entre Windows forms.

3. Instancia fuera la clase para que acceda cualquier event handler.



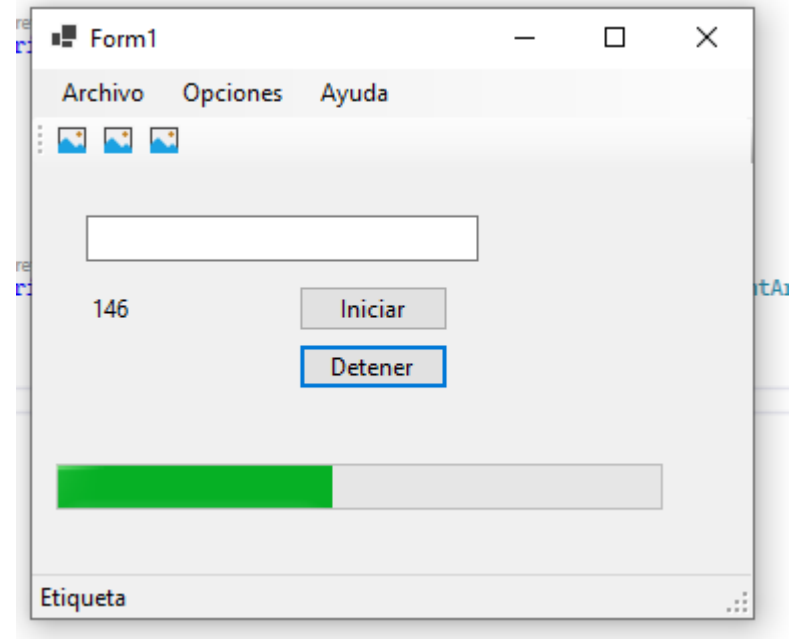
1.11 Componentes

Timers

Propiedades	
tmr_1 System.Windows.Forms.Timer	
(Name)	tmr_1
Enabled	False
GenerateMember	True
Interval	100
Modifiers	Private
Tag	

En milisegundos.

Progress bar
Track bar



“Value”

“Step” delimita el paso

“Perform Step”, no hay que preocuparse del desborde

1.11 Componentes

1.11 Componentes

1.11 Componentes

1.10.2

1.9 ANEXO C#.

1.10.x. Componentes

1.9 ANEXO C#.

Ejercicios 1.1

Ex1. Crear una aplicación de consola en .Net que se un juego de piedra, papel o tijera contra la máquina.

- Demanda elección por línea de comando.
- El ordenador tendrá que decidir su opción...
- Será una aplicación que no finalice.

Ex2. Crear una aplicación para una hamburguesería que permita hacer pedidos de comida y de el precio del pedido al final.

1.9 ANEXO C#.

Ejercicios 1.2

1. Pedir al usuario que introduzca 3 números por pantalla, una vez introducidos, mostrar por pantalla los números ordenados de menor a mayor.
2. El programa devolverá el área de un círculo, un cuadrado o un triángulo. Para ello se solicitará al usuario los datos necesarios por consola. Utilizar una estructura switch donde los case sean los tipos de figura.

$\text{Area_circulo} = \text{Pi} \times r^2$

$\text{Area_cuadrado} = b \times h$

$\text{Area_triangulo} = b \times h/2$

3. Mostrar por pantalla los primeros 100 números primos (Un número primo es aquel que únicamente puede ser dividido por 1 y por sí mismo sin residuos).

1.9 ANEXO C#.

Ejercicios 1.2

4. Mostrar por pantalla los números primos existentes entre 1 y 100.
5. Calcular el factorial del número introducido por consola(Ej: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$)
6. Introduciendo un valor de euros por consola, el programa devuelve el número mínimo de billetes y monedas necesarios para juntar la cantidad introducida.
7. Diseñar y desarrollar una calculadora por consola. Introcucción de los números por consola y selección de al operación.

