El lenguaje C#

BOOM WARE TECHNOLOGIES

¿Qué no es .NET?

- El asesino de Java
- El asesino de Linux
- El asesino de Windows
- MS apropiándose de Internet
- Un complot para dominar el mundo

BOOM WARE TECHNOLOGIES

¿Qué es .NET?

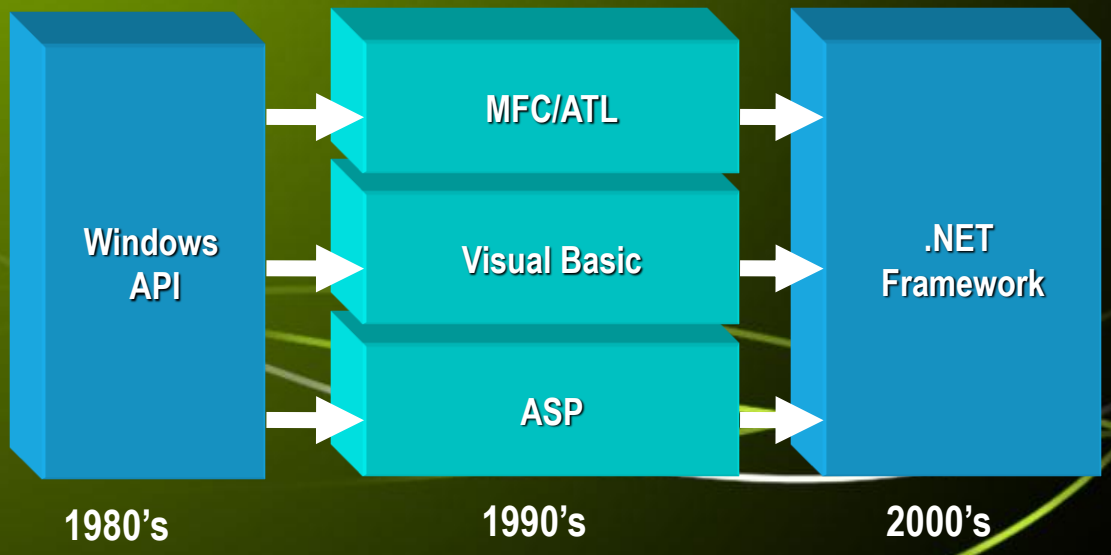
- Una plataforma software
  - Nuevo nivel: .NET Framework
  - Nueva herramienta: Visual Studio.NET
  - Unos servidores: .NET Enterprise Servers

BOOM WARE TECHNOLOGIES

Motivaciones
(Programación)

Motivado por el grado de complejidad que estaba tomando la programación Windows y el propio S.O:

- Interfaces de los API's para los diferentes lenguajes
- Multitud de servicios duplicados
- Pocas posibilidades de reutilización del código



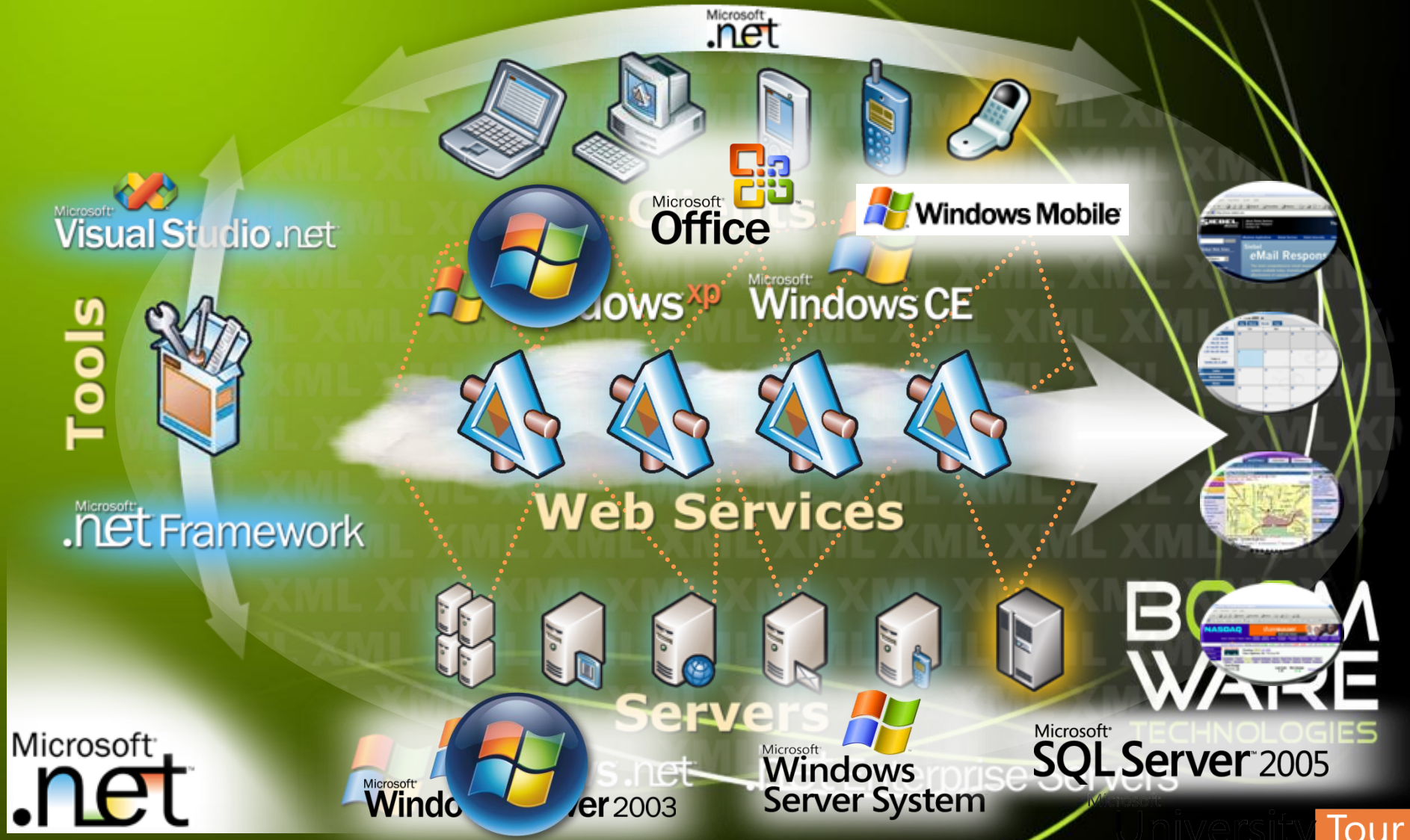
BOOM WARE TECHNOLOGIES

Motivaciones
(Despliegue)

