

# Il Linguaggio C#

## TUTORIAL PER PROGRAMMATORI JAVA

ANGELO CROATTI

a.croatti@unibo.it

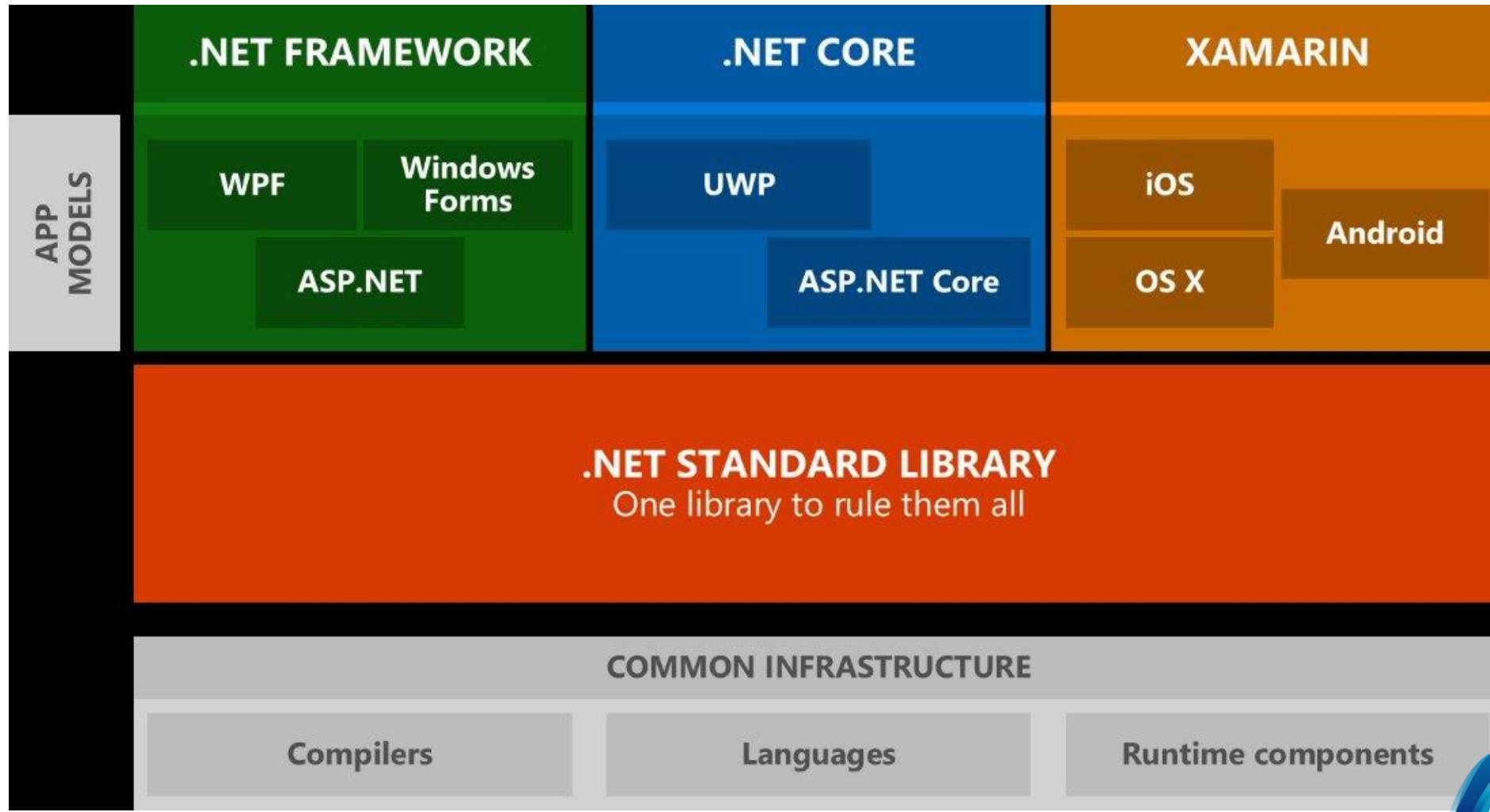


ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA  
Corso di studi in Ingegneria e Scienze Informatiche

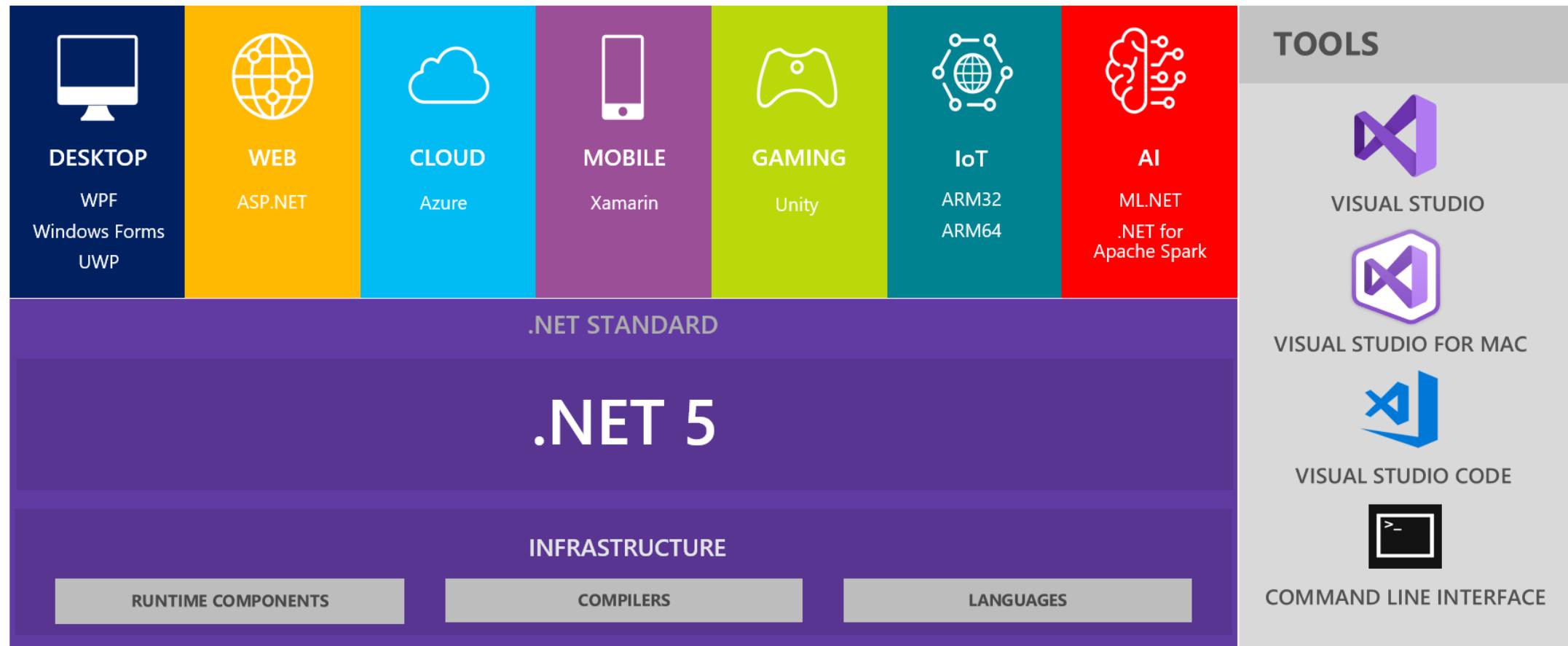
Programmazione ad Oggetti – A.A. 2019/2020

