Tecniche di Programmazione avanzata

Corso di Laurea Specialistica in Ingegneria Telematica
Università Kore – Enna – A.A. 2008-2009

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

Il linguaggio C#
Introduzione e tipi di dato

Origini di C#

- C# was developed by Anders Hejlsberg and Scott Wiltamuth.
- Hejlsberg is also known for creating Turbo Pascal, a popular language for PC programming, and for leading the team that designed Borland Delphi, one of the first successful IDE for client/server programming.
- The goal of C# is to provide a simple, safe, modern, object-oriented, **Internet-centric**, high-performance language for .NET development
- You learn C# **specifically to create .NET applications**; pretending otherwise would miss the point of the language.

Origini di C#

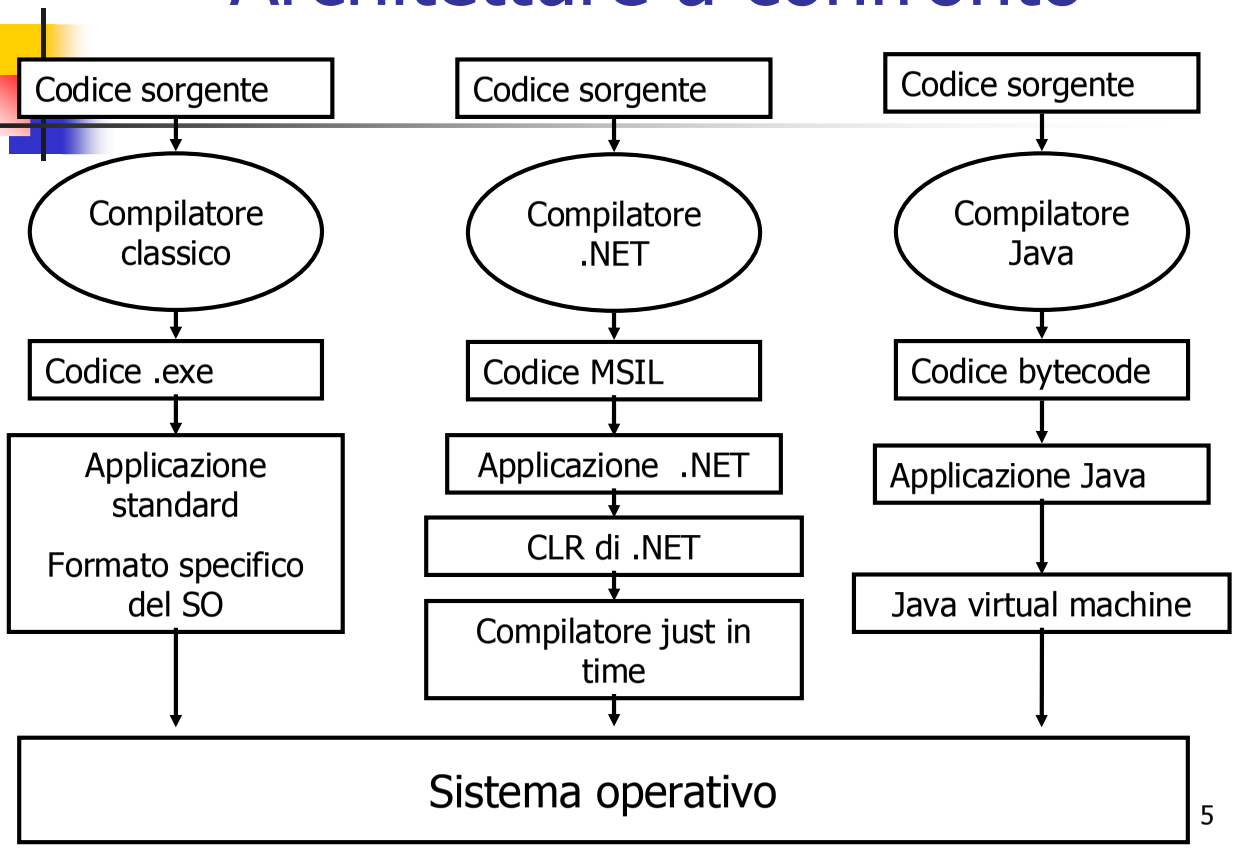
- Inizialmente, Microsoft supportava Java
- Poi, ci fu una lite giudiziaria con Sun, poiché Microsoft voleva aggiungere classi e features specifiche per Windows (motivi di performance)
- Microsoft perse la causa, indi abbandonò Java in Internet Explorer 6 la JVM Microsoft non è più installata per default, e comunque supporta solo fino a Java 1.1.4
- C# è il linguaggio "simil-Java" che Microsoft ha introdotto dopo tale evento, e .NET è l'infrastruttura che fa le veci della Java Platform

C# - Architettura

- L'architettura è "simil-Java", con alcune differenze

Java platform	**C# e .NET**
Compilatore Java – produce bytecode (**.class**) portabile	Compilatore C# – produce un formato (MSIL) portabile per .NET
Interprete Java (platform dependent) - *in realtà opera come compilatore JIT*	Compilatore JIT (produce **.exe** per piattaforma CLR – richiede .NET Framework sul sistema)
nessuno strato embedded nel sistema operativo	Common Language Runtime (CLR): quando si richiede l'esecuzione di un **.exe** di .NET, attiva il compilatore JIT che traduce MSIL in codice nativo e poi lo esegue

Architetture a confronto



Convenzioni e Strumenti

- Anche il processo di sviluppo è "simil-Java", con alcune importanti differenze:

Java platform	**C# e .NET**
Ogni classe pubblica deve stare in un file omonimo **.java**	*nessun vincolo esplicito*
Il compilatore **javac** produce il file **.class** corrispondente	Il compilatore **csc** produce il file MSIL **.exe** corrispondente
Prerequisito per l'esecuzione: **jre** (Java 2 Runtime Environment) installato sul sistema	**Prerequisito per l'esecuzione:** **.NET Framework (Redistributable o SDK)** installato sul sistema
L'interprete **java** interpreta il file **.class** (o **.jar**) sopra generato	Lo strato Common Language Runtime (CLR) attiva il compilatore JIT che traduce il "finto **exe**" MSIL in codice nativo, e lo esegue al volo