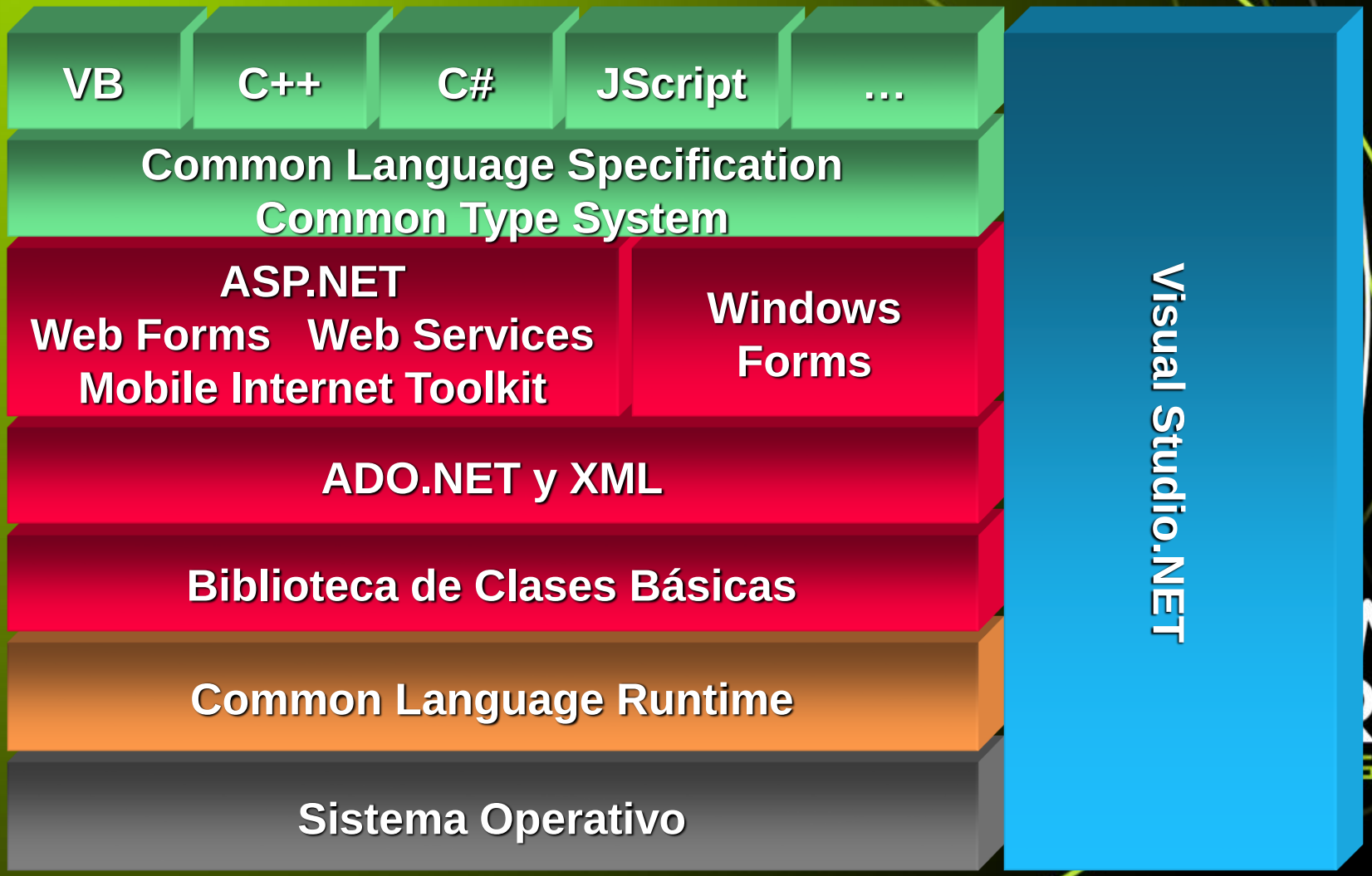
- Eliminar el “infierno de las DLL Win32”
  - Conflictos entre las aplicaciones con una librería en común en diferentes versiones

BOOM WARE TECHNOLOGIES

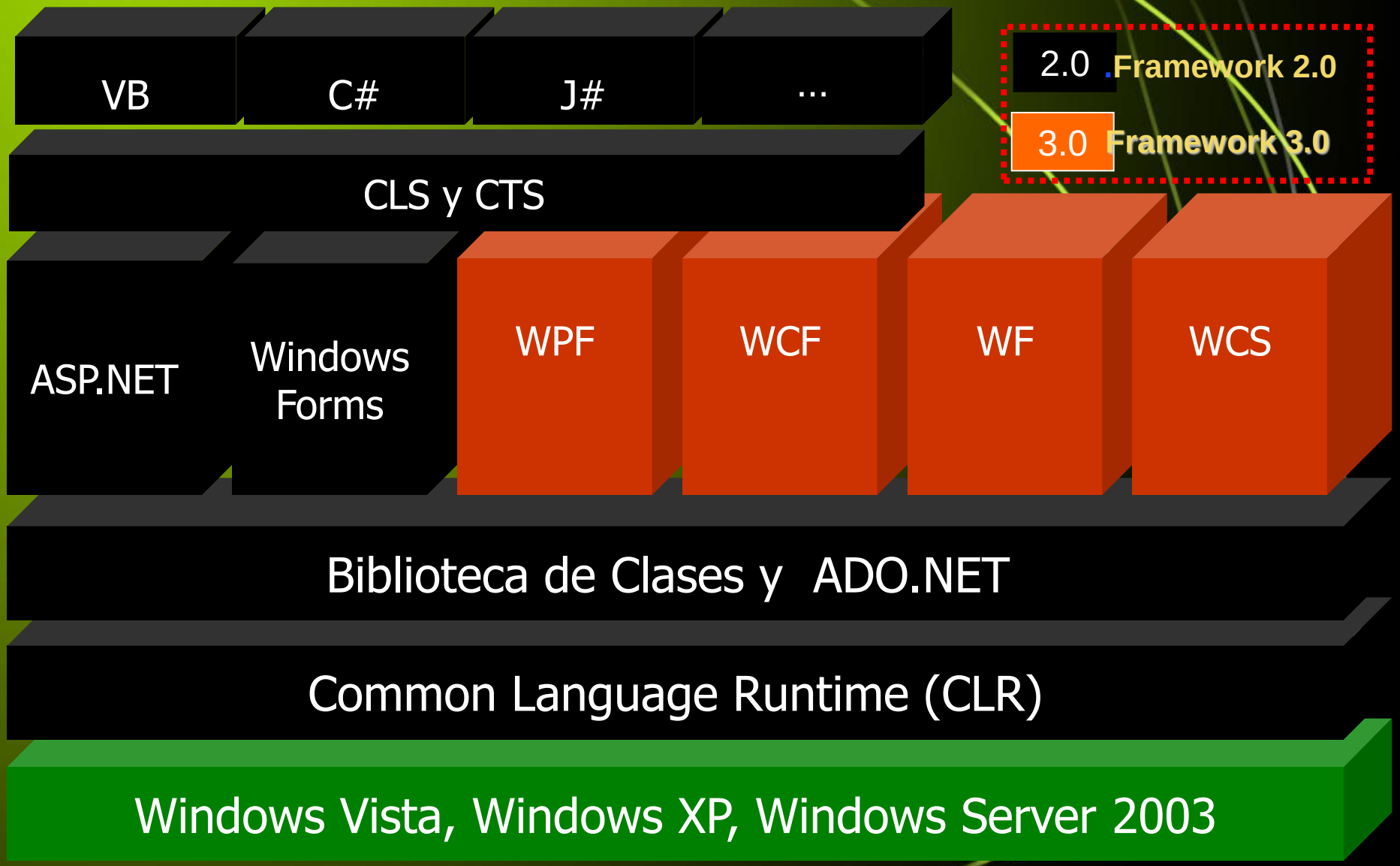
Plataforma.NET



NET Framework (2.0)



NET Framework (3.x)