# OVERVIEW

# Microsoft .NET Development Platform

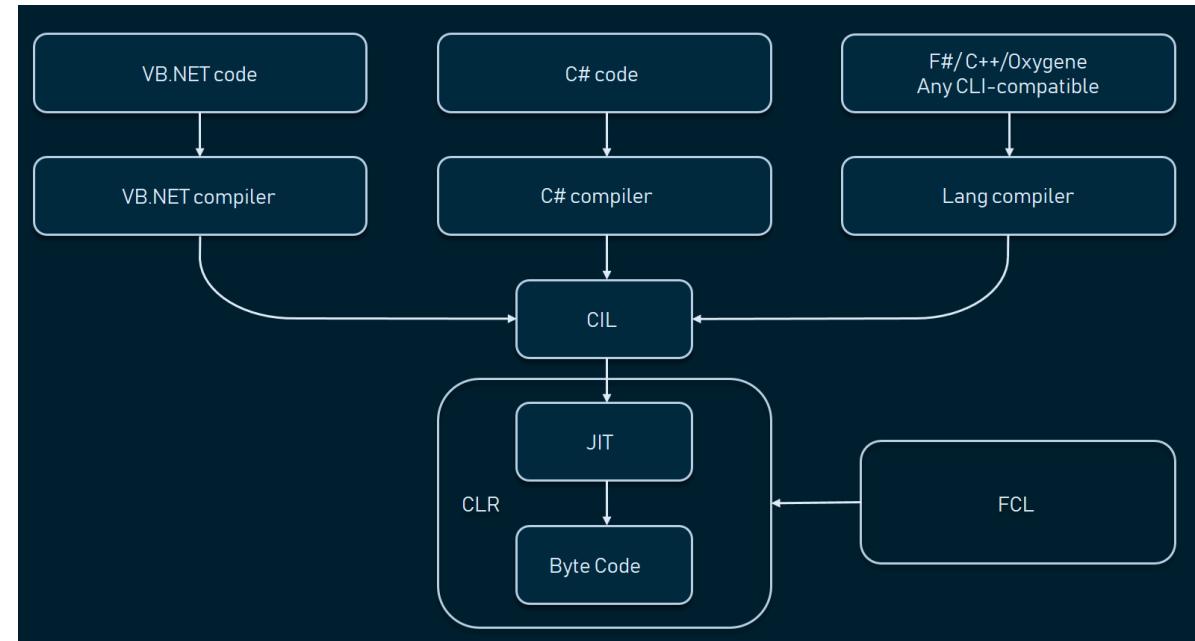


# Microsoft .NET Development Platform



# Microsoft .NET Framework

- Software Framework sviluppato da Microsoft
  - Include un'ampia **Class Library** e consente di compilare software scritti in diversi linguaggi supportati (principalmente **C#**).
- I programmi scritti per .NET sono eseguiti in un ambiente virtuale noto come **CLR** (Common Language Runtime)
  - Fornisce servizi per la gestione della memoria, la gestione delle eccezioni...
  - I programmi sono compilati in un linguaggio intermedio chiamato **CIL** (Common Intermediate Language)
  - Il codice CIL è poi compilato Just-in-Time (**JIT**) in codice macchina dalla CLR in fase di esecuzione, con riferimento alla specifica piattaforma, iniettando ciò che occorre del Framework Class Library (**FCL**).
- Analogia con ecosistema Java
  - Java ↔ C#
  - JDK ↔ Class Library
  - Bytecode ↔ CIL
  - JVM ↔ CLR



