

# .NET FRAMEWORK y C#

Clase 2

Aspectos sintácticos

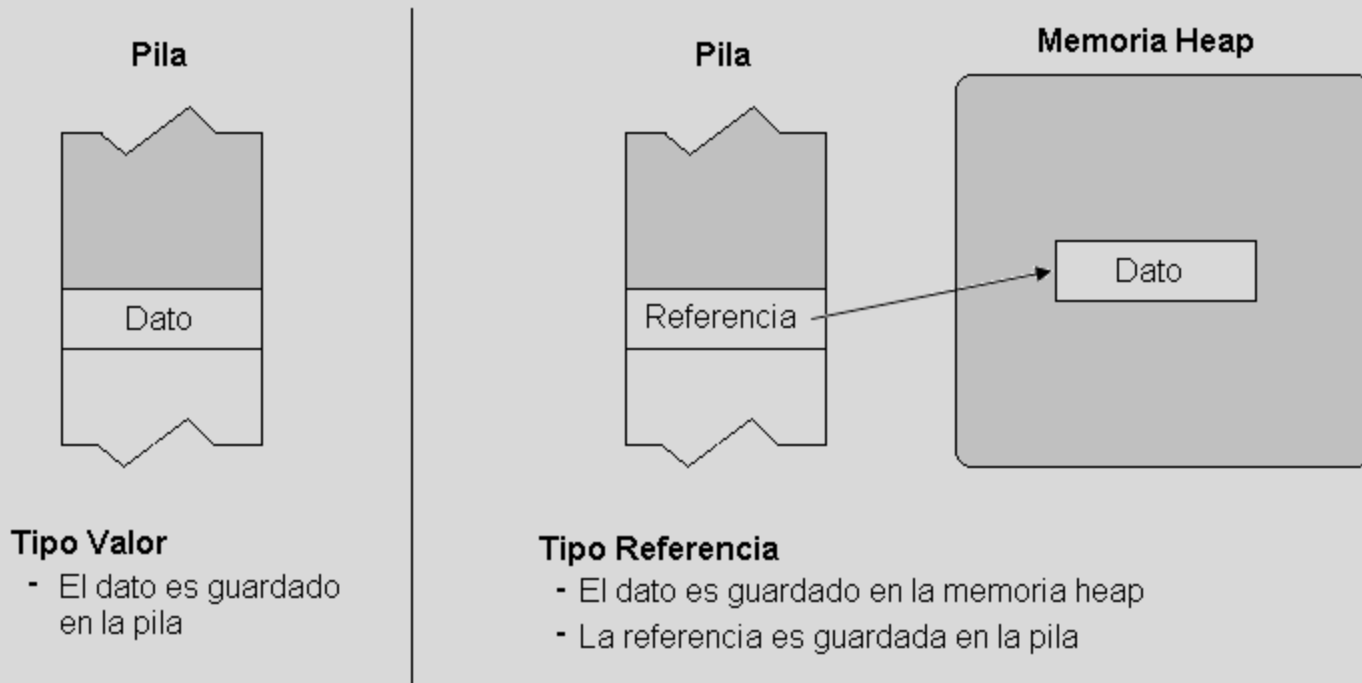
# Sistema de Tipos

- Todos los tipos de .Net Framework son de dos clases:
  - **Tipos Valor.**
  - **Tipos Referencia.**

# Sistema de Tipos

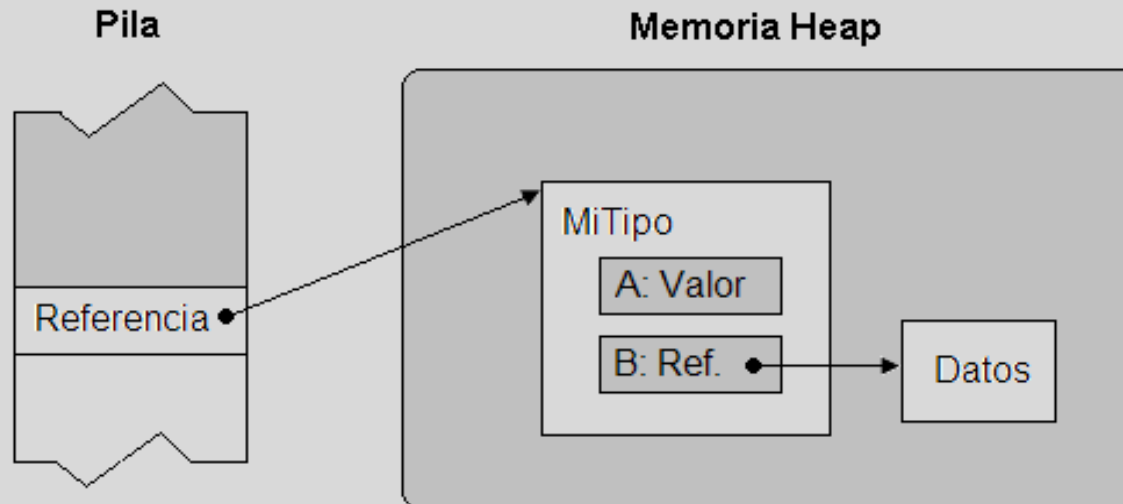
- Todos los tipos de .Net Framework son de dos clases:
  - **Tipos Valor.** Valor real del objeto. Al asignar una variable se realiza una copia de ese valor.
  - **Tipos Referencia.** Referencia que apunta al valor real del objeto. Al asignar a una variable se copia la referencia (no el valor real).