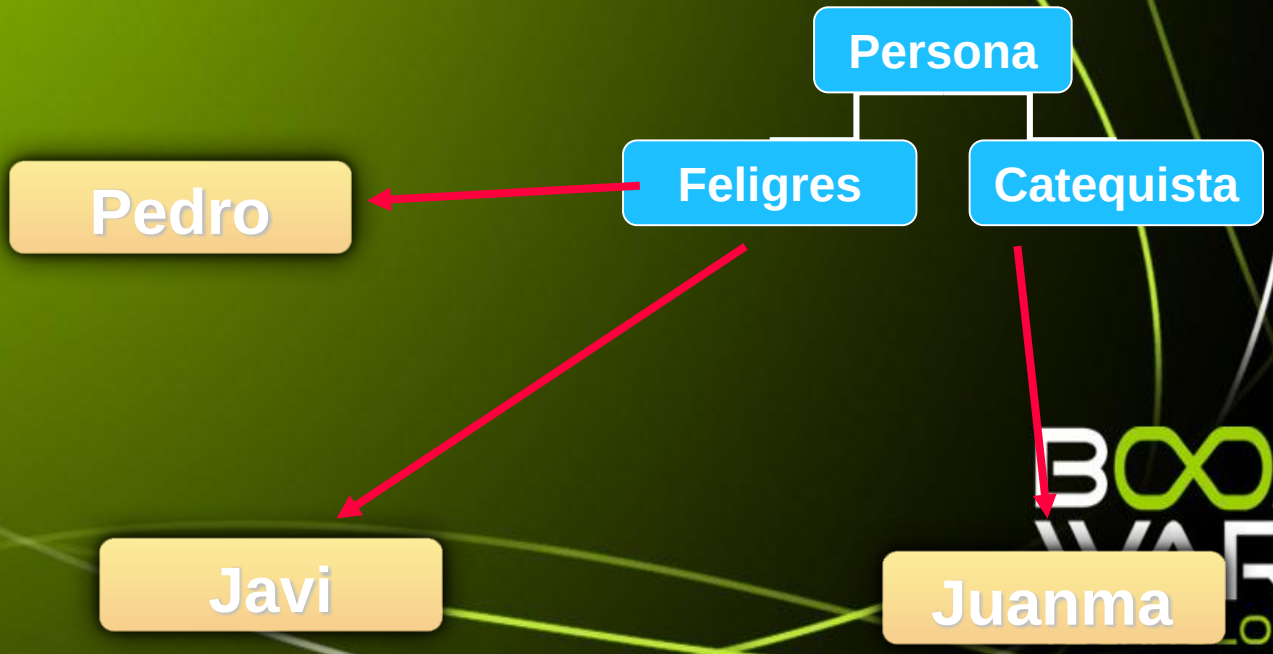
Algo de OO

- Programación Orientada a Objetos (POO)
  - Clases
    - Generador de objetos. Especifica la estructura, propiedades y métodos de los objetos.
  - Objetos
    - Una instancia de una clase.
  - Herencia
    - Relaciones que se pueden establecer entre las clases.
    - Hijos y padres.

BOOM WARE TECHNOLOGIES

Algo de OO (2)


Clase A
Clase B
Clase C
Clase D


Persona
Feligres
Catequista
Pedro
Javi
Juanma

¿C#, Cixarp, C Armoadiya…?

- El último en una línea de evolución lógica de los lenguajes derivados de C, como C++ y Java
- Usado por Microsoft para desarrollar la mayoría del código de .NET. Por tanto es el lenguaje ideal para el desarrollo en .NET.
- Más simple que C++ pero tan poderoso y flexible
  - C# ≈ “C++” + “Java”;
  - Facilitar migración.
  - Facilitar aprendizaje.

BOOM WARE TECHNOLOGIES

Características de C#

- Es un lenguaje sencillo y de alto nivel
- Es muy muy parecido a Java → "Si sabes Java, sabes C#"
- Es un lenguaje orientado a objetos
- Es un lenguaje moderno. Incluye elementos que no existen (o existían) en Java o en C++ y sin embargo se usan muy a menudo, como el bucle *foreach*
- Al igual que cualquier otro lenguaje para .NET, C# dispone de una gestión de memoria automática, aunque también permite liberar memoria manualmente mediante la instrucción *using*

BOOM WARE TECHNOLOGIES

Estructura de un programa en C#

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

namespace MiPrograma
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hola a todos!!");
        }
    }
}
```

Librerías (*namespaces*) usadas

Método principal (obligatorio)

BOOM WARE TECHNOLOGIES

Variables

TIPOS DE VARIABLE

Variables tratadas **por valor**
(contienen el valor)

- **Tipos primitivos (entero, carácter, lógico, etc)**
- **Tipos enumerados**
- **Estructuras**

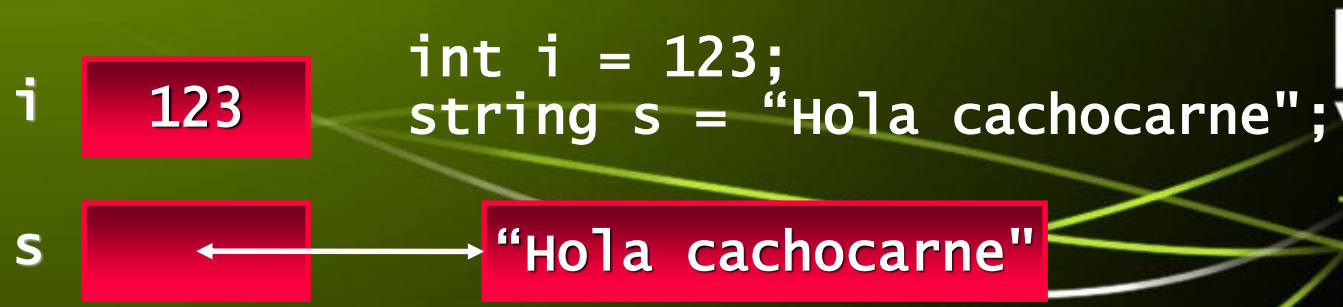
Variables tratadas **por referencia**
(contienen una referencia
al valor)

- **Clases**
- **Interfaces**
- **Arrays**
- **Delegados**

BOOM WARE TECHNOLOGIES

Variables (2)

- **Por valor:**
  - Continen el dato
  - No pueden ser nulas
- **Por referencia:**
  - Contienen la referencia
  - Pueden ser nulas
  - Se crean con “new” (Excepto los strings)



BOOM WARE TECHNOLOGIES

Variables (3)

- Predefinidos
  - Referencia `object, string`
  - Con signo `sbyte, short, int, long`
  - Sin signo `byte, ushort, uint, ulong`
  - Caracter `char`
  - Flotantes `float, double, decimal`
  - Lógico `bool`
- Son como un alias a los tipos definidos en system.
- `int == System.Int32`
- Casting: `float e = (float) f;`
- String -> Tipo: `int e = int.Parse("234");`
- Tipo ->String: `string s = e.ToString();`

BOOM WARE TECHNOLOGIES

Variables (4)

- Las variables por valor pueden declararse e iniciarse:

```
bool bln = true;
byte byt1 = 22;
char ch1='x', ch2='\u0066'; // Unicode para 'a'
decimal dec1 = 1.23M;
double dbl1=1.23, dbl2=1.23D;
short sh = 22;
int i = 22;
long lng1 =22, lng2 =22L; // 'L' long
sbyte sb = 22;
float f=1.23F;
ushort us1=22;
uint ui1=22, ui2=22U; // 'U' unsigned
ulong ul1 =22, ul2=22U, ul3=22L, ul4=2UL;
```

BOOM WARE TECHNOLOGIES

Variables (5)

- Los valores por referencia son creados con la palabra clave **new**:

```
object o = new System.Object();
```

- **Strings** se pueden inicializar directamente:

```
string s = "Hola"; // usan caracteres Unicode de 2
   bytes
```

- C# soporta secuencias de escape como en C:

```
string s1 = "Hola\n"; // salto de línea
string s2 = "Hola\tque\ttal"; // tabulador
```

- Como las sentencias de escape comienzan con '\', para escribir este carácter hay que doblarlo, o usar '@':

```
string s3 = "c:\\WINNT";
string s4 = @"C:\WINNT";
```

BOOM WARE TECHNOLOGIES

Variables:
Tipos enumerados (1)

Un enumerado es un tipo de variable cuyos valores posibles que puede tomar son especificados por nosotros

Definición del tipo:

enum <nombreTipoEnumerado> { valor1, valor2, … , valorN }

Declaración de una variable:

<nombreTipoEnumerado> <nombreVariable>;

Asignación de una variable:

<nombreVariable> = <nombreTipoEnumerado>.valor;

BOOM WARE TECHNOLOGIES

Variables:
Tipos enumerados (2)

Ejemplo:

```
enum ColorFeo { limon, celeste, fucsia, pistacho }

ColorFeo c;
c = ColorFeo.fucsia;
```

→ **Se declara como cualquier otra variable**

→ **Al valor le precede siempre el nombre del enumerado**

BOOM WARE TECHNOLOGIES

Variables: Estructuras (1)

- Una **estructura** es un tipo de dato similar a una clase, pero con dos diferencias principalmente:
  - Se tratan como variables por valor
  - Son más rápidas y ocupan menos memoria y, como consecuencia de esto, no tienen herencia

BOOM WARE TECHNOLOGIES

Variables: Estructuras (2)

➪ Ejemplo:

```
public struct Moto
{
        public string marca;
        public string modelo;

        public Moto(string marca, string modelo)
        {
                this.marca = marca;
                this.modelo = modelo;
        }
}
```

BOOM WARE TECHNOLOGIES

Arrays (1)

- Un array (unidimensional) o tabla es un tipo especial de variable que es capaz de almacenar de manera ordenada 1 ó más datos de un determinado tipo
- El valor de un array es una referencia, por lo que se comporta como un objeto más
- Tienen algunos métodos y propiedades útiles, como la propiedad “Length”, que obtiene el número de elementos que tiene el array
- Un array se crea así:

<tipo>[ ] <nombreVariable> = new <tipo> [<numeroElementos>]

BOOM WARE TECHNOLOGIES

Arrays (2)

Ejemplos:

Tabla de enteros:

```
int[] dni = new int[6];

dni[0] = 2;
dni[1] = 4;
dni[2] = 0;
dni[3] = 2;
dni[4] = 1;
dni[5] = 3;

dni[0] = dni[3] + dni[5];
```

Tabla de *Personas*:

```
Persona[] alumnos = new Persona[3];

alumnos[0] = new Persona(Carlos, 20);
alumnos[1] = new Persona(Antonio, 18);
alumnos[2] = new Persona(Elena, 21);

alumnos[1].imprimirPorPantalla();
```

El índice del primer elemento siempre es 0 y el último *longitud – 1*

BOOM WARE TECHNOLOGIES

Arrays (3)

- Los arrays también pueden tener más de 1 dimensión

  Ejemplos:

  - Array de *lógicos* de 2 dimensiones:

```
bool[,] aprobados = new bool[5, 100];

aprobados[2, 54] = true;
```

  - Array de *cadenas* de 3 dimensiones:

```
string[,,] profesores = new string[5, 5, 100];

aprobados[0, 0, 4] = “Paco Pérez”;
```

BOOM WARE TECHNOLOGIES

Constantes (1)

- Una constante es una variable cuyo valor no puede cambiar durante la ejecución del programa
- El compilador sustituye en el código generado todas las referencias a las constantes por sus respectivos valores, por lo que el código generado será más eficiente, ya que no incluirá el acceso y cálculo de sus valores
- Hay que darles un valor inicial al declararlas:

```
const <tipoConstante> <nombreConstante> = <valor>;
```

Constantes (2)

- Ejemplo de constantes correctas:

```
const int a = 123;

const int b = 24;
const int c = b + 125;
```

El valor siempre debe ser constante

- Ejemplo de constantes incorrectas:

```
int x = 123;
const int y = x + 123;
```

ERROR: *x* no tiene porqué tener valor constante (aunque aquí lo tenga)

Operadores (1)

- De asignación:
  - a = 4;
  - a = b = 2;
  - • Asignación (=)
- Aritméticos:
  - c = 4 + 3;
  - • Suma (+)
  - • Resta (-)
  - • Producto (*)
  - • División (/)
  - • Módulo (%)
- Lógicos:
  - c = a && b;
  - d = !e;
  - • AND con Evaluación Perezosa (&&)
  - • AND sin Evaluación Perezosa (&)
  - • OR con Evaluación Perezosa (||)
  - • OR sin Evaluación Perezosa (|)
  - • NOT (!)
  - • XOR (^)

Asignación y aritmeticos

+=

-=

***=**

/=

%=

Operadores (2)

- Relacionales:

```
a = (b == c);
d = (e > f);
```

- Igualdad (==)
- Desigualdad (!=)
- Mayor que (>)
- Menor que (<)
- Mayor o igual que (>=)
- Menor o igual que (<=)

- A Nivel de Bits

- Desplazamiento Izda. (<<)
- Desplazamiento Dcha. (>>)

```
int i1=32;
int i2=i1<<2; //
i2==128
int i3=i1>>3; //
i3==4
```

BOOM WARE TECHNOLOGIES

Sentencias (1)

- Instrucción IF:

  Permite ejecutar ciertas instrucciones sólo si se cumple una determinada condición

```
if (<condición>)
{
    <instrucciones>
}
else
{
    <instrucciones>
}
```

BOOM WARE TECHNOLOGIES

Sentencias (2)

Ejemplos:

```
public bool esUnaNotaCorrecta(int nota)
{
    bool correcta;

    if (nota >= 0 || nota <= 10)
    {
        correcta = true;
    }
    else
    {
        correcta = false;
    }

    return correcta;
}
```

BOOM WARE TECHNOLOGIES

Sentencias (3)

```
public void saludo(int numeroSaludo)
{
	if (numeroSaludo == 1)
	{
		Console.WriteLine("Hola, esto es el saludo nº 1");
	}
	else if (numeroSaludo == 2)
	{
		Console.WriteLine("Hola, esto es el saludo nº 2");
	}
	else
	{
		Console.WriteLine("Saludo incorrecto");
	}
}
```

BOOM WARE TECHNOLOGIES

Sentencias (4)

Instrucción SWITCH

- **Permite ejecutar unos u otros bloques de instrucciones según el valor de una cierta expresión**
- **Ejemplo: (Dada una variable *i* de tipo entero)**

```
switch (i)
{
      case 1:
            Console.WriteLine("Primer caso");
            break;
      case 2:
            Console.WriteLine("Segundo caso");
            break;
      default:
            Console.WriteLine("Cuaquier otro caso");
            break;
}
```

Se puede usar *break* o *goto* (para salto de línea) aunque es obligatorio usar Uno de ellos.

BOOM WARE TECHNOLOGIES

Sentencias (5)

Instrucción FOR:

Sirve para repetir un una serie de instrucciones un determinado número de veces.

Esquema:

```
for (<inicialización>; <condición>; <modificación>)
{
    <instrucciones>
}
```

BOOM WARE TECHNOLOGIES

Sentencias (6)

Ejemplo:

```
for (int i = 0; i<10; i++)
{
      string texto = “Esta es la iteración nº “ + i;
      Console.WriteLine(texto);
}
```

BOOM WARE TECHNOLOGIES

Sentencias (7)

➪ Instrucción WHILE:

Sirve repetir una serie de instrucciones mientras se cumpla una condición

Esquema:

```
while (<condición>)
{
  <instrucciones>
}
```

BOOM WARE TECHNOLOGIES

Sentencias (8)

Ejemplo:

```
int i = 0;

while (i<10)
{
      string texto = “Esta es la iteración nº “ + i;
      Console.WriteLine(texto);
      i++;
}
```

BOOM WARE TECHNOLOGIES

Sentencias (9)

➪ Instrucción DO WHILE:

Sirve repetir una serie de instrucciones mientras se cumpla una condición y al menos se ejecutan 1 vez.

Esquema:

```
do
{
  <instrucciones>
}
while (<condición>)
```

BOOM WARE TECHNOLOGIES

Sentencias (10)

Ejemplo:

```
int i = 0;

do
{
        string texto = “Esta es la iteración nº “ + i;
        Console.WriteLine(texto);
        i++;
}
while (i<10)
```

BOOM WARE TECHNOLOGIES

Sentencias (11)

Instrucción FOREACH

- Es una variante de la instrucción *for* pensada para arrays y otras colecciones (siempre que todos sus elementos sean del mismo tipo)
- Sirve para recorrer arrays, como los iteradores

Esquema:

```
foreach (<tipoElemento> <elemento> in <colección>)
{
    <instrucciones>
}
```

BOOM WARE TECHNOLOGIES

Sentencias (12)

Ejemplo:

```
string[] equipos = new string[4];

equipos[0] = "Barcelona";
equipos[1] = "Sevilla";
equipos[2] = "Real Madrid";
equipos[3] = "Valencia";

foreach(string eq in equipos)
{
        Console.WriteLine(eq);
}
```

En cada iteración, *eq* vale lo que Valga el elemento correspondiente Del array (siempre empezando por El primer elemento, el 0)

BOOM WARE TECHNOLOGIES

Clases (1)

- Un **objeto** es un conjunto de datos (variables) y de métodos (funciones) que permiten manipular dichos datos

- Una **clase** es la definición de las características concretas de un determinado tipo de objetos

Una clase es como una estructura del lenguaje C, pero que, además de contener subvariables, contiene funciones.

BOOM WARE TECHNOLOGIES

Clases (2)

Estructura de una clase:

```
<modificador> class <nombreClase>
{
        Atributos
        Constructor/es
        Métodos
}
```

Declaración de un objeto:

```
<nombreClase> <nombreVariable>;
```

Asignación/creación de un objeto:

```
<nombreVariable> = new <nombreClase>();
```

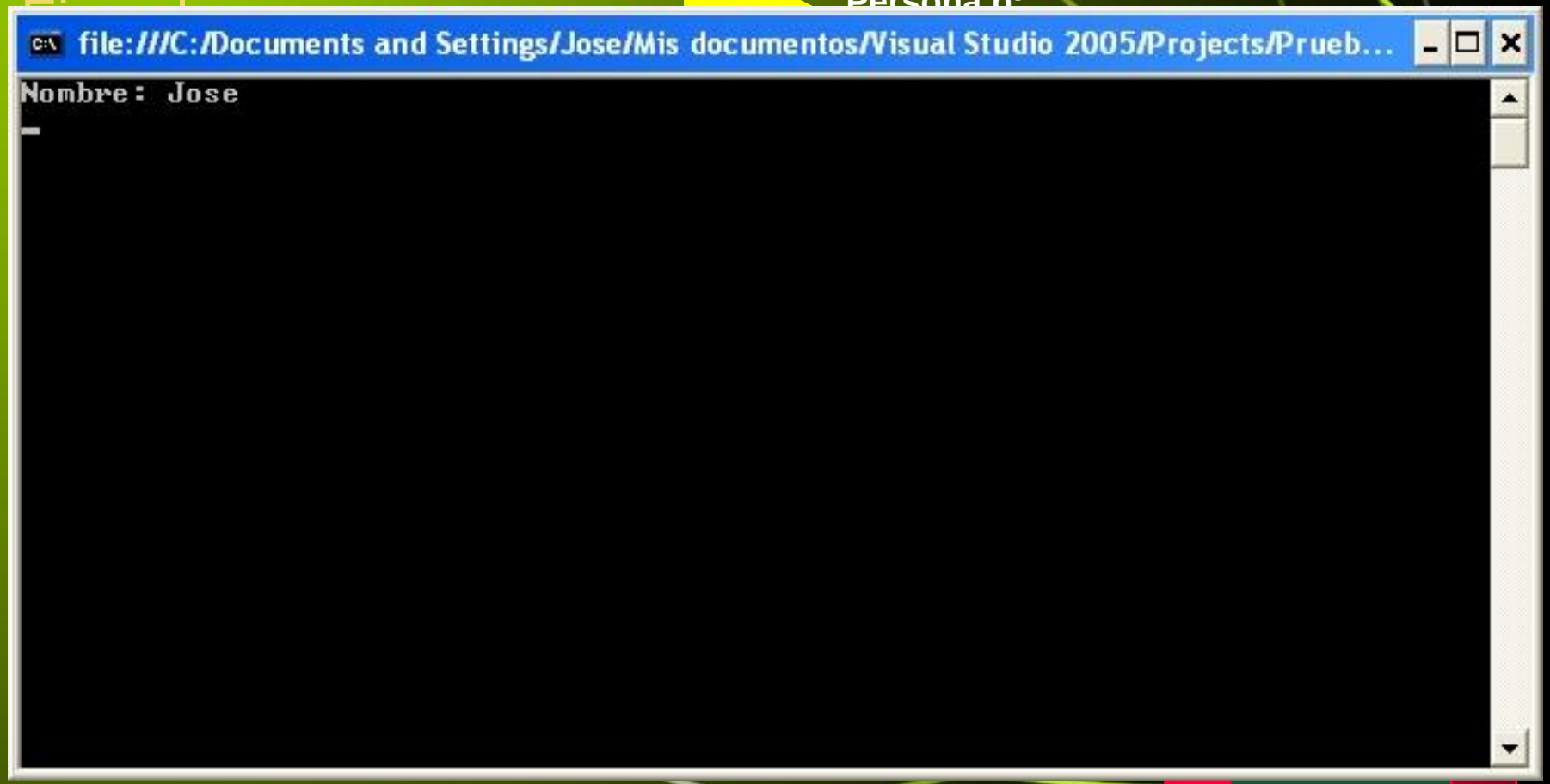
El objeto se crea con el operador *new* seguido de una llamada al constructor que elijamos

- Un **atributo** es una variable de cualquier tipo que estará contenida dentro de cada objeto que se cree de la clase en cuestión (en cada objeto tendrá un valor distinto)
- Un **método** es una función que permite modificar los atributos del objeto desde el que se llama
- Un **constructor** es un método o función que se ejecuta automáticamente al crear una variable de la clase en cuestión. Se puede omitir (dando lugar al constructor por defecto), aunque suele utilizarse para dar valores iniciales a los atributos.

  Puede haber más de 1, pero sólo se ejecutará uno de ellos, escogiendo cuál se ejecuta al crear el objeto

Clases (3)

Persona p;



}

wf35gd

Clases:
Propiedades (1)

- Una **propiedad** es una mezcla entre el concepto de atributo (variable) y el concepto de método (función).

- Externamente es accedida como si de un atributo normal se tratase, pero internamente es posible asociar código a ejecutar en cada asignación o lectura de su valor.

  - Éste código puede usarse para comprobar que no se asignen valores incorrectos, etc.

BOOM WARE TECHNOLOGIES

Clases: Propiedades (2)

Ejemplo:

```
public class Persona
{
        public string nombre;
        private int edad;

        public int Edad
        {
            get
            {
                return edad;
            }
            set
            {
                if (edad < 0 || edad > 120) edad = 0;
                else edad = value;
            }
        }
}
```

```
Persona a = new Persona();

a.nombre = “Fran”;          → Se permite
                              calquier valor

a.Edad = 22;                → Valor permitido

a.Edad = 121;               → Valor prohibido
```

- ***return*** devuelve el valor de *edad*
- ***value*** es el valor que se intenta asignar a *edad*

BOOM WARE TECHNOLOGIES

Clases: Propiedades (3)

```
public class Person
{
  private int age;

  public int Age {
    get { return age; }
    set { age = value;
          Console.WriteLine(age);
}
   }
}
```

Propiedad

C#

```
Person p = new Person();
p.Age = 27;
p.Age++;
```

En vez de

Otro

```
Person p = new Person();
p.setAge(27);
p.setAge(p.getAge() + 1);
```

Clases parciales

```
// Demo.Part1.cs

using System;
public partial class Demo
{
public Demo()
{
        Console.Write( "P1" );
}
}
```

```
// Demo.Part2.cs
public partial class Demo
{
private int i;
}
```

```
// Demo.Part3.cs
// Error!
public class Demo
{
// Error!
private int i;

// OK
        private int j;

    public void Test()
{       // Error!       Console.Write
"P3" );         }
}
```

BOOM WARE TECHNOLOGIES

Clases y más…

- Un miembro `static` puede ser accedido sin crear una instancia de una clase (se suelen usar para guardar valores globales)

```
class Persona {
    public static int MinimumAge = 18;
    …
}
int age = Persona.MinimumAge; // accedemos a MinimumAge usando nombre clase
```

- Las clases se pueden anidar:

```
class C1 {
    int i, j;   string s;
    void m() {  // …   }
    class c2 {
       // …
    }
}
```

BOOM WARE TECHNOLOGIES

Herencia de clases (1)

- Es un mecanismo que permite definir nuevas clases a partir de otras ya definidas
- Si una clase A deriva de una clase B, el compilador, al compilar, incluye en la clase A el contenido de la B
- Al igual que en Java, sólo se permite la herencia simple, es decir, una clase no puede heredar de más de 1 clase
- Esquema:

```
<modificador> class <nombreClaseHija>:<nombreClasePadre>
{
   <miembrosHija>
}
```

BOOM WARE TECHNOLOGIES

Herencia de clases (2)

- Ejemplo

```
public class Persona
{
        public string nombre;
        public int edad;

        public Persona(string nombre, int edad)
        {
            this.nombre = nombre;
            this.edad = edad;
        }

        public void imprimirDatos()
        {
            string datos = "Nombre: " + nombre + ", Edad: " + edad;
            Console.WriteLine(datos);
        }
}
```

BOOM WARE TECHNOLOGIES

Herencia de clases (3)