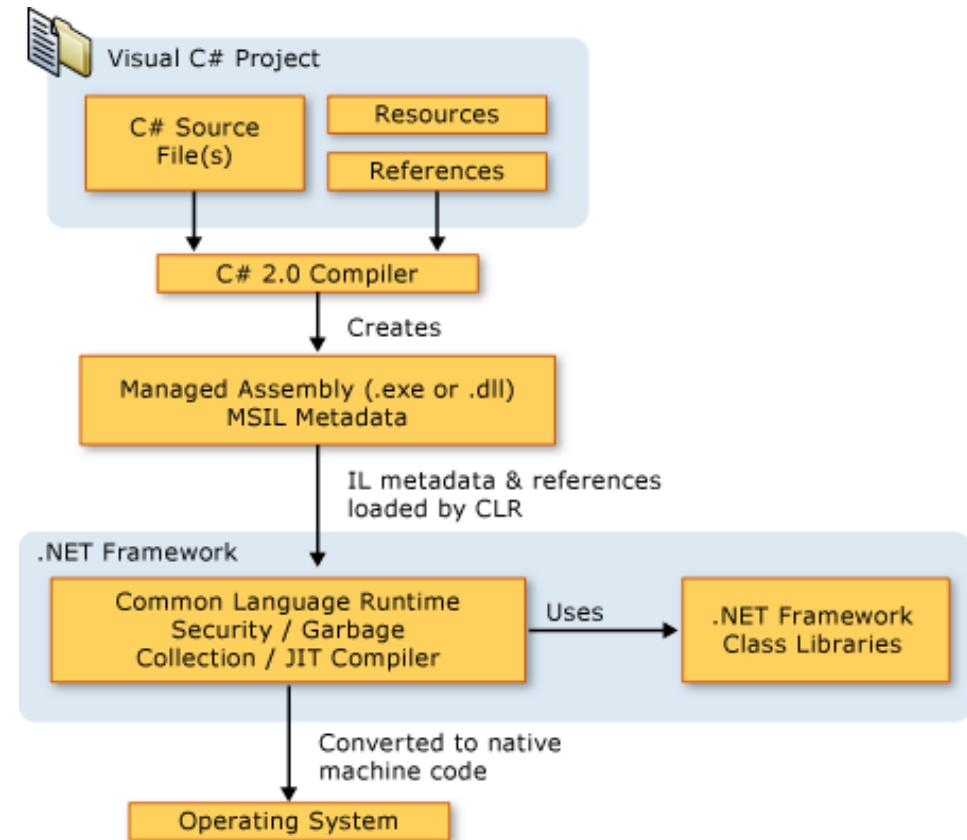
# Linguaggio C# - Caratteristiche generali

## ○ Linguaggio (general-purpose) principale di .NET

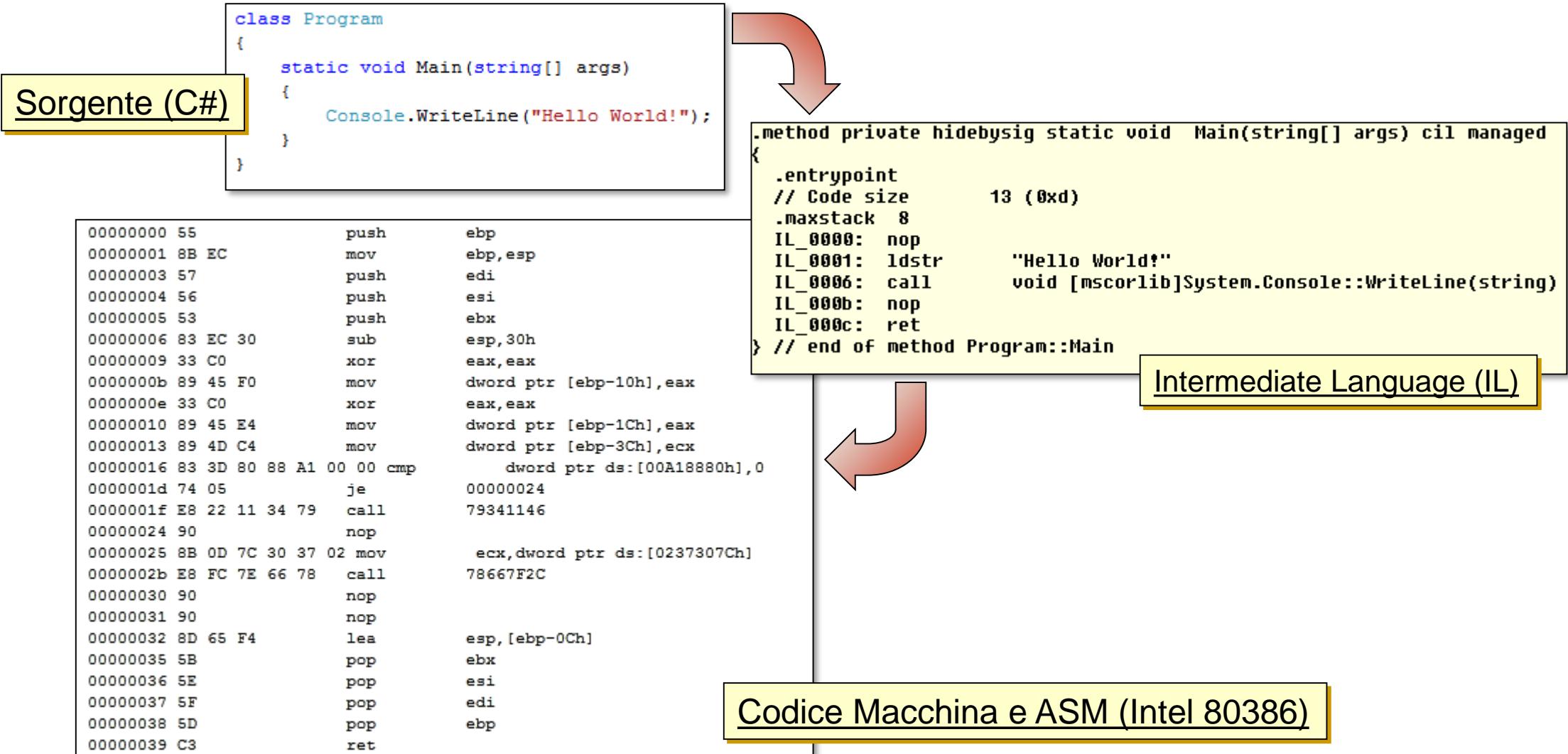
- Prevalentemente basato sul **paradigma ad oggetti**, ma dalle ultime versioni è definibile come linguaggio **multi-paradigma**
- È standardizzato Ecma (ECMA-334) e USI (ISO/IEC 23270:2006)
- Nato nel 2000, è attualmente disponibile nella versione 7.0 (datata 2017)
- Attualmente è disponibile per tutte le piattaforme (Windows, Linux, ...) e il relativo compilatore è stato rilasciato come open-source code.
- L'IDE principale per lo sviluppo in C# è **Microsoft Visual Studio**

## ○ Dal Sorgente all'Esecuzione

- Il compilatore C# produce codice intermedio (CIL)
- Codice e risorse (es. immagini) formano uno o più **Assembly**
  - contenuti in appositi file (.exe o .dll in Windows)
- Il programma C# può utilizzare la Class Library che è messa a disposizione tramite il .NET Framework
- La CLR carica il codice CIL e ne traduce le parti da eseguire in linguaggio macchina, compilandole mediante JIT