# Sistema de Tipos



# Sistema de Tipos

- La porción de datos de un tipo referencia siempre se guarda en la memoria heap
- Los datos de un tipo valor pueden guardarse en la pila o en la memoria heap dependiendo de las circunstancias



# Sistema de Tipos

- **Common Type System** en .NET Framework admite las cinco categorías de tipos siguientes:



# Sistema de Tipos

- **Tipos valor en C#**
  - Estructuras
    - Tipos numéricos
      - Tipos enteros (sbyte, byte, char, short, int ...)
      - Tipos de punto flotantes (float y double)
      - decimal
    - bool
    - System.DateTime
    - Estructuras definidas por el usuario
  - Enumeraciones

# Sistema de Tipos

- **Tipos referencia en C#**
  - Clases
  - Delegados
  - Interfaces
  - En particular **object** es un tipo referencia y constituye la raíz de la jerarquía de tipos (Sistema unificado de tipos).



# Sistema de Tipos

- Todos los tipos (valor y referencia) se heredan directa o indirectamente de **object**.
- Las variables de tipo **object** pueden recibir valores de cualquier tipo.
- Cuando una variable de un **tipo valor** se convierte en **object**, se dice que se le ha aplicado la conversión **boxing**.
- Cuando una variable de tipo **object** se convierte en un **tipo valor**, se dice que se le ha aplicado la conversión **unboxing**.

# Sistema de tipos -Boxing

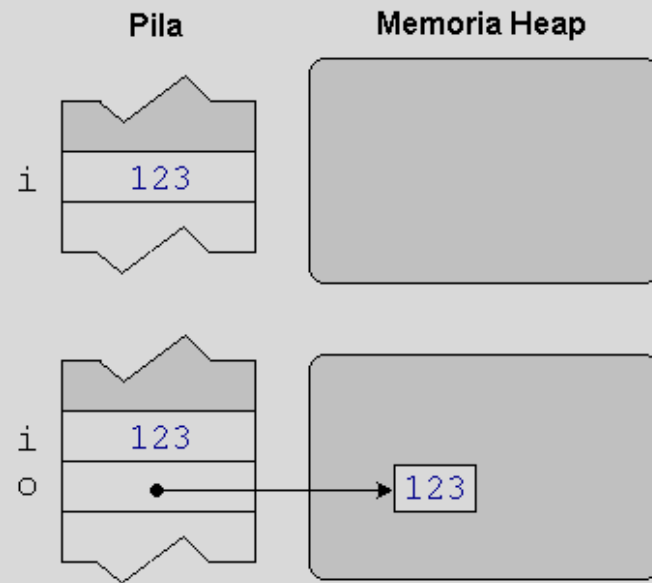
- **Boxing** es generalmente una conversión implícita de tipos valores al tipo **object** (o a cualquier tipo de interfaz implementado por este tipo valor).
- Al aplicar la conversión **boxing** se asigna una instancia de objeto en la memoria heap y se copia el valor en el nuevo objeto.

# Sistema de tipos - Boxing

- Se crea una referencia de objeto en la pila que hace referencia a un valor del tipo **int** en la memoria heap. Este valor es una copia del valor asignado a la variable *i*.

```
int i=123;
```

```
object o=i; //boxing implícito
```



# Sistema de tipos - Unboxing

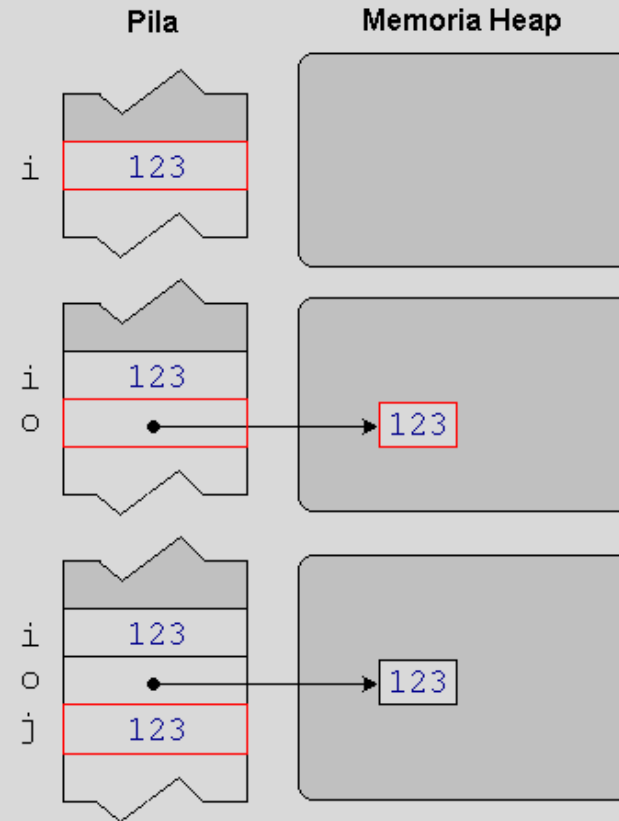
- **Unboxing** es una conversión explícita del tipo **object** a un tipo valor (o de un tipo interfaz a un tipo valor que implementa esa interfaz).
- Una operación **unboxing** consiste en:
  - Comprobar la instancia de objeto para asegurar que se trata de un valor convertido mediante **boxing** del tipo de valor dado
  - Copiar el valor de la instancia en la variable de tipo valor