Caratteristiche di C#

1. E' orientato agli oggetti
2. Component-orientation
3. Software robusto
4. Mantenimento degli investimenti

Object oriented

- Ogni cosa è un oggetto
  - I tipi primitivi di Java non sono oggetti (sono speciali) e non interagiscono con gli oggetti
  - I tipi primitivi di Smalltalk, Lisp sono oggetti ma hanno qualche problema di performance
- C# tratta i tipi primitivi come oggetti senza problemi di performance
- Migliore estendibilità e riutilizzabilità del codice
  - Nuovi tipi primitivi: Decimal, SQL…
  - Classi come Collections

Component oriented

- C# è il primo linguaggio della "famiglia C" Component-Oriented
- Cos'è un componente?
  - Un modulo indipendente (reuse e deployment)
  - Minor Granularita degli oggetti (gli oggetti sono costrutti linguistici)
  - Possono includere più classi
  - Spesso language-independent
- Sviluppo e uso del componente non necessitano di nessuna interazione

Component oriented

- I componenti sono supportati dal linguaggio
  - Proprietà, metodi ed eventi
  - Attributi design time e run time
  - Documentazione integrata XML
- Tutto in un file
  - Nessun file di header, IDL, etc.
  - Dichiarazione ed implementazione coincidono
  - Può essere utilizzato in pagine ASP.NET (autosufficienti)

Software robusto e facile manutenzione del software

- Garbage collection
  - Fine dei memory leaks (consumo non voluto di memoria dovuto alla mancata deallocazione dalla memoria di variabili/dati non più utilizzati da parte dei processi) e dei puntatori pendenti
- Eccezioni
- Type safety
  - Non ci sono variabili non inizializzate
  - limiti alle operazioni di cast (no unsafe cast)
- Versioning
  - Supporto del versioning

Caratteristiche di C#

- Supporto per interfacce e ereditarietà singola
- Gli oggetti nell'heap sono gestiti dal garbage collected
- Permette l'allocazione nello stack di quasi-objects
- Espone metadata come attributi
- Introduce tipi delegate e eventi
- I parametri dei metodi devono essere labellati in/out/ref
- Le classi possono avere un distruttore come in C++
- Il rilascio esplicito delle risorse è gestito attraverso le interfacce
- Miglior controllo dei nomi
- Consente l'overloading degli operatori
- Gli arrays possono essere multidimensionali
- Sono consentiti alias per i tipi
- E' supportata la documentazione XML
- E' consentito l'accesso alla memoria attraverso i puntatori (unsafe mode)
- E' supportata una preprocessor-like syntax
- E' possibile accedere a funzioni esterne memorizzate come dll
- Interoperabilità (con XML, SOAP, COM, DLL e qualsiasi linguaggio di .NET)
- Milioni di linee di C# in .NET, quindi bassa curva di apprendimento

C# and Java

- Java è essenzialmente un subset di C# con quattro importanti differenze:
  - Throws clause on methods
  - Inner classes
  - Thread synchronization
  - Class loading

C# and Java (cont.)

- Inner classes in C# sono gestite come namespace, lo stato deve essere passato esplicitamente come argomento
- I Threads sono resi disponibili attraverso la classe Monitor
- Il modello di Class loading ha la stessa espressività
- C# e CLR sono state progettate insieme ma sono due elementi separati quindi le loro funzionalita sono meglio separabili che tra Java/JVM

C# and Java (cont.)

- C# permette di definire più classi all'interno dello stesso file: l'unità di sviluppo è l'Assembly
- I tipi sono organizzati in namespaces che sono simili ai packages Java
- Namespaces sono definiti come blocchi all'inizio del file
- Un singolo file può contenere definizioni che riguardano differenti namespaces

Il Linguaggio C#

Sintatticamente, è un "quasi-clone" di Java…

- …con aggiunte di C++
  - overloading degli operatori
  - operatori di conversione di tipo
- …di alcuni nuovi concetti
  - letterali "verbatim"
  - proprietà, indicizzatori
- … e altri più "discutibili"
  - passaggio per riferimento (anche per tipi primitivi)
  - codice "non gestito", puntatori
- Le maggiori differenze sintattiche riguardano la scomparsa delle keywords extends e implements non presenti in C#
  - package si trasforma in namespace
  - C# adotta le convenzioni Pascal (iniziale maiuscola)

Il Linguaggio C#: Convenzioni

- In Java:
  - i package hanno nomi con iniziale minuscola
  - le classi hanno nomi con iniziale *maiuscola*
  - i metodi hanno nomi con iniziale minuscola
  - le variabili hanno nomi con iniziale minuscola
- In C#:
  - i namespace hanno nomi con iniziale *maiuscola*
  - le classi hanno nomi con iniziale *maiuscola*
  - i metodi hanno nomi con iniziale *maiuscola*
  - le variabili hanno nomi con iniziale *maiuscola*

ESEMPIO

In Java:

```
public class Esempio {
        public static void main(String args[]){
                System.out.println("Hello World");
}
```

main / print / println Main / Write / WriteLine argomenti del main obbligatori argomenti del main facoltativi

In C#:

```
class Esempio {
        static void Main(){
                System.Console.WriteLine("Hello World");
}
```

classe e main dichiarati public classe e main senza qualifiche

Caratteristiche di C#

Nuove (vs. Java)
- Call-by-reference parameters
- Stack-allocated objects (structs)
- Block matrices
- Enumerations
- Uniform type system
- goto statement
- Attributes
- System-level programming
- Versioning

Variazioni sintattiche
- Component-based programming
  - Properties
  - Events
- Delegates
- Indexers
- foreach loop
- Boxing / unboxing
- ...

Hello World

File Hello.cs

```
using System;
class Hello {
 static void Main() {
 Console.WriteLine("Hello World");
 }
}
```

Importa il namespace *System*
La funzione principale deve essere chiamata *Main*
Stampa su console
Il nome del file e quello della classe possono essere diversi

Compilazione (dalla consolle; produce Hello.exe)
csc Hello.cs
Execution
Hello

- every C# program must have a Main( ) method
- It is technically possible to have multiple Main( ) methods in C#; in that case you use the /main command-line switch to tell C# which class contains the Main( ) method that should serve as the entry point to the program.