# Un esempio: C# > CIL > ASM



# Quale Ambiente di Sviluppo (IDE) per .NET ?

---

- In ambienti Windows o MacOS, l'IDE più indicato e usato per .NET è **Visual Studio**
  - Molto maturo
  - Closed source, [versione Community gratuita](#) + versioni a Professional ed Enterprise a pagamento
    - Gli studenti UNIBO possono ottenere una licenza Enterprise gratuitamente tramite il programma [Microsoft Image](#) di UNIBO
  - Supporta molti linguaggi: C, C#, F#, Vb.NET, Python, etc..
  - Designer WYSIWYG per WinForms & WPF molto flessibile ed efficace
  - Ottimo per sviluppare... in/per ambienti Windows
- **MonoDevelop** è l'alternativa Open Source, e.g. per gli utilizzatori di sistemi Linux-based
  - Basato su **Mono**, il porting Open Source di .NET
  - Ha raggiunto ormai una maturità soddisfacente, seppure non sia ancora al livello di Visual Studio
  - Non supporta né WinForms né WPF (eredità di Mono) ma supporta invece GTK 2
  - Disponibile (così come Mono) sia [per Linux](#), che per Windows e MacOS
    - Su Windows va però compilato, NON è disponibile un pacchetto auto-installante

# IDE alternativi per .NET

---

- L'IDE **Rider**, di JetBrains, è forse l'alternativa migliore a Visual Studio
  - Funziona sia su Windows, che su MacOS, che su Linux
  - È disponibile **solo** previo acquisto di una licenza
  - Però è possibile registrarsi a **JetBrains** con un account **@(studio.)unibo.it** e ottenere una licenza gratuita
  - Molto comodo per chi è abituato a IntelliJ Idea e/o Android Studio
  
- È possibile configurare opportunamente editor come:
  - Visual Studio Code
  - Atom
  - Sublime Textmediante l'installazione di appositi plugin dai rispettivi Plugin-Store/Marketplace

# C# e Java

---

- Entrambi sono linguaggi object-oriented, class based
- Progettati per essere eseguiti su macchina virtuale (CLR e JVM)
- Entrambi supportano la Garbage Collection e i meccanismi di Exception Handling
- Appartengono alla categoria dei linguaggi curly brace (come C, C++, ...)
- Sono entrambi linguaggi **statically and strongly typed**
- Entrambi **non** supportano l'ereditarietà multipla
- ...

# C# vs. Java

---

«*C# is an imitation of Java with reliability, productivity and security deleted*»

- **James Goslin**, ideatore del linguaggio Java

«*Java and C# are almost identical programming languages.  
Boring repetition that lacks innovation*»

«*C# borrowed a lot from Java, and vice versa*»

- **Klaus Kreft e Angelika Langer**, autori di molti libri sul linguaggio C++

«*C# is not a Java clone and is much closer to C++ in its design*»

- **Anders Hejlsberg**, team leader del progetto C#

# PARTE 1

## Analogie e differenze tra C# e Java

# Convenzioni di scrittura del codice

- Una sola istruzione/dichiarazione per linea
  - Indentazione con spazi al posto della tabulazione
  - Separazione di ciascuna definizione di metodo dalle altre con almeno una linea vuota
- Parentesi graffa aperta
  - (Java) in fondo alla linea precedente
  - (C#) a capo su nuova linea
- Variabili locali e parametri dei metodi
  - camelCase
- Tipi (es. classi/interfacce)
  - PascalCase
- Metodi
  - (Java) camelCase
  - (C#) PascalCase

» <https://msdn.microsoft.com/en-us/library/ff926074.aspx>

```
using System;

namespace it.unibo.oop.example01
{
    class Program
    {
        private static int mySum;

        public static void Main(string[] args)
        {
            mySum = SumTwoNum(10, 5);

            Console.WriteLine(mySum.ToString());
        }

        private static int SumTwoNum(int firstPar, int secondPar)
        {
            return firstPar + secondPar;
        }
    }
}
```

# Struttura dei programmi

## ANALOGIE

- Non vi sono file «header» (come accade invece in C/C++)
  - il codice tutto è scritto «in-line»
- Classi raggruppate all'interno di un contenitore
  - «**namespace**» in C#, «**package**» in Java
- Metodo «main» come entry-point
  - In C# inizia con la 'M' maiuscola, come è convenzione in .Net per i metodi pubblici
  - Deve essere definito con il modificatore «**static**» e può accettare i command-line parameters

```
public static void Main()
{
    //TODO
}

public static void Main(string[] args)
{
    //TODO
    string par1 = args[0];
}
```

## DIFFERENZE

- In C# la struttura *fisica* di file/directory non deve corrispondere a quella *logica* di classi/namespace
  - Non vi sono restrizioni al numero di classi e namespace (annidati o meno) che possono essere contenuti in un file
  - Per importare classi contenute in namespace non presenti in quello corrente, si utilizza la keyword «**using**» (equivalente a «import» in Java)
- In C# è possibile dividere una classe fra più file
  - Più file possono specificare lo stesso nome di classe preceduto dalla keyword «**partial**»
  - In caso di ereditarietà, è sufficiente che solo una delle parti definisca la classe base

# Struttura dei programmi – Esempi



```
package it.unibo.oop.example01;

public class Program {
    /*
     * Main method
     * Application Entry-Point
     */
    public static void main(String[] args) {
        //Print a message on the StdOut
        System.out.println("Hello, World!");
    }
}
```



```
using System;

namespace it.unibo.oop.example01
{
    class Program
    {
        /*
         * Main method
         * Application Entry-Point
         */
        static void Main(string[] args)
        {
            //Print a message on the StdOut
            Console.WriteLine("Hello, World!");
        }
    }
}
```

# Console I/O

---

## ANALOGIE

- Le «console application», che leggono e scrivono sullo standard input e output senza interfaccia grafica, hanno struttura simile
- Entrambi i linguaggi dispongono di apposite classi per input e output sulla console
  - Java: `System.out.println("testo")` oppure `System.in.read()`
  - C#: `Console.WriteLine("testo")` oppure `Console.Read()`