# Sistema de tipos - Unboxing

```
int i=123;
```

```
object o=i; //boxing
```

```
int j = (int)o; //unboxing
```



# Sistema de Tipos

- Las variables de tipos referencia o bien contienen la referencia al objeto en la memoria heap, o bien poseen un valor especial nulo (palabra clave **null**)
- Las variables de tipos valor siempre contienen un valor válido para su tipo y por lo tanto no admiten el valor **null** (a excepción de los “**nullable value type**”).
- Ejemplo:

```
int i; object obj;
```

```
obj = null;
```

```
i = null;
```

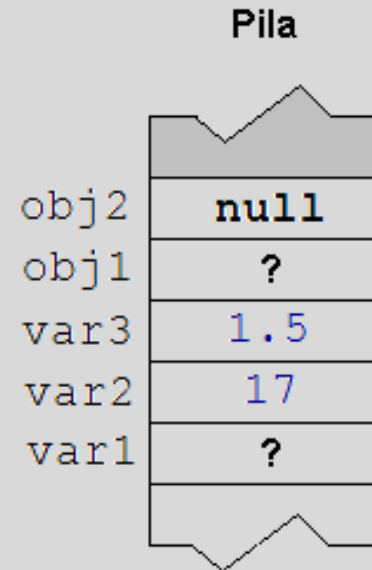
Válido

Inválido (error  
de compilación)

# Sistema de Tipos

- Declaración de **variables locales** - Pila resultante

```
int var1;  
int var2 = 17;  
float var3 = 1.5;  
object obj1;  
object obj2 = null;
```



- Las **variables locales** sin inicializar poseen un valor indefinido considerándose erróneo todo acceso de lectura que se haga a las mismas mientras no se les escriba algún valor.

# Sistema de Tipos – Arreglos I

- Un arreglo es un conjunto de **elementos homogéneos** (mismo tipo) accedidos a través de **uno o más índices**.
- Es un tipo referencia.
- Los arreglos pueden tener varias dimensiones (vector, matriz, tensor, etc.) el número de dimensiones se denomina **rank**
- El número total de elementos de un arreglo (en todas sus dimensiones) se llama longitud del arreglo (**Length**)



# Sistema de Tipos – Arreglos I

Arreglos de una dimensión (vectores). Ejemplo

```
int[] vector1;
```

Declara un vector

```
vector1 = new int[200];
```

Instancia un vector de 200 elementos de tipo entero (se aloca en la heap)

```
int[] vector2 = new int[100];
```

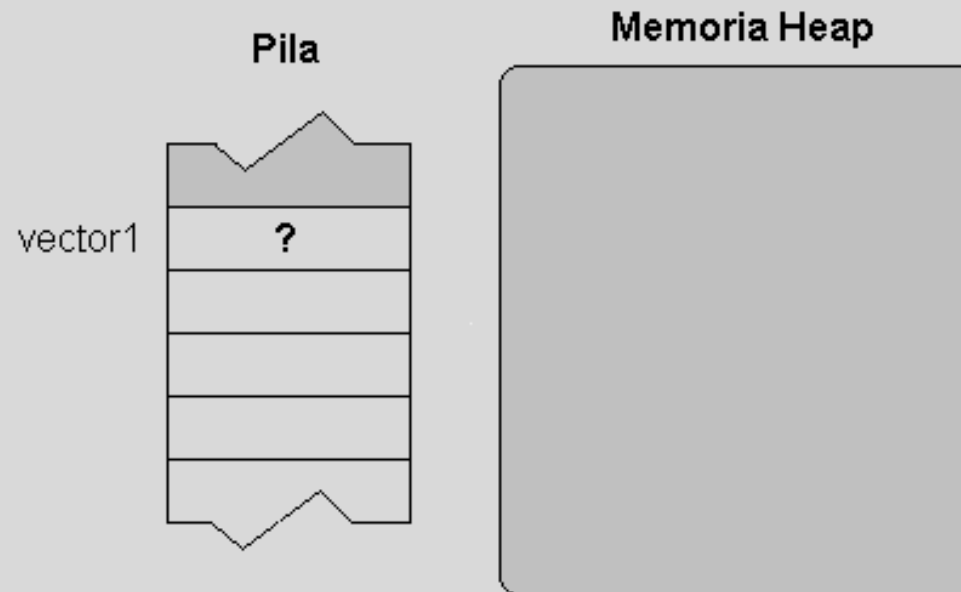
Declara e Instancia un vector de 100 elementos de tipo entero

```
int[] vector3 = new int[] {5,1,4,0};
```