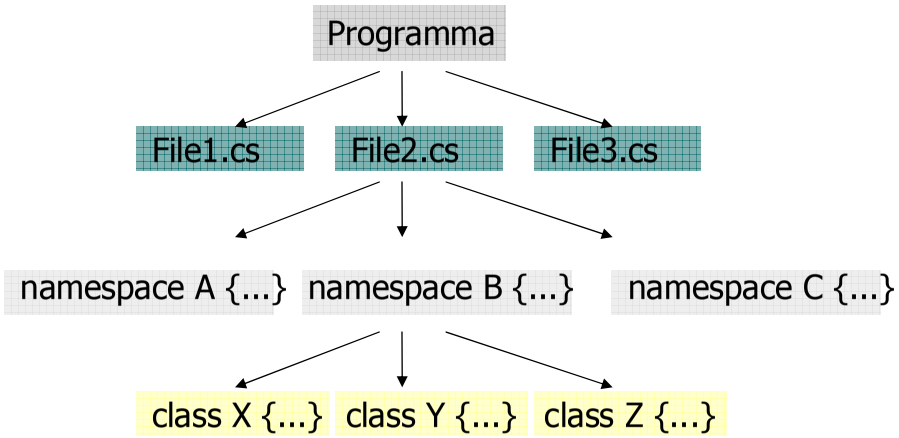
Hello World

- Lo sviluppo può chiaramente essere fatto anche da IDE come il Visual Studio, che offrono il valore aggiunto della syntax highlighting, indentazione automatica, completamento automatico
- To create the "Hello World" program in the Visual Studio IDE, choose File→New Project from the menu toolbar, e selezionare "Console Application" nella sezione Visual C#→Windows
- Dopo avere scritto il codice sorgente, si esegue la compilazione che genera il codice .exe;

Hello World

- the .exe file contains op-codes written in Microsoft Intermediate Language (MSIL); in addition to producing the IL code (which is similar in spirit to Java's byte-code), the compiler creates a read-only segment of the .exe file in which it inserts a standard Win32 executable header. The compiler designates an entry point within the read-only segment; the operating system loader jumps to that entry point when you run the program, just as it would for any Windows program.
- l'esecuzione può essere fatta dall'IDE o da riga di comando
- The operating system cannot execute directly the IL code, indeed the entry point in the Win32 header actually jump to the .NET Just In Time (JIT) compiler, which produces native CPU instructions (note however that it generates machine language code only for required function)

Struttura di un programma C#



- Se il namespace non è specificato => default namespace (anonimo)
- I namespace possono contenere structs, interfaces, delegates e enums
- I Namespace possono essere usati in altri file

Namespaces (C#) VS Packages (Java)

Java	C#
Packages are used to organize files or public types to avoid type conflicts. Package constructs can be mapped to a file system.	Namespaces are used to organize programs, both as an "internal" organization system for a program, and as an "external" organization system.
System.Security.Cryptography.AsymmetricAlgorithm aa; may be replaced: **import** System.Security.Crypography; class xxx { ... AsymmetricAlgorithm aa; There is no alias for packages. You have to use import statement or fully-qualified name to mention the specific type.	System.Security.Cryptography.AsymmetricAlgorithm aa; may be replaced: **using** System.Security.Crypography; AsymmetricAlgorithm aa; Alternatively, one could specify an alias for the the namespace, eg using myAlias = System.Security.Crypography; and then refer to the class with myAlias.AsymmetricAlgorithm

pacakge N1.N2;
class A {}
class B {}
or
package N1.N2;
class A {}
 	
//another source file
package N1.N2;
class B {}
 	
package cannot be nested.
One source file can only have one package statement.
	namespace N1.N2
	{
	class A {}
	class B {}
	}
	or
	namespace N1
	{
	namespace N2
	{
	class A {}
	class B {}
	}
	}

Namespaces (C#) VS Packages (Java)

- Java does not allow multiple packages in the same source file, while C# does allow multiple namespaces in a single .cs file
- In Java, Packages and classes are mapped to directories and files, whereas Namespaces and classes are not mapped to directories and files
- Java has visibility package, C# has only visibility internal (!= namespace)

Namespaces (C#) VS Packages (Java)

Visibility Modifier "internal"

- Visibility internal refers to what is visible during a compilation
- csc A.cs B.cs C.cs
  - All internal members of A, B, C see each other.
- csc /addmodule:A.netmodule,B.netmodule C.cs
  - C sees the internal members of A and B
  - (A and B may be written in different languages)

Un programma in 2 File

Counter.cs

```
class Counter {
  int val = 0;
  public void Add (int x) { val = val + x; }
  public int Val () { return val; }
}
```

Prog.cs

```
using System;
class Prog {
        static void Main() {
        Counter c = new Counter();
        c.Add(3); c.Add(5);
        Console.WriteLine("val = " + c.Val());
        }}
```

Compilazione

csc /target:exe Counter.cs Prog.cs

Execuzione

Prog

Uso delle DLL

csc /t:library Counter.cs
=> generates Counter.dll

csc /r:Counter.dll Prog.cs
=> generates Prog.exe

Identificatori

Sintassi

Identifier = (letter | '_' | '@') {letter | digit | '_'}.