```
public class Policia : Persona
{
        public long numeroPlaca;

        public Policia(string nombre, int edad, long numeroPlaca)
            : base(nombre, edad)
        {
            this.numeroPlaca = numeroPlaca;
        }
}
```

BOOM WARE TECHNOLOGIES

Herencia de clases (4)

- Modificadores

  - Los modificadores de acceso controlan la visibilidad de los atributos y métodos de una clase

  - Modificadores más importantes:

    - *public*: Visible fuera de la clase
    - *private* : Visible sólo dentro de la clase
    - *protected*: Visible dentro de la clase y de sus clases hija

BOOM WARE TECHNOLOGIES

Herencia de clases (5)

- Redefinición de métodos

Para redefinir un método, usamos la palabra ***virtual*** en el método de la clase padre que queremos redefinir y la palabra ***override*** en el método nuevo

BOOM WARE TECHNOLOGIES

Herencia de clases (6)

Ejemplo:

```
public class Persona
{
        public string nombre;
        public int edad;

        public Persona(string nombre, int edad)
        {
            this.nombre = nombre;
            this.edad = edad;
        }

        public virtual void imprimirDatos()
        {
            string datos = "Nombre: " + nombre + ", Edad: " + edad;
            Console.WriteLine(datos);
        }
}
```

BOOM WARE TECHNOLOGIES

Herencia de clases (7)

```
public class Policia : Persona
{
    public long numeroPlaca;

    public Policia(string nombre, int edad, long numeroPlaca)
        : base(nombre, edad)
    {
        this.numeroPlaca = numeroPlaca;
    }

    public override void imprimirDatos()
    {
        string datos = "Nombre: " + nombre + ", Edad: " + edad +
        ", Número placa: " + numeroPlaca;
        Console.WriteLine(datos);
    }
}
```

BOOM WARE TECHNOLOGIES

Clases abstractas (1)

- Supongamos que estamos interesados en modelar hombres y mujeres, pero no personas *perse* → No queremos permitir la creación de objetos `Persona` directamente, pero queremos permitir la creación de objetos `Hombre` y `Mujer` directamente.
- Usamos `abstract` delante de `Persona` para evitar que esta clase se pueda instanciar
- Usamos `abstract` en el método `Saludar` de `Persona` para indicar que las clases que heredan deben sobreescribirlo

Clases abstractas (2)

```
using System;
abstract class Persona {
    protected string nombre, apellido1;
    public Persona(string nombre, string
  apellido1) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
    }
    abstract public void Saludar();
}
```

Clases abstractas (3)

```
class Hombre: Persona {
    public Hombre(string nombre, string apellido1):
   base(nombre, apellido1) {}
    public override void Saludar() {
        Console.WriteLine("¡Hola Señor " +
   this.apellido1 + "!");
    }
}
class Mujer: Persona {
    public Mujer(string nombre, string apellido1):
   base(nombre, apellido1) {}
    public override void Saludar() {
        Console.WriteLine("¡Hola Señorita " +
   this.apellido1 + "!");
    }
}
```

BOOM WARE TECHNOLOGIES

Clases abstractas (4)

```
class Test {
    public static void Main() {
        Hombre h = new Hombre("Diego", "Ipiña");
        h.Saludar(); // Visualizar ¡Hola Señor Ipiña!
        Mujer m = new Mujer("Usue", "Artaza");
        m.Saludar(); // Visualizar ¡Hola Señorita Artaza!
    }
}
```

- Para deshabilitar la herencia se puede marcar una clase como sealed, resultando en un error de compilación si se intenta derivar de ella

```
sealed class Persona {
    …
}
```

BOOM WARE TECHNOLOGIES

Clases más usuales

- String: Manejo de Cadenas
- StringBuilder: Modificacion de Cadenas
- Math: funciones Matematicas
- Random: Números aleatorios
- DateTime: Fecha y Hora
- FileStream, StreamWriter, StreamReader: Ficheros
- File, Directory y Path: Sistema de ficheros
- …

BOOM WARE TECHNOLOGIES

Interfaces (1)

- Una interfaz especifica un “contrato” que una clase (o una estructura) debe cumplir. Fijan el comportamiento
- Una interfaz contiene las cabeceras de los métodos (y propiedades) que debe tener como mínimo una clase
- Una clase puede implementar 1 ó más interfaces
- Las interfaces sirven para abstraerse y crear tipos de variables que admitan distintas implementaciones, especificando sólo lo básico del tipo (sus miembros)

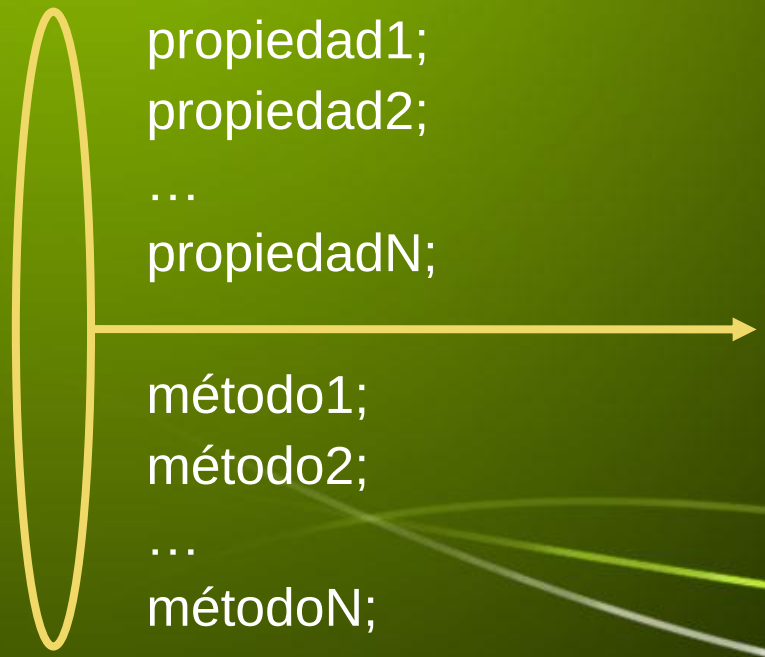
BOOM WARE TECHNOLOGIES

Interfaces (2)

Estructura de una interfaz:

```
<modificador> interface <nombreInterfaz>
{
        propiedad1;
        propiedad2;
        …
        propiedadN;

        método1;
        método2;
        …
        métodoN;
}
```



Los miembros de una interfaz no llevan nunca modificadores, se suponen siempre como *public*

BOOM WARE TECHNOLOGIES

Interfaces (3)

Ejemplo:

```
public interface Coche
{
        string getMarca();
        int avanza(int tiempo);
}
```

getMarca → Devuelve la marca

avanza → Devuelve los km recorridos en un tiempo determinado

```
public class Renault : Coche
{
        private string marca;
        private int metrosPorSegundo;

        public Renault()
        {
            marca = "Renault";
            metrosPorSegundo = 2;
        }

        public string getMarca()
        {
            return marca;
        }

        public int avanza(int tiempo)
        {
            int m = metrosPorSegundo * tiempo;
            return m;
        }

        public int getMetrosPorSegundo()
        {
            return metrosPorSegundo;
        }
}
```

Interfaces (4)

- Podemos crear un objeto tipo Clase o tipo Interfaz:

Tipo Clase:

```
Renault miCoche = new Renault();
```

Podemos acceder a todos sus métodos

Tipo Interfaz:

```
Coche miCoche = new Renault();
```

Podemos acceder a todos sus métodos que estén en la interfaz

BOOM WARE TECHNOLOGIES

Herencia de interfaces (1)

- Nos permite incluir la definición de una interfaz en otra
- Si una interfaz A deriva de una interfaz B, el compilador, al compilar, incluye en la interfaz A el contenido de la B
- A diferencia de las clases, las interfaces pueden heredar de más de 1 interfaz (herencia múltiple)

Herencia de interfaces (2)

Ejemplo:

```
public interface Persona
{
     string getNombre();
     int getEdad();
}
```

```
public interface Deportista : Persona
{
     long getCaloriasQuemadas();
}
```

```
public interface Famoso
{
    bool esMuyFamoso();
}
```

```
public interface JugadorLFP : Deportista, Famoso
{
     string getEquipo();
}
```

BOOM WARE TECHNOLOGIES

Sobre Métodos (1)

- Los métodos acceptan parámetros de tipo primitivo y devuelven un resultado