## DIFFERENZE

- I dettagli delle classi e dei relativi metodi variano
- Le stringhe di formato usano sequence di escape differenti:
  - Java usa specificatori di formato simili al C (es. `%d`), ma estendibili a nuovi tipi mediante l'interfaccia `Formattable`
  - C# usa specificatori che hanno la forma `{<indice>[, allineamento] [: formato]}` ed è possibile introdurre formati per nuovi tipi implementando l'interfaccia `IFormatable`

# Console I/O – Esempio

```
public static void main(String[] args)
{
    DataInput in = new DataInputStream(System.in);

    System.out.println("Come ti chiami?");
    String name = in.readLine();

    System.out.println("Quanti anni hai?");
    int age = Integer.parseInt(in.readLine());
    System.out.printf("%s ha %d anni", name, age);

    System.out.println("Valore intero a:");
    int a = Integer.parseInt(in.readLine());
    System.out.println("Valore intero b:");
    int b = Integer.parseInt(in.readLine());

    System.out.println("1) a+b\n2) a*b");
    int c = System.in.read();

    int r;
    switch (c){
        case '1': r = a+b; break;
        case '2': r = a*b; break;
        default: System.out.println("Err"); return;
    }

    System.out.printf("Risultato: %d", r);
}
```



```
public static void Main()
{
    Console.WriteLine("Come ti chiami?");
    string name = Console.ReadLine();

    Console.WriteLine("Quanti anni hai?");
    int age = int.Parse(Console.ReadLine());
    Console.WriteLine("{0} ha {1} anni", name, age);

    Console.WriteLine("Valore intero a:");
    int a = int.Parse(Console.ReadLine());
    Console.WriteLine("Valore intero b:");
    int b = int.Parse(Console.ReadLine());

    Console.WriteLine("1) a+b\n2) a*b");
    int c = Console.Read();

    int r;
    switch (c)
    {
        case '1': r = a + b; break;
        case '2': r = a * b; break;
        default: Console.WriteLine("Err"); return;
    }

    Console.WriteLine("Risultato: {0}", r);
}
```



# Tipi di dato

## ANALOGIE

- Tipi primitivi in Java e «tipi valore» predefiniti in C# (**int**, **char**, **byte**, **double**, **bool**, ...)
  - Si noti che in C# il tipo valore relativo al tipo Boolean è definito con la keyword «**bool**»
- Tipi riferimento (reference type): **object**, **string**, array, classi, interfacce
  - Si noti che in C# il tipo riferimento per le stringhe è accessibile con la keyword «**string**» ('s' minuscola)
- Costanti: keyword «**final**» in Java, «**const**» e «**readonly**» in C#
- Cast e conversioni (es. conversione a stringa con metodo `ToString()`)

## DIFFERENZE

- In C# non si usano classi «wrapper» per i tipi primitivi: i tipi valore predefiniti sono sempre visti come «derivati» da **object**, mediante boxing automatico
  - Si possono utilizzare sia le keyword del linguaggio (es. «**object**», «**string**», «**int**»), sia i nomi dei tipi corrispondenti nella class library (con la maiuscola es. «**Object**», «**String**», «**Int32**»).
- Enum – sono semplici tipi valore in C#, mentre sono vere e proprie classi in Java
  - In C# possono essere associati esplicitamente i valori integer associati ai membri dell'enumerato

# Tipi di dati – Esempi

JAVA

```
// Tipi valore - Esempi di tipi valore predefiniti
boolean flag = true;
byte a = -128; // N.B. con segno

char c = 'e';
// Altri tipi valore predefiniti: short, int, long,
// float, double

// Tipi riferimento predefiniti
Object o = new Object();
String s = "Java";
// Tipi riferimento definibili dal programmatore:
// array, class, interface

// Costanti
final double Value = 42;

// Esempi di conversioni
int x = 42;
String y = Integer.toString(x); // y = "42"
x = Integer.parseInt(y);
double z = 3.5;
x = (int) z; // x = 3 (tronca i decimali)
// Esempi di enum
enum Action {START, STOP, REWIND, FORWARD};
Action a = Action.STOP;
if (a != Action.START)
    System.out.println(a); // "Stop"
```

C#

```
// Tipi valore - Esempi di tipi valore predefiniti
bool flag = true;
byte a = 255; // N.B. senza segno
sbyte b = -128; // N.B. con segno
char c = 'è';
// Altri tipi valore predefiniti: short, ushort, int, uint,
long,
// ulong, float, double, decimal

// Tipi riferimento predefiniti
object o = new object();
string s = "C#";
// Tipi riferimento definibili dal programmatore:
// array, class, interface, delegate

// Costanti
const double PI = 3.14; // iniz. a compile-time
readonly int Value = 42; // iniz. a runtime

// Esempi di conversioni
int x = 42;
string y = x.ToString(); // y = "42"
x = int.Parse(y); // o x = Convert.ToInt32(y);
double z = 3.5;
x = (int) z; // x = 3 (tronca i decimali)

// Esempi di enum
enum Action { START, STOP, REWIND, FORWARD };
Action a = Action.STOP;
if (a != Action.START)
    Console.WriteLine(a); // "Stop"
```