Declara, instancia e inicializa un vector de 4 elementos de tipo entero

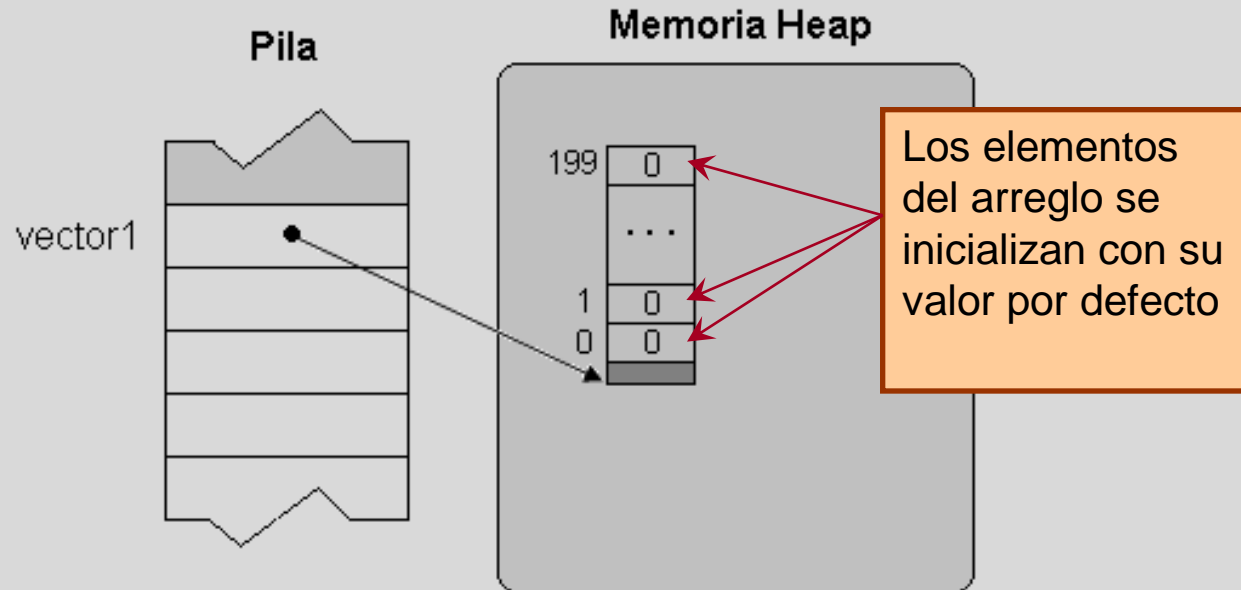
# Sistema de Tipos – Arreglos I

```
int[] vector1;
```



# Sistema de Tipos – Arreglos I

```
int[] vector1;  
vector1 = new int[200];
```



# Sistema de Tipos – Arreglos I

- Cuando se instancia un arreglo, el tamaño puede especificarse por medio de una variable

```
int tam=5;
```

```
char[] vocal = new char[tam];
```

- Acceso a los elementos con operador [ ]

```
vocal[1] = 'E';
```

- El primer elemento ocupa la posición 0

```
vocal[0] = 'A';
```

- Último elemento:

```
vocal[vocal.Length - 1] = 'U';
```

# Sistema de Tipos – Arreglos I

- Estructura de control **foreach**

- Se utiliza con **arreglos** y otras **colecciones**
- Recorre toda la colección.
- Reemplazo del **for** en forma compacta
- Sintaxis

```
foreach (<tipoElem> <elem> in <colección>)  
    <instrucción o bloque de instruc.>
```

- Restricción: La variable de iteración <elem> no puede ser asignada en el cuerpo del `foreach`

- Ejemplo:

```
string[] vector = new string[] { "curso", "de", ".Net" };  
foreach (int elemento in vector)  
    Console.WriteLine(elemento);
```

# Sistema de Tipos – Diferencias entre tipos valor y referencia

```
char c1 = 'A';
```

```
char c2 = 'A';
```

```
Console.Write("Comparando c1 con c2: ");
```

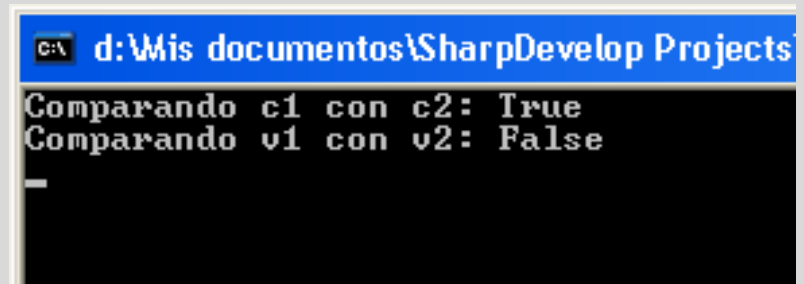
```
Console.WriteLine(c1==c2);
```

```
char[] v1 = new char[] {'A'}; //vector de un elemento
```

```
char[] v2 = new char[] {'A'}; //vector de un elemento
```