1.9 ANEXO C#.

Ejercicios 1.3

8. (For) Cálculo de la tabla de multiplicar de cualquier número introducido por consola
9. (IFs) Crear un programa donde el usuario escriba una cantidad de dinero y el programa diga en qué billetes le vamos a dar esa cantidad. Ejemplo:

195: un billete de 100, un billete de 50, 2 billetes de 20, un billete de 5.

Se pueden poner monedas también si se quiere.

Mínimo hacerlo con billetes de 100, 50, 20, 10 y 5.

Introduciendo un valor de euros por consola, el programa devuelve el número mínimo de billetes y monedas necesarios para juntar la cantidad introducida.

10. (For) Suma de todos los número pares y todos los números impares de los números hasta una cifra introducida por consola.

1.9 ANEXO C#.

Ejercicios 1.3

11. (For, Math) Crear un ejercicio que te pida cuántos números va a meter el usuario, por ejemplo, 10 números.

Una vez tenemos la cantidad de números que nos va a dar, ¿qué tenemos que hacer con esos números? Sumarlos y sacar la media.

- 11-2 Vamos a elegir sumarlos, restarlos, multiplicarlos y dividirlos entre ellos. Si sale menor que cero, lanzar un mensaje de alerta.

12. (while) Tabla de multiplicar con while.

13. Generar un número aleatorio que no vea el usuario y tiene 3 intentos para acertarlo, en cada intento le diremos si se ha quedado bajo o alto, y si en las 3 veces no lo ha acertado le decimos que no ha acertado y le mostramos el número que es.

Si lo acierta paramos y le decimos que lo ha acertado.

1.9 ANEXO C#.

Ejercicios 1.3

14. (For, char) Abecedario descendente.

15. Hamburguesería

Se va a crear un interfaz de ventana de pedido de comida de un restaurante.

Un clase contendrá la información de los precios de los tipos de hamburguesa y 2 métodos: uno que diga el precio por selección y otro que diga los ingredientes, por selección.

Al final devolverá el precio de un pedido que puede incluir hamburgues, bebida de distintos tamaños y patatas.

1.9 ANEXO C#.

Ejercicios - Extra

ASCII Hex Símbolo			ASCII Hex Símbolo			ASCII Hex Símbolo			ASCII Hex Símbolo		
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

1.9 ANEXO C#.

Ejercicios - Extra

Convertir un dato STRING (cadena de caracteres) a byte (Enteros de 0 a 255):

Byte.Parse (CAN = byte.Parse(cad))

Convertir un dato STRING a Double (Decimales):

Double.Parse (SUE = double.Parse(linea);)

A Char (1 carácter):

OP = char.Parse(linea);

Convertir un dato STRING a Integer:

Int.Parse

Convert.ToInt32()

1.9 ANEXO C#.

Ejercicios - Extra

¿Que diferencia hay entre `convert.ToInt32()` con `Int.Parse()`?

Ambas tanto `int.Parse` como `Convert.ToInt32` se usan para convertir Strings (cuerdas) a Integers (enteros). Pero la diferencia es que `Convert.ToInt32` puede manejar data de type NULL y retorna '0' como resultado. `int.Parse` no puede manejar valor Null y te va a mostrar un `Argument Null Exception`.

Diferencia entre Float, Double y Decimal

Un valor del tipo float tiene una precisión de 7 dígitos, mientras que un valor del tipo double entre 15-16 dígitos. Por otra parte, un valor del tipo decimal, tiene una precisión de 28-29 dígitos.

Cambiar a Mayúsculas:

`ToUpper`

A minúsculas:

`ToLower`

1.9 ANEXO C#.

Ejercicios - Extra

Para debuguear (ir recorriéndolo) por el programa:

En el menú vamos a Depurar-Paso a paso por instrucciones, o vamos pulsando F11

Para poner un punto de interrupción:

Nos ponemos en la línea que queremos añadir el punto de interrupción y vamos al menú Depurar-Alternar puntos de interrupción o pulsamos F9.