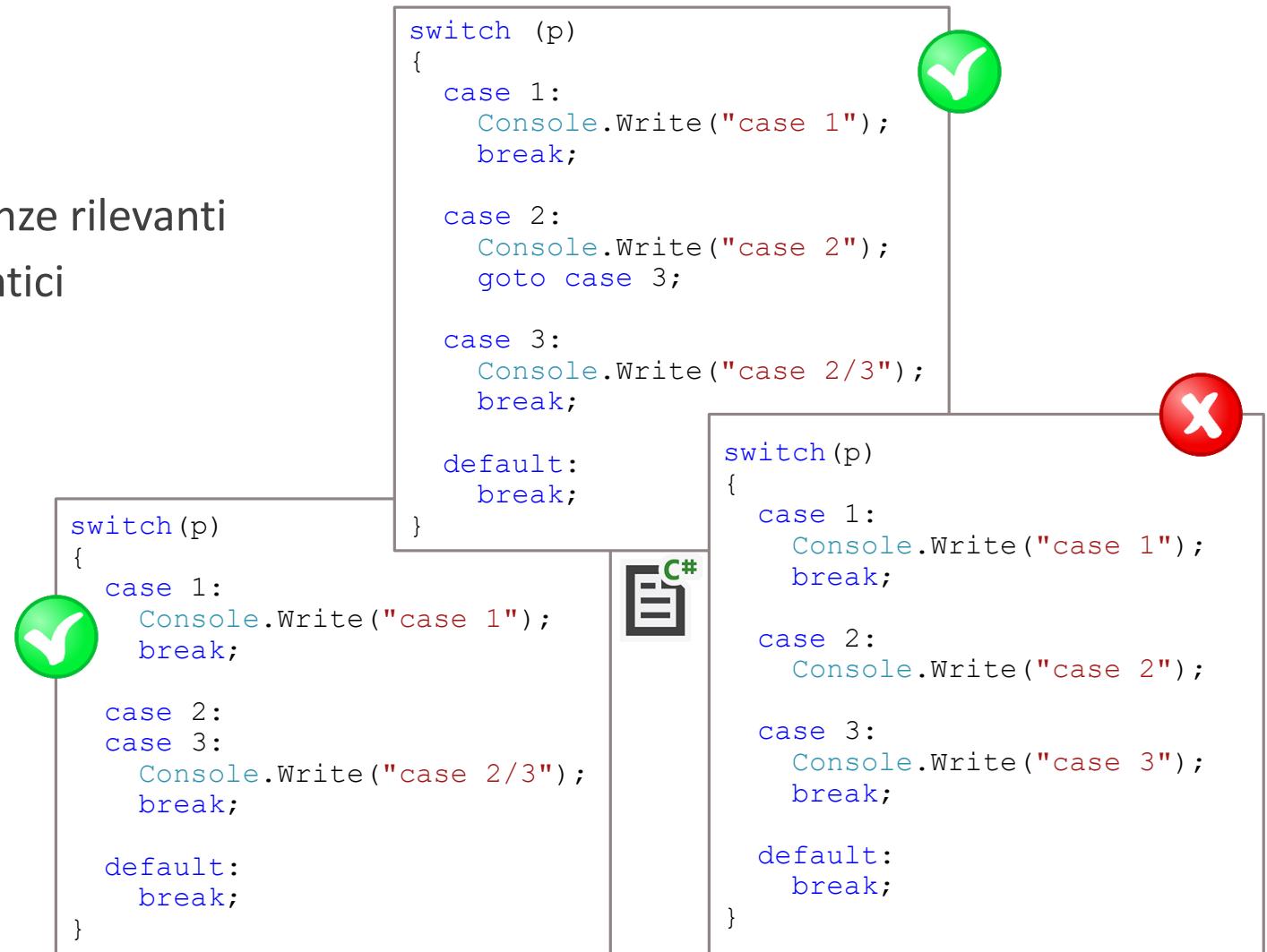
# Istruzioni: controllo di flusso e cicli

## ANALOGIE

- «**if**», «**else**», «**else if**»: identici
- «**switch**»: analogo ma con alcune differenze rilevanti
- cicli «**for**», «**while**», «**do...while**»: identici
- ciclo «**foreach** (... **in** ...) » in C#
  - corrisponde a «**for** (... : ...) » in Java

## DIFFERENZE

- switch: C# non consente il «passaggio» da un «**case**» al successivo omettendo il «**break**», ma è possibile esplicitamente trasferire il controllo a un qualsiasi «**case**» con la sintassi «**goto case ...**».



# Istruzioni: controllo di flusso e cicli – Esempi



```
int j = 0;
while (j < 10){
    j++;
}

for (int i = 2; i <= 10; i += 2){
    System.out.println(i);
}

do{
    j++;
} while (j < 20);

for (int i : array){
    sum += i;
}

ArrayList<Object> list = new ArrayList<Object>();
list.Add(42);
list.Add("unibo");
list.Add(3.14);

for (Object o : list){
    System.out.println(o);
}
```



```
int j = 0;
while (j < 10)
{
    j++;
}

for (int i = 2; i <= 10; i += 2)
{
    System.Console.WriteLine(i);
}

do
{
    j++;
} while (j < 20);

foreach (int i in array)
{
    sum += i;
}

ArrayList list = new ArrayList();
list.Add(42);
list.Add("unibo");
list.Add(3.14);

foreach (object o in list)
{
    System.Console.WriteLine(o);
}
```

# Caratteri e stringhe

## ANALOGIE

- Tipo `char` – caratteri con codifica UTF-16
- Classe `string`: sequenza di caratteri *immutabile* (qualsiasi operazione su stringhe crea una nuova stringa)
- Sequenze di escape in caratteri o stringhe (es. `'\0'`, `"\n\t\"`, `'\u00C4'`, ...)
- Concatenazione efficiente di stringhe mediante apposite classi
  - `StringBuilder` in C#, `StringBuffer` in Java

## DIFFERENZE

- In C# il prefisso `@` consente di definire un «**verbatim string literal**»
  - tutti i caratteri che seguono sono «interpretati così come sono» (utile se si devono ignorare le sequenze di escape o scrivere stringhe multi-linea)
  - Es. `@"C:\temp\"` equivale a `"C:\\temp\\\"`
- In C# gli operatori `==` e `!=` sono ridefiniti sulla classe `string` e confrontano i caratteri invece dei riferimenti
- In C# **string** implementa un indexer per cui è possibile accedere ai singoli caratteri con la sintassi degli array
  - Es. `char c1 = testo[i]` oppure `char c2 = "mio testo"[2]`