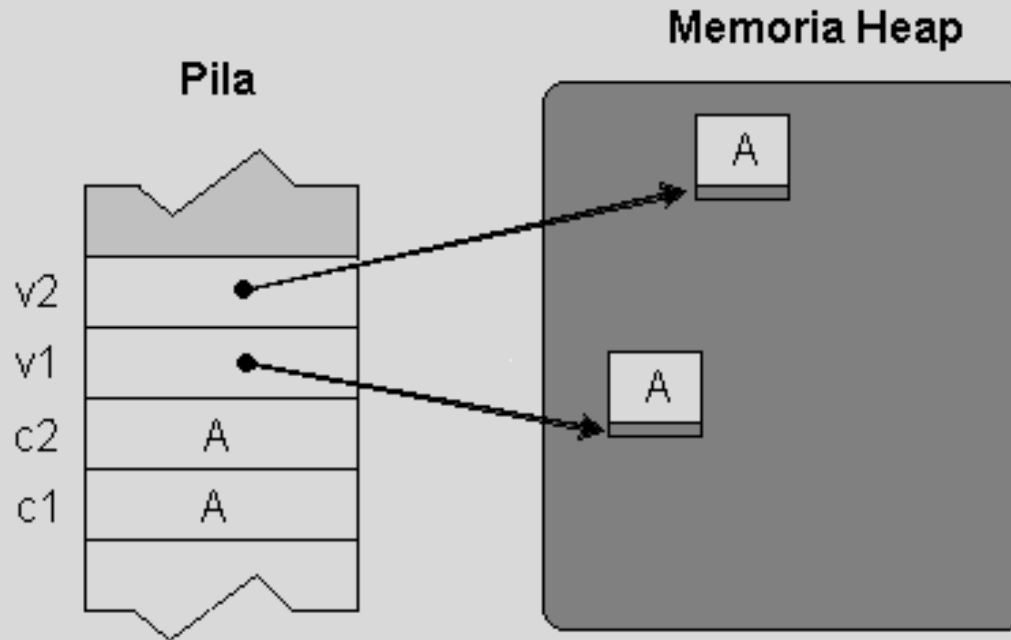
```
Console.Write("Comparando v1 con v2: ");
```

```
Console.WriteLine(v1==v2);
```



```
C:\ d:Wis documentos\SharpDevelop Projects
Comparando c1 con c2: True
Comparando v1 con v2: False
_
```

# Sistema de Tipos – Diferencias entre tipos valor y referencia

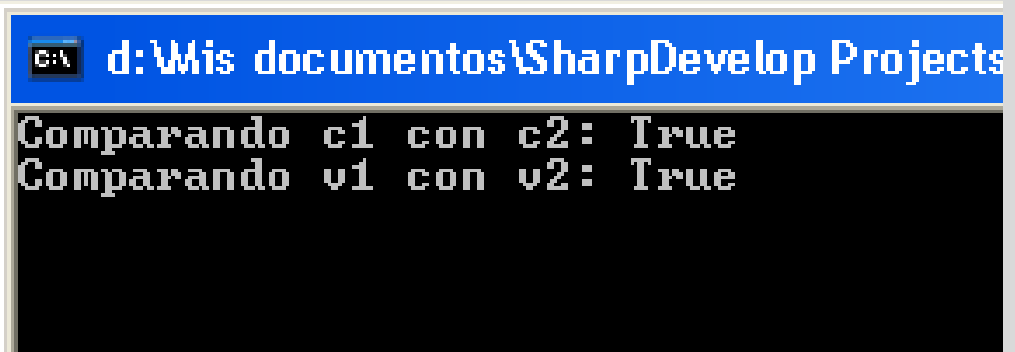


La comparación por igualdad de v1 y v2 resulta falsa puesto que, por tratarse de tipos referencia, no se compara el contenido sino la propia referencia

# Sistema de Tipos – Diferencias entre tipos valor y referencia

```
char c1 = 'A';  
char c2 = c1; //copia el valor 'A'  
Console.Write("Comparando c1 con c2: ");  
Console.WriteLine(c1==c2);
```