- Unicode!
- Case-sensitive
- "@" permette di usare parole chiave come identificatori
  - if ... Parola chiave
  - @if ... Identificatore
- Possono contenere sequenze di escape Unicode (e.g. \u03c0 per π)

Esempi

someName
sum_of3
_10percent

@while	identificatore *while*
π	identificatore π
\u03c0	identificatore π
b\u0061ck	identificatore *back*

Parole chiave

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

77 parole chiave in C# rispetto alle 47 di Java

Naming

Maiuscole/minuscole

- Le parole scritte in maiuscolo (es. *ShowDialog*)
- La prima lettera in maiuscolo eccetto che per le variabili private o locali, le costanti e i campi

constants	*maiuscole*	*SIZE, MAX_VALUE*
local variables	*minuscole*	*i, top, sum*
private fields	*minuscole*	*data, lastElement*
public fields	*maiuscole*	*Width, BufferLength*
properties	*maiuscole*	*Length, FullName*
enumeration constants	*maiuscole*	*Red, Blue*
methods	*maiuscole*	*Add, IndexOf*
types	*maiuscole*	*StringBuilder*
predefined types	*minuscole*	*int, string*
namespaces	*maiuscole*	*System, Collections*

La prima parola

- Nomi di funzioni void devono iniziare con un verbo (e.g. *GetHashCode*)
- Gli altri nomi con un sostantivo (e.g. *size, IndexOf, Collections*)
- Gli enumerativi o i booleani possono iniziare con un aggettivo (*Red, Empty*)

Numeri Interi

Sintassi

DecConstant = digit {digit} {IntSuffix}.
HexConstant = "0x" hexDigit {hexDigit} {IntSuffix}.
IntSuffix = 'u' | 'U' | 'l' | 'L'.

Type

senza suffisso: il più piccolo fra int, uint, long, ulong
suffisso u, U: il più piccolo fra uint, ulong
suffissi l, L: il più piccolo fra long, ulong

Esempi

17 int
9876543210 long
17L long
17u uint
0x3fint
0x10000 long
0x3fL long

Numeri in Virgola mobile

Sintassi (semplificata)

RealConstant = [Digits] ["." [Digits]] [Exp] [RealSuffix].
deve contenere almeno una cifra e ".", oppure Exp
Digits = digit {digit}.
Exp = ("e" | "E") ["+" | "-"] Digits.
RealSuffix = "f" | "F" | "d" | "D" | "m" | "M".

Type

senza suffisso: double
suffisso f, F: float
suffisso d, D: double
suffisso m, M: decimal

Examples

3.14 double
1E-2 double
.1 double
10f float

Stringhe e costanti carattere

Sintassi

CharConstant = ' char '.
StringConstant = " {char} ".
char può essere qualsiasi carattere tranne virgolette, fine linea e \

Escape sequences

\' '
\" "
\\ \
\0 0x0000
\a 0x0007 (alert)
\b 0x0008 (backspace)
\f 0x000c (form feed)
\n 0x000a (new line)
\r 0x000d (carriage return)
\t 0x0009 (horizontal tab)
\v 0x000b (vertical tab)

Unicode- or hexadecimal escape sequences

\u0061 a
\x0061 a

Stringhe e costanti carattere (2)

Esempio di sequenze di escape

"file \"C:\\sample.txt\"" file "C:\sample.txt"
"file \x0022C:\u005csample.txt\x0022"

Se la stringa è preceduta da @

- \ non è considerato carattere di escape
- Ogni " deve essere raddoppiata
- La stringa può andare a capo

Esempio

@"file file
""C:\sample.txt"" "C:\sample.txt"

Commenti

Su singola linea

// un commento

Delimitati

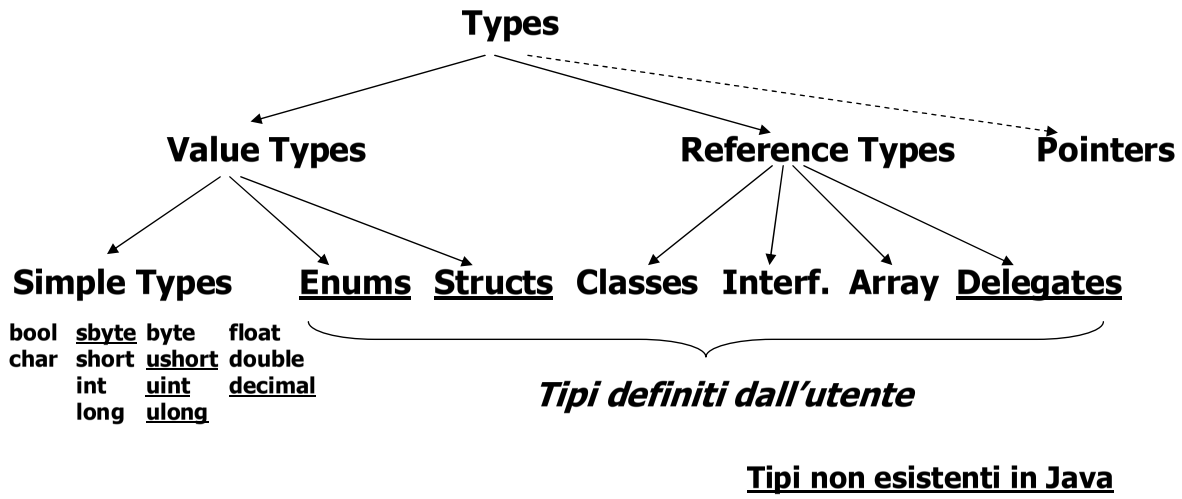
/* un'altro commento */

(non uno dentro l'altro)

Documentazione

/// un commento per la documentazione

Tipi di dato



Tutti i tipi sono derivati da object

- Possono essere assegnati ad una variabile di tipo *object*
- Tutti i metodi di object possono essere applicati a tutti i tipi

Tipi di dato

- Un programma C# infatti può essere definito come una collezione di tipi
- Tutti i dati e il codice sono definiti all'interno dei tipi, non esistono quindi funzioni o variabili globali
- I tipi contengono:
  - Data members
    - Fields, constants, arrays
    - Events
  - Function members
    - Methods, operators, constructors, destructors
    - Properties, indexers
  - Altri tipi
    - Classes, structs, enums, interfaces

Tipi di dato

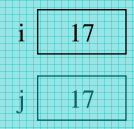
- I tipi possono essere istanziati…
  - …e quindi utilizzati: chiamando metodi, get and set properties, etc.
- Possono essere convertiti da un tipo ad un'altro
  - Implicitamente o esplicitamente
- I tipi sono organizzati
  - Namespaces, files, assemblies
- I tipi hanno un'organizzazione gerarchica
- Sono disponibili due principali categorie: tipi value e tipi riferimento