# Caratteri e stringhe – Esempi

```
char newLine = '\n';
int n = (int)newLine; // 13
char test = (char)13; // '\n'

// Concatenazione
String u = "Bologna";
u = "Università di " + u; // "Università di Bologna"

// Confronto
String c = "Cesena";
boolean e1 = c == "Cesena"; // Attenzione...
boolean e2 = c.equals("Cesena"); // true
boolean e3 = c.equalsIgnoreCase("CESENA"); // true
boolean e4 = c.compareTo("Cesena") == 0; // true

// Formattazione date
Calendar d = new GregorianCalendar(1974, 11, 29);
String s = String.format("Data: %1$td/%1$tm/%1$ty", d);

// Modifica e concatenazione efficiente
StringBuffer buffer = new StringBuffer("due ");
buffer.append("tre ");
buffer.insert(0, "uno ");
buffer.replace(4, 7, "DUE");
System.out.println(buffer); // "uno DUE tre"

for (char c : s.toCharArray()) { System.out.println(c); }
```



```
char newLine = '\n';
int n = (int)newLine; // 13
char test = (char)13; // '\n'

// Concatenazione
string u = "Bologna";
u = "Università di " + u; // "Università di Bologna"

// Confronto
string c = "Cesena";
bool e1 = c == "Cesena"; // confronta i caratteri
bool e2 = c.Equals("Cesena"); // true
bool e3 = c.Equals("Cesena", StringComparison.InvariantCultureIgnoreCase); //true
bool e4 = c.CompareTo("Cesena") == 0; // true

// Formattazione date
DateTime d = new DateTime(1974, 12, 29);
string s = string.Format("Data: {0:dd/MM/yyyy}", d);

// Modifica e concatenazione efficiente
StringBuilder buffer = new StringBuilder("due ");
buffer.Append("tre ");
buffer.Insert(0, "uno ");
buffer.Replace("due", "DUE");
Console.WriteLine(buffer); // "uno DUE tre"

foreach (char c in s) { Console.WriteLine(c); }
```



# Operatori

## ANALOGIE

- Java e C# condividono i principali operatori
  - +, -, \*, /, ++, --, &&, ||, ?:, ==, =,

## DIFFERENZE

- L'operatore Java `>>>` (unsigned right shift) non esiste in C#
  - non è necessario poiché il C# supporta variabili intere unsigned
- L'operatore C# corrispondente a «`instanceof`» di Java è «**is**»

Java	C#	Description
<code>x.y</code>	<code>x.Y</code>	Member access "dot" operator
<code>f(x)</code>	<code>f(x)</code>	Method invocation operator
<code>a[x]</code>	<code>a[x]</code>	Array element access operator
<code>++, --</code>	<code>++, --</code>	Increment and decrement operators (pre and post-fix)
<code>new</code>	<code>new</code>	Object instantiation operator
<code>instanceof</code>	<code>is</code>	Type verification operator
<code>(T)x</code>	<code>(T)x</code>	Explicit cast operator
<code>+, -</code>	<code>+, -</code>	Addition and subtraction operators (binary).
		Positive and negative operators (unary)
<code>+</code>	<code>+</code>	String concatenation operator
<code>!</code>	<code>!</code>	Logical negation operator
<code>&amp;&amp;,   </code>	<code>&amp;&amp;,   </code>	Conditional AND and OR operators (short-circuited evaluation)
<code>&amp;,  , ^</code>	<code>n/a</code>	Conditional AND, OR, and XOR operators (full evaluation of operands)
<code>~</code>	<code>~</code>	Bitwise complement operator
<code>&amp;,  , ^</code>	<code>&amp;,  , ^</code>	Bitwise AND, OR, and XOR operators
<code>&lt;&lt;, &gt;&gt;</code>	<code>n/a</code>	Signed left-shift and right-shift operators
<code>&gt;&gt;</code>	<code>&gt;&gt;</code>	Unsigned right-shift operator
<code>*, /, %</code>	<code>*, /, %</code>	Multiply, divide, and modulus operators
<code>==, !=</code>	<code>==, !=</code>	Is-equal-to and is-not-equal-to operators
<code>&lt;, &gt;, &lt;=,</code>	<code>&lt;, &gt;, &lt;=,</code>	Relational less-than, greater-than, less-than-or-equal-to, and greater-than-or-equal-to operators
<code>&gt;=</code>	<code>&gt;=</code>	
<code>x?y:z</code>	<code>x?y:z</code>	Conditional operator
<code>=</code>	<code>=</code>	Assignment operator

# Array

---

## ANALOGIE

- Ogni array è un oggetto (reference type), con lunghezza fissa
  - Sintassi simile al C/C++ per accedere ai valori, con controllo validità dell'indice a runtime.
- Stessa sintassi per allocare un array, con possibilità di specificare l'elenco dei valori (array initializer)
  - se si specifica solo la dimensione, c'è un valore di default predefinito per tutti gli elementi (0, false, null).
- Si possono creare **jagged array** (ovvero, array di array)
- Classe contenente i metodi di utilità per gli array (es. sort, ricerca binaria, copia elementi, ...)
  - `Array` in C#, `Arrays` in Java.

## DIFFERENZE

- C# supporta array multidimensionali, oltre agli array jagged, mediante la sintassi `[ , ]`
  - Es. `int[,] matrix = new int[5,8];`
- In C# qualsiasi classe può definire un «indexer» per utilizzare la stessa sintassi degli array per l'accesso a suoi elementi

# Array – Esempi



```
int[] v = new int[] { 5, 4, 3, 2, 1 };
for (int i = 0; i < v.length; i++) {
    System.out.println(v[i]);
}

String[] nomi = new String[5];
nomi[0] = "James Gosling";

float[][] mat = new float[][][]{
    new float[] {1,2,3}, new float[] {5,8,13}};
mat[1][0] = 4.2f;

int rows = mat.length; // 2 (numero righe)
int cols = mat[0].length; // 3 (numero colonne)

int[][] jagged = new int[][]{
    new int[5], new int[2], new int[3]};
jagged[0][4] = 5;

int[] v2 = v; // N.B. non crea una copia
int[] v3 = (int[])v.clone(); // Crea una copia

Arrays.sort(v); // 1, 2, 3, 4, 5
System.out.println(Arrays.toString(v));
int j = Arrays.binarySearch(v, 2); // 1
```



```
int[] v = new int[] { 5, 4, 3, 2, 1 };
for (int i = 0; i < v.Length; i++)
{
    Console.WriteLine(v[i]);
}

string[] nomi = new string[5];
nomi[0] = "Anders Hejlsberg";

float[,] mat = new float[,]{ {1,2,3}, {5,8,13} };
mat[1, 0] = 4.2f;

int r = mat.Rank; // 2 (numero di dimensioni)
int tot = mat.Length; // 6 (totale elementi)
int rows = mat.GetLength(0); // 2 (numero righe)
int cols = mat.GetLength(1); // 3 (numero colonne)

int[][] jagged = new int[3][]{
    new int[5], new int[2], new int[3]};
jagged[0][4] = 5;

int[] v2 = v; // N.B. non crea una copia
int[] v3 = (int[])v.Clone(); // Crea una copia

Array.Sort(v); // 1, 2, 3, 4, 5
Console.WriteLine("[{0}]", string.Join(", ", v));
int j = Array.BinarySearch(v, 2); // 1
Array.Reverse(v); // 1, 2, 3, 4, 5
```