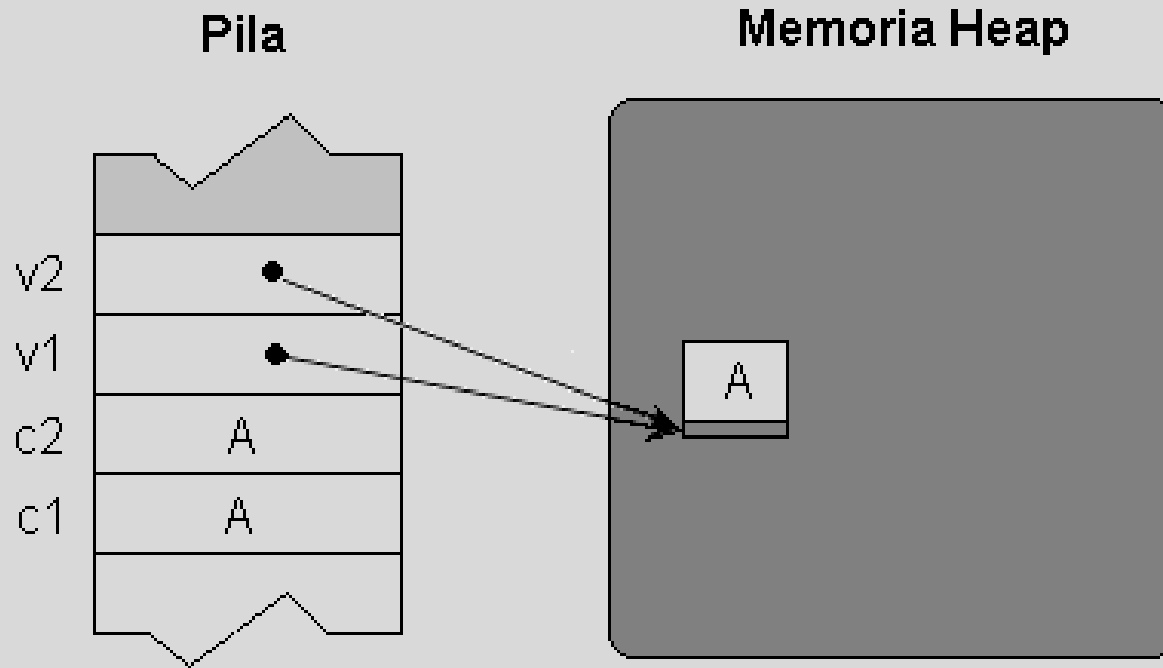
```
char[] v1 = new char[] {'A'};  
char[] v2 = v1; //copia la referencia  
Console.Write("Comparando v1 con v2: ");  
Console.WriteLine(v1==v2);
```



```
d:\Mis documentos\SharpDevelop Projects  
Comparando c1 con c2: True  
Comparando v1 con v2: True
```



# Sistema de Tipos – Diferencias entre tipos valor y referencia

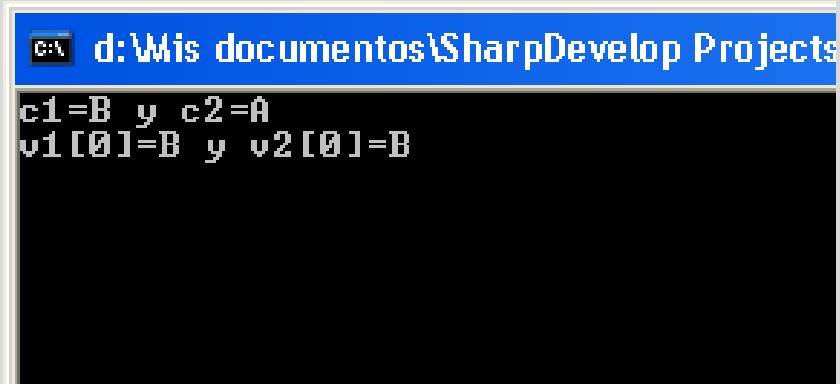


La comparación por igualdad de **v1** y **v2** resulta verdadera puesto que ambas variables poseen la misma referencia (apuntan al mismo objeto)

# Sistema de Tipos – Diferencias entre tipos valor y referencia

```
char c1 = 'A';  
char c2 = c1; //copia el valor 'A'  
c1='B';  
Console.WriteLine("c1=" + c1 + " y c2=" + c2);
```

```
char[] v1 = new char[] {'A'};  
char[] v2 = v1; //copia la referencia  
v1[0]='B';  
Console.WriteLine("v1[0]=" + v1[0] + " y v2[0]=" +  
v2[0]);
```



```
C:\ d:\Mis documentos\SharpDevelop Projects  
c1=B y c2=A  
v1[0]=B y v2[0]=B
```

# Sistema de Tipos – La clase String

- Secuencia de caracteres

```
string st1 = "es un string";  
string st2 = "";  
string st3 = null;
```

- Es un tipo referencia

- Por lo tanto acepta el valor **null**

- Sin embargo la comparación no es con la dirección de memoria

- Se ha redefinido el operador == para realizar una **comparación lexicográfica**
    - Tiene en cuenta mayúsculas y minúsculas

# Sistema de Tipos – La clase String

- Los **string** son de sólo lectura (no pueden modificarse caracteres individuales)
- El acceso a los elementos [ ]
- Primer elemento: índice cero

```
string st = "Hola";
```

```
char c = st[0];
```

```
st[0]='A';
```

Error de compilación: Los string son de sólo lectura

```
string cad = "Hola";
```