Tipo riferimento e tipo valore

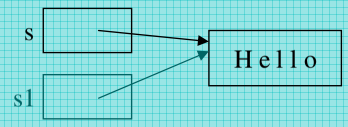
	Valore	Riferimento
contiene	valore	riferimento
memorizzata	stack	heap
inizializzata	0, false, '\0'	null
assegnamento	copia il valore	copia il riferimento

esempio

```
int i = 17;
int j = i;
```




```
string s = "Hello";
string s1 = s;
```



Tipi: Unified Type System

- Vantaggi tipi value
  - No heap allocation, quindi minor appesantimento del GC
  - Più efficenza nell'uso della memoria
  - Minor reference indirection
- Unified type system
  - Nessuna dicotomia primitive/object (ecco perchè "Unified")

Tipi primitivi

	long form	in Java	range
sbyte	System.SByte	byte	-128 .. 127
byte	System.Byte	---	0 .. 255
short	System.Int16	short	-32768 .. 32767
ushort	System.UInt16	---	0 .. 65535
int	System.Int32	int	-2^{31} .. $2^{31}-1$
uint	System.UInt32	---	0 .. $2^{32}-1$
long	System.Int64	long	-2^{63} .. $2^{63}-1$
ulong	System.UInt64	---	0 .. $2^{64}-1$
float	System.Single	float	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	double	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	---	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	boolean	true, false
char	System.Char	char	Unicode character

Rispetto ai tipi a virgola mobile, il tipo **decimal** è caratterizzato da una maggiore precisione (29 cifre significative vs 7 di float e 15 di double) e da un intervallo ridotto che lo rendono adatto per calcoli finanziari.

Tipi primitivi

C# Type	System Type	Size (bytes)	Signed?
sbyte	System.Sbyte	1	Yes
short	System.Int16	2	Yes
int	System.Int32	4	Yes
long	System.Int64	8	Yes
byte	System.Byte	1	No
ushort	System.UInt16	2	No
uint	System.UInt32	4	No
ulong	System.UInt64	8	No

Tipi predefiniti: floating point

- Segue lo standard IEEE 754
- Supporta ± 0, ±infinito, NaN

Tipo C#	System Type	Bytes
float	System.Single	4
double	System.Double	8

Tipi predefiniti: decimal

- 128bits
- 96 bits scalate di una potenza di 10
- Non supporta ± 0, ±infinito, NaN

Tipo C#	System Type	Bytes
decimal	System.Decimal	16

All integer types can be implicitly converted to a decimal type
Conversions between decimal and floating types require explicit conversion due to possible loss

Tipi predefiniti: bool

- Rappresenta i valori logici
- I suoi valori sono: true e false
- Non si possono usare 1 e 0
- Non esiste una conversione standard con altri tipi

Tipo C#	System Type	Bytes
bool	System.Boolean	1 (2 per array)

Tipi predefiniti: char

- Rappresenta un carattere UNICODE
- Literals
  - 'A' // Simple character
  - '\u0041' // Unicode
  - '\x0041' // Unsigned short hexadecimal
  - '\n' // Escape sequence character

Tipo C#	System Type	Bytes
char	System.char	2

decimal

128 bit in virgola fissa

$(-1)^s * c * 10^{-e}$

s = 0 or 1
$0 <= c < 2^{96}$
$0 <= e <= 28$

Per le operazioni
- Con grandi numeri
- Precisione alta (e.g. $0.1 = 1 * 10^{-1}$)

=> Necessari nei calcoli finanziari

Tipi predefiniti:reference type

- Object
- string

Object

- Root of object hierarchy
- Storage (book keeping) overhead
  - 0 bytes for value types
  - 8 bytes for reference types
- An actual reference (not the object) uses 4 bytes, since memory is allocated in chunks of 4 bytes at a time

Tipo C#	System Type	Bytes
object	System.Object	0/8 overhead

C#: La Classe System.Object

- La classe System.Object (cui corrisponde l'alias object) svolge in C# il ruolo della classe Object (java.lang.Object) di Java
  - è la capostipite di tutte le classi
  - definisce i metodi Equals (virtuale), GetType, ToString (virtuale), GetHashCode (virtuale),
  - definisce le funzioni statiche Equals/2 e ReferenceEquals/2
  - definisce inoltre alcuni metodi protetti come Finalize, MemberwiseClone
- ATTENZIONE: Equals per default verifica l'uguaglianza fra riferimenti, ma alcune classi derivate possono ridefinirlo e/o possono definire l'operatore ==. Ad esempio, string ridefinisce == così da verificare l'uguaglianza di contenuto, *ma non fa lo stesso con Equals!*

String

- An immutable sequence of Unicode characters
- Reference type
- Special syntax for literals
- string s = "I am a string";

Tipo C#	System Type	Bytes
string	System.String	Minimo 20

C# vs JAVA: STRINGHE

Java

- Le stringhe sono istanze della classe String
- Si creano nuove istanze stringa con new
- Si creano nuove istanze specificando stringhe costanti ("ciao")
- L'operatore == vale fra riferimenti (no aliasing): l'identità di contenuto è verificata dal metodo equals
- L'operatore != vale fra riferimenti (no aliasing): la diversità di contenuto è controllata da !equals

C#

- Le stringhe sono istanze della classe predefinita string
- Non si possono creare nuove istanze stringa con new
- Si creano nuove istanze specificando stringhe costanti ("ciao")
- L'operatore == è overloaded per le stringhe e valuta l'identità di contenuto (non esiste equals)
- L'operatore != è overloaded per le stringhe e valuta la diversità di contenuto (non esiste equals)

Test Uguaglianza In C#

l'operatore **==** è ridefinito per **string** → test verifica l' **identità di contenuto**

```
using System;
        class Esempio5 {
        public static void Main(){
        string s1 = "ciao", s2 = "ciao"; // new string è ERRATO
        Console.WriteLine((s1==s2) ? "uguali" : "diverse");
        Persona p1 = new Persona("John", 23),
        p2 = new Persona("Holly", 18);
        Console.WriteLine((p1==p2) ? "uguali" : "diverse");
        }}
class Persona {
        string nome;
        int eta;
        public Persona(string nome, int eta){
                this.nome=nome;
                this.eta =eta;}
```

l'operatore **==** non è ridefinito per **persona** → test **fra riferimenti** (aliasing)

