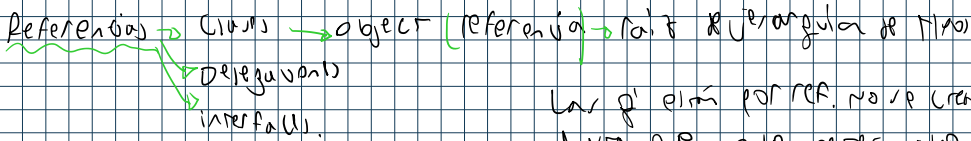
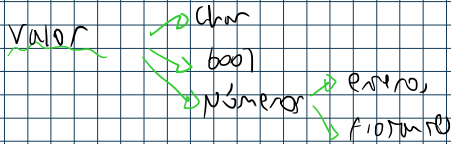


tipo de valor y referencia
clases y primitivos son tipos
Números son de valor, referencias son punteros a ellas.
string es de referencia.



Las q' están por ref. no se crean en la heap
hasta que no le metes algo.

- conversiones de tipos
- Simple Casts (int := bool) → Cuando no hay riesgo
- Explicit Casts
- Cast to nullable → (int?.Parse(string))
→ string convert
→ parsing → bool
- Operadores por el valor los tipos no cambian

```
byte b = 10;
double x = 12.25;
int i = b;
double y = 1;
short j = 1;
i = x;
```

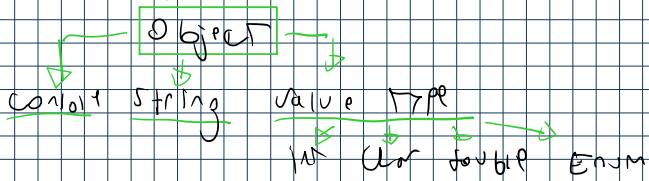
→ conversión implícita byte a int
→ conversión implícita int a double
error: No se puede hacer conv. implícita (int a short)
→ siempre int

ej: on { short s = (short) 1;
s = (int) 1;
Problema de flujo.
Si el número es grande de lo q' entra.

using i = short(true)

Objetos peticion de memoria

esto habra de int a float, implícitamente pero en otros muy pocas veces pero
ej: int i := 100 000 000 000
bool j = i → q' entra en 100 000 000 000.



Any se puede meter todo x jerarquía de tipos variables de tipo object
almacenar valores de cualquier tipo.

ej: int i = 123;
object o = i;
Lo de arriba puede ser como se
abaja, no al revés

problema unificado de tipos

boxing - unboxing → asignar variables de tipo valor a tipo referencia y al revés

object o;
int i = 123;
o = i; → cuando asigno tipo valor a ref se hace boxing

int j = (int) o;
→ unboxing es lo que se hace en lugar de plus
para seguir con referencia del objeto que usas

if i;
 int j = [int]

II

II ref. a valor but unboxing

una referencia en heap en lugar de pila
(pila se genera con referencia del objeto que es una en heap)
↓
valor q' estaba en heap queda en pila.

Null

para q' una VM random admita Null, tengo q' declararla con un?

Ej:
 int? j = Null
 string? pipi = Null

string? s1 = null;
 string? s2 = "Hola";

Pila Heap
 s1 0x000000
 s2 0x12345 → "Hola"

Has value - value

int? i1 = 10;
 int? i2 = Null;

Pila
 s1 { true → has value
 10 → value
 s2 { false → has value
 0 → value

• i = s?? -1;

Si no es null asigna el valor, sino asigna -1

• s = j. Hasvalue? j.value: -1 → s = -1
 j = j.value

• s = j != Null? (int) j: -1
 j != Null? true s = j, sino -1

Operator 'is'

• Convierte x tipo de valor.

Objeto 0 = 1

CompareWithInt(0, is, int) → True

Operator 'is'

• Se dan las cosas que aceptan null
 conversiones explicitas.

Cuando nos piden hacer la conversión, en lugar de dar error de Null

Es el T donde se convierte en valor y T es nombre de tipo

E V T? (E) (T): Null

Objeto obj = "Hola";

string? st = (string) obj;

obj = 12;

→ tengo q' asegurarme no pq podría meter algo q' no es string en string.

st = obj es string → va a dar Null (pq no pudo pasar int a string)

st = (string) obj; → tira error de ejecución

Variables locales

• C# no te permite leer una VM que tiene basura.

Mo Compila

Array

```
int[] vector1;
```

```
vector1 = new vector1;
```

```
int[] vector1 = new int[3] {5, 1, 3}
    ||      ||      = new int[3] {5, 1, 3}
    ||      ||      = {5, 1, 3}
```

Preparar para asignar tamaño al arreglo

`int[int.length - 1]` → para acceder a ult. dato.

`.length` es tamaño, no cuánto está lleno.

foreach

- estructura de control. Recorre toda la estructura.
- `foreach (int num in vector)`
bueno para conservar la q' hay.

String

- El x referencia pero no se comparan punteros al haber == sino q' se comparan lo dentro
- No se pueden modificar strings

→ `st = st + "nuevo"` → crea una nueva string, no una copia.
va haciendo copias.

String builder

- como una string pero con + opciones

mutable

```
string builder sb = new StringBuilder("C sharp")
```

```
Console.WriteLine(sb);
```

Tipos primitivos

enum tamaño { chico, mediano, grande }

tamaño t;

t = tamaño.grande

t = (tamaño) 0 ; // vale tamaño chico ahora

Métodos

biogol) con nombre de código ejecutable

se invoca

```
int sum(a, int b)
```

```
int res = a + b
```

```
return res;
```

Manejo de salida

→ reasignar la a valor de método

→ se pueden pasar par. de salida no inicializados.

```
void sum(int a, b, out int res)
```

```
res = a + b
```

2. referencia → `suma(10, 20, ref r)`

↳ `void suma(int a, b; ref int r)`

param de entrada → se pasa x ref pero no se puede modificar
↳ + presente en algunos casos
↳ (no usar xp)

formas → Me da más flexibilidad con la consola de comandos

Método en forma de instrucción

para métodos con retorno.

`int suma(int a, int b)` \Rightarrow `suma(int a, int b) => Console.WriteLine(a+b);`
 `return a+b;`