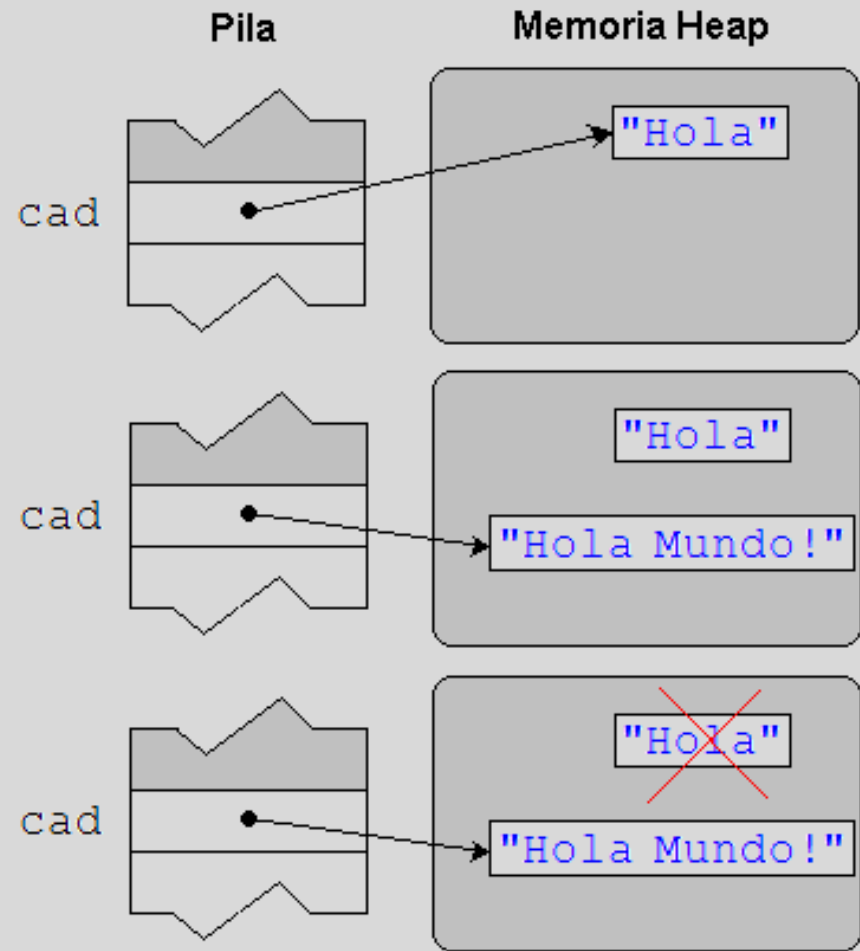
```
cad = cad + " Mundo!";
```

Correcto, se está creando un nuevo string

# Sistema de Tipos – La clase String

```
string cad = "Hola";
```

```
cad = cad + " Mundo!";
```

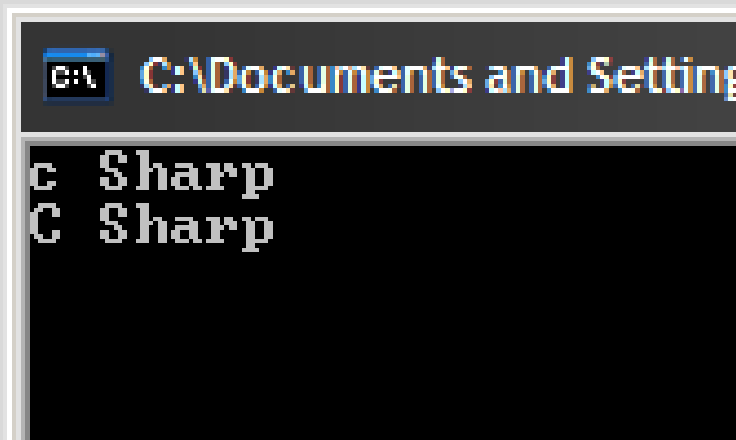


# Sistema de Tipos – La clase StringBuilder

- String de lectura/escritura
- Definida en el espacio de nombre System.Text
- Métodos adicionales
  - Append
  - Insert
  - Remove
  - Replace
  - etc.

# Sistema de Tipos – La clase StringBuilder

```
using System;
using System.Text;
class Ejemplo
{
    static void Main(String[] args)
    {
        StringBuilder stb;
        stb=new StringBuilder("c Sharp");
        Console.WriteLine(stb);
        stb[0] = 'C';
        Console.WriteLine(stb);
    }
}
```



```
C:\Documents and Settings
c Sharp
C Sharp
```

# Sistema de Tipos – Conversión de tipos

- En C#, puede realizar los siguientes tipos de conversiones:
  - **Conversiones implícitas** (Vistas en la teoría 1)
  - **Conversiones explícitas** (Vistas en la teoría 1)  
Requieren un operador de conversión.
  - **Conversiones definidas por el usuario**
  - **Conversiones con clases auxiliares**: para realizar conversiones entre tipos no compatibles, como la clase **System.Convert**, los métodos **Parse** de los tipos numéricos y el método **ToString** redefinible en todos los tipos