System.String

Può essere usata come il tipo standard string

string s = "Alfonso";

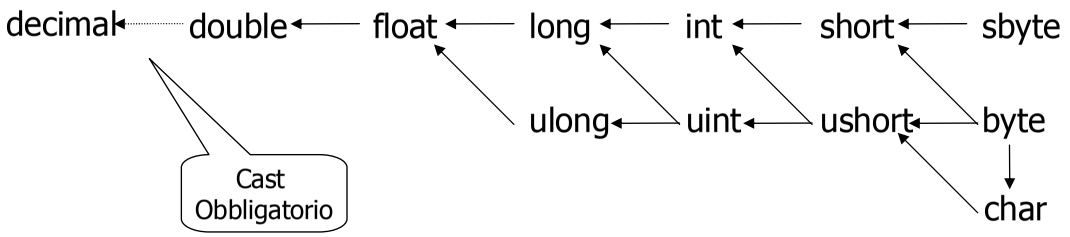
Note

- Le Strings non possono essere modificate (tipo StringBuilder per modificare)
- Possono essere concatenate +: "Don " + s
- Possono essere indicizzate: s[i]
- Calcolo della lunghezza: s.Length
- Le stringhe sono dei riferimenti e, quindi, adottano la semantica dei riferimenti nella assegnazioni
- == e != confrontano il contenuto: if (s == "Alfonso") ...
- La classe String definisce I metodi per manipolare le stringhe: CompareTo, IndexOf, StartsWith, Substring, ...

User-defined Types

- Enumerations — enum
- Arrays — int[], string[]
- Interface — interface
- Reference type — class
- Value type — struct
- Function pointer — delegate

Compatibilità fra i tipi primitivi



Le seguenti assegnazioni sono valide

intVar = shortVar;
intVar = charVar;
floatVar = charVar;
<u>decimalVar = (decimal)doubleVar;</u>

Attenzione alle conversioni con possibile perdita
(long→float→long, il numero long potrebbe cambiare

Tipi: conversioni

- Implicit conversions
  - Occur automatically
  - Guaranteed to succeed
  - No information (precision) loss
- Explicit conversions
  - Require a cast
  - May not succeed
  - Information (precision) might be lost
- Both implicit and explicit conversions can be user-defined

Tipi: conversioni

Esempi:

- int x = 123456;
- long y = x; // implicit
- short z = (short)x; // explicit
- double d = 1.2345678901234;
- float f = (float)d; // explicit
- long l = (long)d; // explicit

Tipi: conversioni

- Come in C++, in C# è possibile **definire operatori di conversione di tipo** per specificare come convertire (automaticamente o meno) un tipo in un altro
- Operatori di conversione:

  public static operator implicit tipoFinale(tipoIniz v)
  { return valoreConvertito }
  public static operator explicit tipoFinale(tipoIniz v)
  { return valoreConvertito }

- Gli operatori impliciti vengono chiamati automaticamente quando necessario, mentre quelli espliciti vengono chiamati solo in presenza di una conversione esplicita di tipo (cast)

Tipi: conversioni

Esempio di conversione implicita automatica da **Counter** a **int**:

```
Counter c = new Counter(); c.stato = 18;
int valore = c; // conv. implicita da Counter a int
...
public static implicit operator int(Counter c){return c.stato;}

public class Counter { // una classe qualsiasi
        private int val;
        public Counter(){ val=0;}
        public int stato { // proprietà
                                get { return val; }
                                set { if(value>0) val=value; }
}}
```

Tipi: conversioni

- Esempio di conversione esplicita (quindi non automatica, richiede un cast) da Counter a int:

```
Counter c = new Counter(); c.stato = 18;
int valore = (int)c; // conv. esplicita Counter -> int
public static explicit operator int(Counter c){return c.stato;}
public class Counter { // una classe qualsiasi
    int val;
    public Counter(){ val=0;}
    public int stato { // proprietà
                                    get { return val; }
                                    set { if(value>0) val=value; }
    }}
```

- Questo esempio ed il precedente sono entrambi possibili; in generale, si deve definire una conversione esplicita se il criterio stabilito potrebbe non avere successo e/o se si possono verificare perdite di precisione, se invece si è scelto un criterio che NON fallisce, si può adottare quella implicita

Tipi: conversioni

- L'operatore as consente di tentare una conversione esplicita di tipo (a run-time), ma senza scatenare un'eccezione in caso di fallimento
- Se la conversione non riesce, viene semplicemente restituito null

```
...
Counter c = new Counter();
if((c as int)!=null) {
    int valore=c++;
}
```

Enumeration

Lista di costanti

Dichiarazione (livello del namespace)

```
enum Color {Red, Blue, Green}    // valori: 0, 1, 2
enum Access {Personal=1, Group=2, All=4}
enum Access1 : byte {Personal=1, Group=2, All=4}
```

Uso

```
Color c = Color.Blue;  // un enumeratore deve essere qualificato
Access a = Access.Personal | Access.Group;
      // a contiene un insieme di valori
if ((Access1.Personal & a) != 0) Console.WriteLine("access granted");
```


Operazioni sulle enumerazioni

Operazioni valide

```
confronto   if (c == Color.Red) ...
            if (c> Color.Red && c <= Color.Green) ...
+, -        c = c + 2;
++, --      c++;
&           if ((c & Color.Red) == 0) ...
|           c = c | Color.Blue;
~           c = ~ Color.Red;
```

Il compilatore non verifica se il risultato è un valore appartenente all'insieme dei valori validi

Nota

- Non è possibile assegnare ad un int (tranne con il cast).
- Enumeration deriva da object (*Equals*, *ToString*, ...).
- La classe *System.Enum* esporta operazioni sulle enumerazioni (*GetName*, *Format*, *GetValues*, ...).