```
int Add(int x, int y) {
    return x+y;
}
```

- Los parámetros x e y se pasan por valor, se recibe una copia de ellos
- Si queremos modificar dentro de una función un parámetro y que el cambio se refleje en el código de invocación de la función, usaremos **ref**, tanto en la declaración del método como en su invocación:

```
void Increment(ref int i) {
    i = 3;
}
…
int i;
Increment(ref i); // i == 3
```

BOOM WARE TECHNOLOGIES

Sobre Métodos (2)

- Mientras **ref** se usa para modificar el valor de una variable, si el método asigna el valor inicial a la variable habrá que usar **out**:
- Para pasar un numero variable de parámetros se usa **params**:

```
class Adder {
    int Add(params int[] ints) {
        int sum=0;
        foreach(int i in ints)
            sum+=i;
        return sum;
    }
    public static void Main() {
        Adder a = new Adder();
        int sum = a.Add(1, 2, 3, 4, 5);
        System.Console.WriteLine(sum); // visualiza "15"
    }
}
```

BOOM WARE TECHNOLOGIES

Sobre Métodos (3)

- Métodos de **sobrecarga**, es decir, métodos con el mismo nombre, con firmas (número y tipo de parámetros) diferentes:

```
class Persona {
    string nombre, apellido1, apellido2;
    public Persona(string nombre, string apellido1) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
    }
    public Persona(string nombre, string apellido1, string
    apellido2) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
        this.apellido2 = apellido2;
    }
}
Persona p1 = new Persona("Diego", "Ipiña");
Persona p2 = new Persona("Diego", "Ipiña", "Artaza");
```

Sobre Métodos (4)

- Sobrecarga de Operadores

[acceso] static NombreClase operator+(Tipo a[, Tipo b])

```
public class Metros
{
    private double cantidad=0;

   public Metros(double cant)
    {
        this.cantidad=cant;
    }
    public static Metros operator+(Metros m, Centimetros c)
    {
        Metros retValue=new Metros();
        retValue.Cantidad=m.Cantidad+c.Cantidad/100;

        return retValue;
    }
}
…
Metros SumaMetros=m-c;
```

BOOM WARE TECHNOLOGIES

Modificadores de acceso

- Los modificadores de acceso controlan la visibilidad de los miembros de una clase
  - `private`, sólo código dentro de la misma clase contenedora tiene acceso a un miembro privado. Es el modo de acceso por defecto.
  - `public` , visible a todos los usuarios de una clase
  - `protected`, miembros accesibles tanto por dentro de la clase como en clases derivadas
  - `internal`, miembros accesibles sólo dentro de un assembly
  - `protected internal`, permite acceso protected e internal

Excepciones (1)

- Una excepción es un objeto de alguna clase derivada de *System.Exception* que se genera cuando se produce algún error en tiempo de ejecución y que contiene información sobre el mismo.
- El código donde pueda haber errores se trata con el bloque *try…catch*
- Hay dos tipos de excepciones:
  - **Las excepciones predefinidas por .NET:**

    Si ocurre algún error relacionado con estas excepciones, la excepción salta automáticamente
  - **Las excepciones definidas por el programador:**

    Para que salte una excepción definida por el programador se tiene que indicar explícitamente en el código
- A diferencia de Java, no hace falta propagar las excepciones, se pueden tratar tanto en la función que se produce como en otra función que a su vez llama a la función en la que se produce

Excepciones (2)

Ejemplo:

```
Public class Program
{
     public static void Main(string[] args)
     {
          int a = 3;
          int b = 0;
          int r = Calculadora.dividir(a, b);
     }
}
```

BOOM WARE TECHNOLOGIES

Excepciones (3)

- `ArgumentNullException` → una referencia nula es pasada como argumento
- `ArgumentOutOfRangeException` → nos hemos salido de rango, e.j. entero demasiado grande
- `DividedByZeroException`
- `IndexOutOfRangeException` → se usa un índice inválidado del array
- `InvalidCastException`
- `NullReferenceException` → se intenta invocar un método en un objeto que es null
- `OutOfMemoryException`

BOOM WARE TECHNOLOGIES

Excepciones (4)

```
using System;
class Persona {
    string nombre, apellido1;
    int edad;
    public Persona(string nombre, string apellido1, int
   edad) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
        this.edad = edad;
        if (edad < 18) // La edad debe ser mayor que 18 sino
   excepción
            throw new Exception("ERROR: Persona debajo edad
   legal");
        this.edad = edad;
    }
}
```

Excepciones (5)

```
class Test {
   public static void Main() {
      try {
         Persona p = new Persona("Diego, "Ipiña", 12);
      } catch (Exception e) { // capturar excepción lanzada
         Console.WriteLine(e.Message);
      }
}
```

- Se puede incluir un bloque finally también, para asegurarnos que recursos abiertos son cerrados:

```
try {
   ...
} catch {
   ...
} finally {
   ...
}
```

BOOM WARE TECHNOLOGIES

Delegados y eventos (1)

- Un **delegado** es un tipo especial de clase cuyos objetos pueden almacenar referencias a uno o más métodos de tal manera que a través del objeto sea posible solicitar la ejecución en cadena de todos ellos.

- No se crea como una clase normal, sino mediante esta sintaxis:

  <modificadores> **delegate** <tipoRetorno> <nombreDelegado> (<parámetros>)**;**

- Los métodos que almacene un objeto de un delegado tienen que tener el mismo tipo de salida y los mismos parámetros.

- Se suelen usar para manejar eventos. Los eventos se manejan mucho en las aplicaciones con una interfaz de usuario gráfico (con ventana).

- También se usan para ejecutar funciones de forma asíncrona.

BOOM WARE TECHNOLOGIES

Delegados y eventos (2)

Ejemplo:

```
public class Program
{
    public static void Main(string[] args)
    {
        FuncionSuma func = new FuncionSuma(suma);
        int c = func(3, 1);                    // Devuelve 4

        func = new FuncionSuma(resta);
        int d = func(3, 1);                    // Devuelve 2
    }

    public static int suma(int x, int y)
    {
        return x + y;
    }

    public static int resta(int x, int y)
    {
        return x - y;
    }
}

public delegate int FuncionSuma(int a, int b);
```

BOOM WARE TECHNOLOGIES

Colecciones (1)

- La plataforma .NET tiene un espacio de nombres dedicado a estructuras de datos como pilas, arrays dinámicos, colas...
- `System.Collections` dispone de una serie de interfaces que implementan todas estas colecciones.

BOOM WARE TECHNOLOGIES

Colecciones (2)

- Entre otras estructuras podemos encontrar:
  - `ArrayList`: Array dinámico.
  - Vector: Vector de Datos
  - `HashTable`: Tablas de Hashing.
  - `Queue`: Colas.
  - `SortedList`: Listas ordenadas.
  - `Stack`: Pilas.
  - `BitArray`: array de bits (guardan booleanos)
- Todas las colecciones menos `BitArray` guardan objetos de tipo `System.Object`

C# 2.0: Colecciones Genéricas

- Similares a los Templates de C++
  - `Dictionary`: Clave y Valores
  - `LinkedList`: Lista enlazada
  - `List`: Lista
  - `Queue`: Colas
  - `Stack`: Pilas