# Sistema de Tipos – Conversión de tipos

- **Conversiones explícitas.**

- Se utilizan los operadores **()** y **as**.
- Cuando una conversión de tipo no puede llevarse a cabo el operador **()** provoca una excepción (error en tiempo de ejecución)
- Cuando una conversión de tipo no puede llevarse a cabo el operador **as** devuelve el valor **null** (no provoca una excepción)
- Por lo tanto **as** se utiliza sólo para tipos **referencia** o tipos **nullables**

# Sistema de Tipos – Conversión de tipos

- Conversiones explícitas. Ejemplo.

```
object obj = "casa";
```

```
string st = (string) obj;
```

Conversion Ok

```
obj = 12;
```

```
st = obj as string;
```

st recibe el valor null porque no se puede convertir un entero en un string

```
st = (string) obj;
```

Provoca error en tiempo de ejecución (InvalidCastException)

# Sistema de Tipos – Conversión de tipos

- Conversiones con clases auxiliares. Ejemplo.

```
int i = int.Parse("321");
```

```
double d = int.Parse("321.34");
```

Error en ejecución  
(FormatException)

```
d = double.Parse("321.45");
```

```
string st = i.ToString();
```

```
st=27.654.ToString();
```

```
DateTime fecha = DateTime.Parse("23/3/2012");
```

```
i=(int) true;
```

Error de compilación

```
i=Convert.ToInt32(true);
```

# Tipos Enumerativos

- **Definición de enumeraciones**

```
enum Tamaño{  
    chico, mediano, grande  
}
```

- **Uso de enumeraciones**

```
Tamaño t;  
t=Tamaño.grande;  
t=(Tamaño)0; //ahora t vale Tamaño.chico
```

# Parámetros a Main

- `static void Main()`
- `static int Main()`
- `static int Main(string[] args)`
- `static void Main(string[] args)`

# Parámetros a Main. Ejemplo

```
using System;
class Ejemplo
{
    static void Main(String[] args)
    {
        foreach(string st in args)
            Console.WriteLine(st);
        Console.ReadKey();
    }
}
```

# Parámetros a Main

- El IDE **SharpDevelop** permite establecer los parámetros que se envían por la línea de comando desde el propio entorno
- Utilizar menú *Proyecto / Opciones de proyecto*, pestaña *Depurar* cuadro de texto *Argumentos de línea de comandos*

# Métodos

- **Definición de métodos**

```
<tipoRetorno> <nombreMétodo> (<parámetros>)  
{  
    <cuerpo>  
}
```

–Si el método no devuelve ningún valor se utiliza **void** como <tipoRetorno>.

- **Llamada a métodos**

```
<objeto>.<nombreMétodo> (<valoresParámetros>)  
<tipo>.<nombreMétodo> (<valoresParámetros>)  
<nombreMétodo> (<valoresParámetros>)
```



# Métodos - Parámetros

- **Parámetros de entrada (por valor)**
- Recibe una copia del valor pasado como parámetro

```
int suma( int par1, int par2)
{
    return par1+par2;
}
```

# Métodos - Parámetros

- **Parámetros de salida**

- Se debe asignar dentro del cuerpo del método antes de cualquier lectura.
- Es posible pasar parámetros de salida que sean variables no inicializadas.

# Métodos - Parámetros

- **Parámetros de salida**

Observar que un método estático llama a otro método estático

```
static void Main() {  
    int r;  
    suma(10,20,out r);  
    Console.WriteLine(r);  
}
```

```
static void suma(int n1,int n2,out int result) {  
    result = n1+n2;  
}
```

Si se omite static no podría ser llamado desde el método Main

# Métodos - Parámetros

- **Parámetros por referencia**

- Similar a un parámetro de salida sólo que no es obligatorio escribirlo dentro del método al que pertenece
- Es obligatorio pasarle una variable inicializada ya que no se garantiza su inicialización en el método.

# Métodos - Parámetros

- **Parámetros por referencia**

```
static void Main() {  
    int r=0;  
    suma(10,20,ref r) ;  
    Console.WriteLine(r) ;  
}
```

```
static void suma(int n1,int n2,ref int result) {  
    result = n1+n2;  
}
```

# Repaso práctica 1

```
Console.WriteLine("c:\\documento.txt");
```

Secuencia de escape no reconocida.  
Solución: "c:\\\\documento.txt"

```
Console.WriteLine("100".Length);
```

Imprime la longitud del string "100", es decir el número 3.

```
Console.WriteLine(st=Console.ReadLine());
```

Instrucción perfectamente válida. Imprime lo que se lee por teclado y además lo asigna a la variable st

# Repaso práctica 1

**double** d=1/3; ←

Cuidado! Asigna cero a la variable d porque los literales 1 y 3 se consideran enteros y la división entre enteros retorna un entero

Console.**WriteLine**(3+3); ←

Imprime 6

← Console.**WriteLine**("3"+3);

Imprime 33 porque el operador + entre un string y un número convierte el número a string y lo concatena con el otro operando

← **if** ((b != 0) & (a/b > 5)) Console.**WriteLine**(a/b);

Si b es cero da error en tiempo de ejecución.

Solución: utilizar el operador &&