Nullable types

- I tipi nullable sono istanze della struttura System.Nullable. Un tipo nullable può rappresentare l'intervallo normale di valori per il relativo tipo di valore sottostante, più un valore **null** aggiuntivo.
- Ad esempio, al valore Nullable<Int32>, che si legge "Nullable di Int32", può essere assegnato qualsiasi valore compreso tra -2147483648 e 2147483647 oppure il valore **null**. A un valore Nullable<bool> può invece essere assegnato il valore **true** o **false** oppure **null**.
- La possibilità di assegnare **null** a tipi numerici e Boolean risulta particolarmente utile quanto si utilizzano database e altri tipi di dati contenenti elementi a cui non sarebbe possibile assegnare un valore. Ad esempio, in un campo Boolean di un database è possibile archiviare il valore **true** o **false** oppure è possibile lasciarlo indefinito.
- Il nullable type serve per estendere ai tipi primitivi il valore null già usato per i tipi riferimento

Nullable types

```
class NullableExample {
   static void Main()     {
      int? num = null;
      if (num.HasValue == true)
         { System.Console.WriteLine("num = " +num.Value); }
   else
         { System.Console.WriteLine("num = Null"); }
      //y is set to zero
      int y = num.GetValueOrDefault();
      // num.Value throws an InvalidOperationException if num.HasValue
   is false
      try
         {  y = num.Value;    }
   catch (System.InvalidOperationException e)
         {       System.Console.WriteLine(e.Message);        }
   }}
```

Nullable types

- Non è possibile creare un tipo nullable basato su un tipo di riferimento. (i tipi di riferimento supportano già il valore null)
- La sintassi T? è una forma abbreviata di System.Nullable<T>, in cui T è un tipo di valore. Le due forme sono intercambiabili.
- Assegnazione un valore a un tipo nullable int? x = 10; o double? d = 4.108;
- Utilizzare la proprietà System.Nullable.GetValueOrDefault per restituire il valore assegnato o il valore predefinito per il tipo sottostante se il valore è null, ad esempio int j = x.GetValueOrDefault();
- I tipi nullable nidificati non sono consentiti, ad esempio: Nullable<Nullable<int>> n; NON verrà compilata.

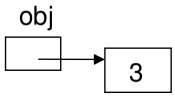
Nullable types

- Utilizzare le proprietà in sola lettura HasValue e Value per verificare se il valore è null e per recuperarlo, ad esempio if(x.HasValue) j = x.Value;
  - HasValue restituisce true se la variabile contiene un valore, false se è null.
  - La proprietà Value restituisce un valore se ne è stato assegnato uno, in caso contrario viene generata un'eccezione System.InvalidOperationException.
  - Per impostazione predefinita, una variabile di tipo nullable imposta HasValue su false. Value è indefinito.
- Utilizzare l'operatore ?? per assegnare un valore predefinito che verrà applicato quando un tipo nullable, il cui valore corrente è null, viene assegnato a un tipo non nullable, ad esempio int? x = null; int y = x ?? -1;

Boxing/Unboxing

- I tipi valore (int, struct, enum) sono compatibili con object (non sono necessarie classi wrapper)!

Boxing



l'istruzione

object obj = 3;

trasforma il valore nell'oggetto 3 nell'heap

Unboxing

l'istruzione

int x = (int) obj;

esegue l'operazione inversa

Boxing/Unboxing

- Permette l'implementazioni di contenitori generici

```
class Queue {
  ...
  public void Enqueue(object x) {...}
  public object Dequeue() {...}
  ...
}
```

- Queue può essere usato sia per i riferimenti che per i valori

```
Queue q = new Queue();
q.Enqueue(new Rectangle());
q.Enqueue(3);
Rectangle r = (Rectangle) q.Dequeue();
int x = (int) q.Dequeue();
```

Boxing/Unboxing

- In Java, i tipi primitivi non sono oggetti
  - quando occorre trattarli come oggetti, bisogna ricorrere alle classi wrapper
  - il passaggio da valore primitivo a oggetto e viceversa va svolto esplicitamente, per costruzione o tramite appositi metodi (intValue, floatValue, etc)
- In C#, i tipi primitivi non sono oggetti ma sono assimilabili ad essi quando fa comodo
  - non esistono classi wrapper esplicite
  - il passaggio da valore primitivo a oggetto e viceversa avviene automaticamente, tramite un procedimento noto come boxing / unboxing (l'unboxing richiede un cast)

Boxing

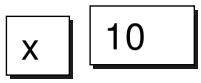
- Copies a value type into a reference type (object)
- Each value type has corresponding "hidden" reference type
- Note that a reference-type copy is made of the value type
- Value types are never aliased
- Value type is converted implicitly to object, a reference type
- Essentially an "up cast"

Unboxing

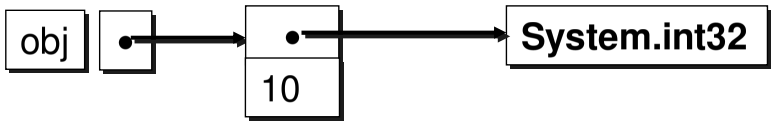
- Inverse operation of boxing
- Copies the value out of the box
- Copies from reference type to value type
- Requires an explicit conversion
- May not succeed (like all explicit conversions)
- Essentially a "down cast"

Boxing/Unboxing

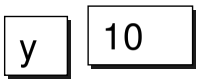
- Esempi:




```
// boxing
int x = 10;
object obj = x;
```




```
// unboxing
int y = (int)obj;
```



Boxing

- Vantaggi:
  - Enables polymorphism across all types
  - Collection classes work with all types
  - Eliminates need for wrapper classes
  - Replaces OLE Automation's Variant
    - In Microsoft Windows applications programming, **OLE Automation**, is an inter-process communication (IPC) mechanism based on Component Object Model (COM) that was intended for use by scripting languages – originally Visual Basic – but now are used by languages run on Windows. It provides an infrastructure whereby applications called *automation controllers* can access and manipulate (i.e. set properties of or call methods on) shared *automation objects* that are exported by other applications. It supersedes Dynamic Data Exchange (DDE), an older mechanism for applications to control one another. As with DDE, in OLE Automation the automation controller is the "client" and the application exporting the automation objects is the "server".
- Svantaggi
  - Performance cost
  - The need for boxing will decrease when the CLR supports generics (similar to C++ templates)