```
List<Persona> personas = new List<Persona>();
personas.Add(new Persona("José", "García", new DateTime(1940, 12, 2)));
personas.Add(new Persona("Pedro", "López", new DateTime(1990, 2, 22)));
personas.Add(new Persona("Antonio", "Pérez", new DateTime(1976, 6, 21)));
…
foreach (Persona persona in personas)
{
Console.Writeln(persona.Nombre + " " + persona.Apellido);
}
```

BOOM WARE TECHNOLOGIES

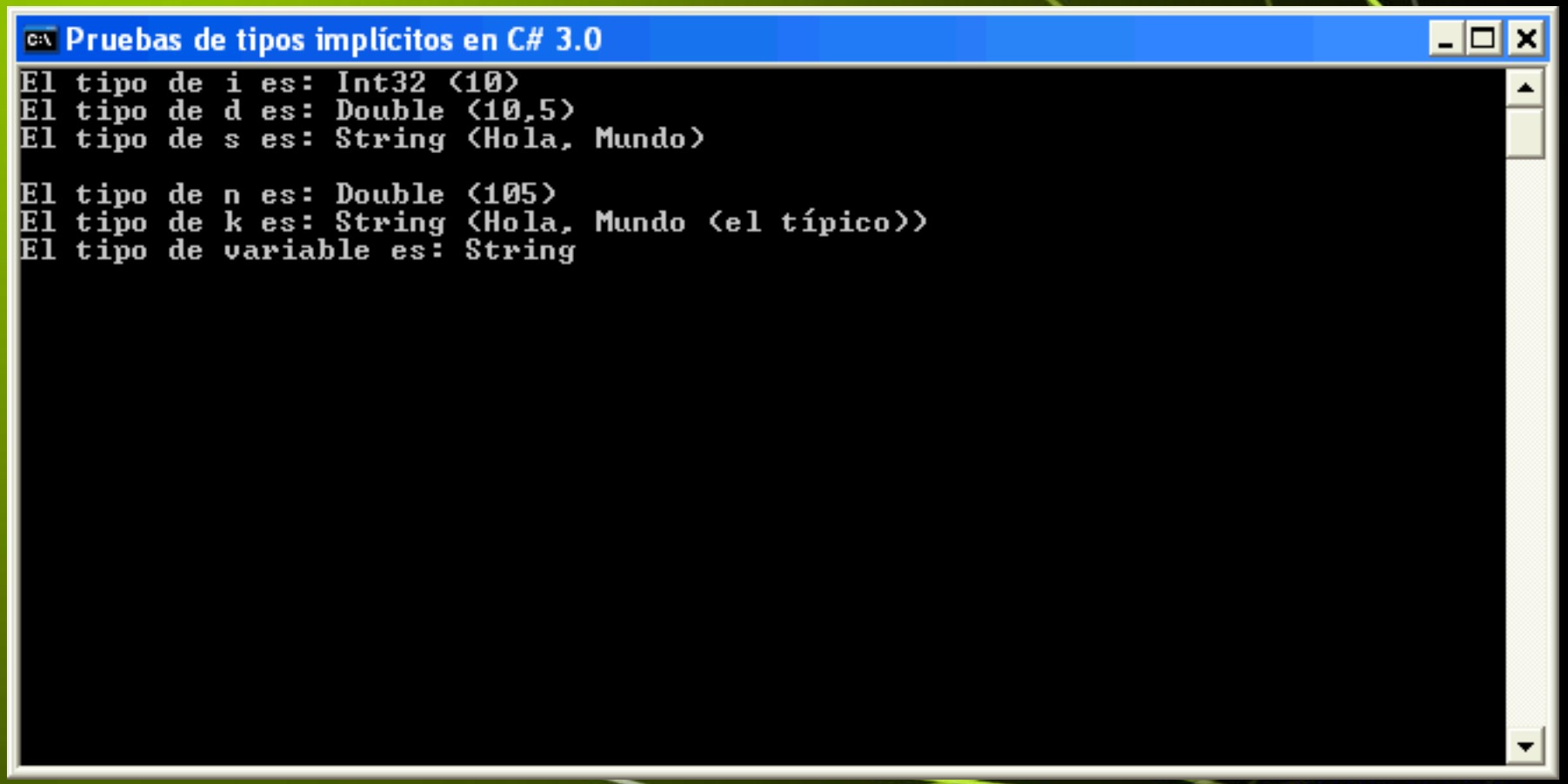
C# 3.x:
Tipos implícitos

- Tipos de datos que el compilador se encargará de averiguar de que tipo es en tiempo de compilación. Utilizamos **var**

```
var i = 10;
var d = 10.5;
var s = "Hola, Mundo";
Console.WriteLine("El tipo de i es: {0} ({1})", i.GetType().Name, i);
Console.WriteLine("El tipo de d es: {0} ({1})", d.GetType().Name, d);
Console.WriteLine("El tipo de s es: {0} ({1})", s.GetType().Name, s);
Console.WriteLine();
var n = i * d;
Console.WriteLine("El tipo de n es: {0} ({1})", n.GetType().Name, n)
var j = i * 3; var k = s + " (el típico)";
Console.WriteLine("El tipo de k es: {0} ({1})", k.GetType().Name, k)
var variable = “club .NET";
```

BOOM WARE TECHNOLOGIES

C# 3.x:
Tipos implícitos



C# 3.x: Expresiones Lambda

- Una extensión de los métodos anónimos, que ofrecen una sintaxis más concisa y funcional para expresarlos.
- La sintaxis de una expresión lambda consta de una lista de variables-parámetros, seguida del símbolo de implicación (aplicación de función) =>, que es seguido a su vez de la expresión o bloque de sentencias que implementa la funcionalidad deseada.
- El tipo **Func** representa a los delegados a funciones (con 0, 1, 2 ó 3 argumentos, respectivamente) que devuelven un valor de tipo T.

```
// en System.Query.dll
// espacio de nombres System.Query
public delegate T Func<T>();
public delegate T Func<A0, T>(A0 a0);
public delegate T Func<A0, A1, T>(A0 a0, A1 a1);
public delegate T Func<A0, A1, A2, T>(A0 a0, A1 a1, A2 a2);
```

BOOM WARE TECHNOLOGIES

C# 3.x:
Expresiones Lambda

```
Func<int, int> cuadrado = x => x * x;
// más ejemplos
static Func<double, double, double> hipotenusa =
(x, y) => Math.Sqrt(x * x + y * y);

static Func<int, int, bool> esDivisiblePor =
(int a, int b) => a % b == 0;

static Func<int, bool> esPrimo =
x => {
                for (int i = 2; i <= x / 2; i++)
                    if (esDivisiblePor(x, i))
                        return false;
                return true;
            };

static void Main(string[] args)
{
int n = 19;
        if (esPrimo(n)) Console.WriteLine(n + " es primo");
                Console.ReadLine();
}
```

BOOM WARE TECHNOLOGIES

C# 3.x:
LINQ, Consultas implicitas

- Nueva forma de consultar datos mejorando SQL
- Más cómodo para el programador, el entorno puede ofrecerle ayuda sobre los campos a consultar y los tipos a devolver
- SQL directamente en el código, variables de tipo definido en caliente, tipos anónimos

```
String consulta =“SELECT Nombre, Apellidos
FROM Amigos WHERE Sexo = ‘M’ AND EstadoCivil = ‘S’ AND
Edad <= 25”
```

¡¡¡Ahora tengo que ejecutarla con los métodos ADO!!!

AHORA dentro del código!!!

```
Var miAmigos= from f in db.Amigos
where f.Sexo == ‘M’ and f.EstadoCivil = ‘S’ and f.Edad <=
25
select new { f.Nombre, f.Apellidos };
```

BOOM WARE TECHNOLOGIES

Comentarios

```
string nombre = "Juan"; // Comentario de una sola línea
 /* Comentario con mas
   de una línea*/

/// <summary>
/// Documentación XML que puede ser consumida por otras
/// herramientas para mostrar ayuda dinámica o generar
/// documentación en varios formatos
/// </summary>
public class Matematica {
    /// <summary>
    /// Calcula la suma de dos números enteros
    /// </summary>
    /// <param name="x">El primer operando de la suma</param>
    /// <param name="y">El segundo operando de la suma</param>
    /// <returns> La suma entera de ambos operandos</returns>
    public int Sumar (int x, int y) {return x + y;}
}
```

Regiones de Código

- Estructura tu codigo. Más legilibilidad

```
#region
        public string PuntoYFinal {
                get {
                        return(“Gracias”);
                }
        }
   #endregion
```

BOOM WARE TECHNOLOGIES

Y aún hay más…

- Ficheros y Flujos de datos
- Acceso a bases de datos
- Componentes gráficos

BOOM WARE TECHNOLOGIES

Muchas gracias !!

BOOMWARE TECHNOLOGIES

Recursos

- http://msdn2.microsoft.com/es-es/library/kx37x362(VS.80).aspx
- http://recursos.dotnetclubs.com/granada
- http://msdn.microsoft.com/msdnmag/issues/07/06/CSharp30/default.aspx?loc=es
- http://www.boomware.pe/blog/

BOOM WARE TECHNOLOGIES