Formattazione Output

- Console.Write(intVal);
- Console.WriteLine(intVal);
- Console.Write("Hello {0}", name); //
- Console.WriteLine("{0} = {1}", x, y);

Placeholder

"{" n ["," width] [":" format [precision]] "}"

n posizione dell'argomento

width dimensione minima

format codice di formato (e.g. d, f, e, x, ...)

precision numero di cifre (dopo la virgola)

Formattazione Output

Esempi:

```
int x = 17;
Console.WriteLine("{0}", x);   17
Console.WriteLine("{0,5}", x);       17
Console.WriteLine("{0:d}", x);    17
Console.WriteLine("{0:d5}", x);   00017
Console.WriteLine("{0:f}", x); 17.00
Console.WriteLine("{0:f1}", x);   17.0
Console.WriteLine("{0:E}", x);    1.700000E+001
Console.WriteLine("{0:E1}", x);   1.7E+001
Console.WriteLine("{0:x}", x);    11
Console.WriteLine("{0:x4}", x);   0011
```


Formattazione Output

Formattazione di stringhe:

```
using System;
using System.Globalization;
class formattazione {
    static void Main(){
// Format a negative integer or floating-point number in various ways.
    Console.WriteLine("Standard Numeric Format Specifiers");
    string s = String.Format(        "(C) Currency: . . . . . . . . {0:C}\n" +
        "(D) Decimal:. . . . . . . . . {0:D}\n" +
        "(E) Scientific: . . . . . . . {1:E}\n" +
        "(F) Fixed point:. . . . . . . {1:F}\n" +
        "(G) General:. . . . . . . . . {0:G}\n" +
        "    (default):. . . . . . . . {0} (default = 'G')\n" +
        "(N) Number: . . . . . . . . . {0:N}\n" +
        "(P) Percent:. . . . . . . . . {1:P}\n" +
        "(X) Hexadecimal:. . . . . . . {0:X}\n",        -123, -789.45f);
    Console.WriteLine(s);…
```

Formattazione Output

Uscita:

-stringa0-=- 000c-
Standard Numeric Format Specifiers
(C) Currency: . . . . . -? 123,00
(D) Decimal:. . . . . -123
(E) Scientific: . . . . . -7,894500E+002
(F) Fixed point:. . . . . -789,45
(G) General:. . . . . -123
(default):. . . . . -123 (default = 'G')
(N) Number: . . . . . -123,00
(P) Percent:. . . . . -78.945,00%
(R) Round-trip: . . . . . -789,45
(X) Hexadecimal:. FFFFFF85

Formattazione Output

```
…
float i=43.18F;
s = i.ToString("c", new CultureInfo("en-GB"));
Console.WriteLine("\nLOCALITAZION Great Britain : "+ s+"\n");
s = i.ToString("c", new CultureInfo("it-CH"));
Console.WriteLine("\nLOCALITAZION Switzerland : "+ s+"\n");

// Format the current date in various ways.
    Console.WriteLine("Standard DateTime Format Specifiers");
    DateTime thisDate = DateTime.Now;
    s = String.Format(        "(d) Short date: . . . . . {0:d}\n" +
        "(D) Long date:. . . . . {0:D}\n" +
        "(t) Short time: . . . . . {0:t}\n" +
        "(T) Long time:. . . . . {0:T}\n",        thisDate);
    Console.WriteLine(s);}
```

Formattazione Output

Uscita:

LOCALITAZION Great Britain : £43.18
LOCALITAZION Switzerland : SFr. 43.18

Standard DateTime Format Specifiers
(d) Short date: 23/11/2008
(D) Long date:. . . . . domenica 23 novembre 2008
(t) Short time: 18.50
(T) Long time:. 18.50.15
(f) Full date/short time: . . domenica 23 novembre 2008 18.50
(F) Full date/long time:. . . domenica 23 novembre 2008 18.50.15
(g) General date/short time:. 23/11/2008 18.50
(G) General date/long time: . 23/11/2008 18.50.15
(default):. . . . . 23/11/2008 18.50.15 (default = 'G')
(M) Month:. . . . . 23 novembre
(R) RFC1123:. . . . . Sun, 23 Nov 2008 18:50:15 GMT
(Y) Year: . . . . . novembre 2008

Output su File

```
using System;
using System.IO;

class Test {

 static void Main() {
      FileStream s = new FileStream("output.txt", FileMode.Create);
      StreamWriter w = new StreamWriter(s);

      w.WriteLine("Table of squares:");
      for (int i = 0; i < 10; i++)
           w.WriteLine("{0,3}: {1,5}", i, i*i);

      w.Close();
 }

}
```

Output su File

Uscita:

D:|>more output.txt
Table of squares:
0: 0
1: 1
2: 4
3: 9
4: 16
5: 25
6: 36
7: 49
8: 64
9: 81

Input

Input da tastiera:

```
int ch = Console.Read();
String s = Console.ReadLine();
```

Input da file:

```
using System;
using System.IO;
class Test {
 static void Main() {
      FileStream s = new FileStream("input.txt", FileMode.Open);
      StreamReader r = new StreamReader(s);
      string line = r.ReadLine();
      while (line != null) {
           Console.WriteLine("Lettura da file…:"+line);
           line = r.ReadLine();
      }
      r.Close();    }}
```

Input da File

Uscita:

Lettura da file :file da leggere (riga1)

Lettura da file :riga2

Lettura da file :riga3

Lettura da file :4...

Lettura da file :5...

Lettura da file :penultima riga (6)

Lettura da file :ultima (riga 7)

Parametri da linea di comando

```
using System;

class Test {

 static void Main(string[] arg) {  //invocato come Test value = 3
      for (int i = 0; i < arg.Length; i++)
           Console.WriteLine("{0}: {1}", i, arg[i]);
                //   0: value
                //   1: =
                //   2: 3
            foreach (string s in arg)
             Console.WriteLine(s); // output:
                //   value
                //   =
                //   3
 }
}
```