# Variabili locali e array implicitamente tipizzati

- Le variabili locali possono avere un tipo *dedotto al momento dell'inizializzazione* anziché un tipo esplicito (**type inference**)
  - Introdotta dalla versione 3.0 del linguaggio
- Si utilizza la parola chiave **var**
  - Si indica al compilatore di inferire il tipo della variabile dall'espressione sul lato destro dell'istruzione di inizializzazione
  - Nota: Il tipo è determinato a **compile-time**, non a run-time...
- Questo meccanismo consente al linguaggio di aderire meglio al principio DRY («Don't Repeat Yourself»)
- N.B.: Java supporta la medesima funzionalità dalla versione 10

```
int i = 123;
string s = "Hello world";
double d = 10.3;
UnaClasse r = new UnaClasse();
int[] v = new int[] {1, 2, 3};
```

```
var i = 123;
var s = "Hello world";
var d = 10.3;
var r = new UnaClasse();
var v = new int[] {1, 2, 3};
```

# Classi

---

## ANALOGIE

- In generale sintassi identica o molto simile
- Controllo accesso: «**public**» e «**private**» hanno lo stesso significato («**protected**» no!, vedi sotto).
- Possibilità di definire campi e metodi «**static**»
- Possibilità di richiamare un costruttore da un altro costruttore (con sintassi leggermente diversa)

## DIFFERENZE

- In C# esiste il modificatore «**internal**» che rende l'elemento accessibile solo all'interno dell'assembly in cui è definito
- In Java «**protected**» consente l'accesso sia da classi derivate che da quelle nello stesso package, in C# solo da classi derivate.
  - In C# esiste «**protected internal**», cioè «**internal**» dentro l'assembly, «**protected**» fuori da esso
- La visibilità di default (se non si specifica un modificatore) in Java è «tutto il package».
  - In C# invece l'accesso di default è «**private**».
- Attenzione alla keyword «**static**» applicata a una classe
  - in C# indica che una classe ha solo membri «**static**»

# Classi – Esempio

```
class Counter
{
    // Il campo è reso inaccessibile direttamente
    private int value;

    // E' il costruttore che inizializza i campi!
    public Counter()
    {
        this(0); // Riusa l'altro costruttore
    }

    // Un costruttore con parametro
    public Counter(int value)
    {
        this.value = value;
    }

    // Unico modo per osservare lo stato
    public int getValue()
    {
        return this.value;
    }

    // Unico modo per modificare lo stato
    public void inc()
    {
        this.value++;
    }
}
```



```
class Counter
{
    // Il campo è reso inaccessibile direttamente
    private int value;

    // E' il costruttore che inizializza i campi!
    public Counter() : this(0) // Riusa l'altro costruttore
    {
    }

    // Un costruttore con parametro
    public Counter(int value)
    {
        this.value = value;
    }

    // Unico modo per osservare lo stato
    // N.B. in C# sarebbe più opportuna una proprietà
    public int GetValue()
    {
        return this.value;
    }

    // Unico modo per modificare lo stato
    public void Inc()
    {
        this.value++;
    }
}
```



# Passaggio dei parametri

## ANALOGIE

- Passaggio per valore: identico.
- Metodi con numero variabile di argomenti
  - visti come un parametro di tipo array all'interno del metodo
  - «void m(int... args)» in Java è equivalente a «**void m(params int[] args)**» in C#

## DIFFERENZE

- C#: supporta anche il passaggio esplicito per riferimento (mediante le parole chiave «**ref**» oppure «**out**»)
  - I parametri contrassegnati con «**out**» devono essere assegnati obbligatoriamente prima del **return** del metodo.
- Il C# consente di definire parametri opzionali (con un valore di default) e di identificare esplicitamente gli argomenti per nome al momento della chiamata.
- In Java è possibile dichiarare un parametro come «**final**», per indicare che il suo valore non sarà modificato dal metodo: non è prevista una cosa analoga in C#

# Passaggio dei parametri – Alcuni Esempi

```
partial class Program
{
    static void M1(int n)
    {
        n++;
        Console.WriteLine("N inside m1 = " + n);
    }

    static void M2(ref int n)
    {
        n++;
        Console.WriteLine("N inside m2 = " + n);
    }

    static void M3(out int a)
    {
        a = 10;
        Console.WriteLine("P inside M3 = " + a);
    }

    static void M4(params int[] v)
    {
    }

    static int Sum(int x, int y = 0)
    {
        return x + y;
    }
}
```

```
partial class Program
{
    public static void Main()
    {
        int n = 0;
        Console.WriteLine("N inside Main() = " + n);
        Console.WriteLine("Calling M1()");
        M1(n);
        Console.WriteLine("N inside Main() = " + n);
        Console.WriteLine("Calling M2()");
        M2(ref n);
        Console.WriteLine("N inside Main() = " + n);

        int p;
        Console.WriteLine("Calling M3()");
        M3(out p);
        Console.WriteLine("P inside Main() = " + p);

        M4(1, 2, 3);

        int s1 = Sum(1, 2);
        int s2 = Sum(5);
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
N inside Main() = 0
Calling M1()
N inside m1 = 1
N inside Main() = 0
Calling M2()
N inside m2 = 1
N inside Main() = 1
Calling M3()
P inside M3 = 10
P inside Main() = 10
```

# Tipi valore (struct)

- Simili alle classi ma con importanti differenze
  - Sono tipi valore: le loro variabili non sono riferimenti ma contengono direttamente il loro valore!
  - Non hanno ereditarietà
  - Possono contenere metodi e proprietà e costruttori
  - I campi non possono essere inizializzati al momento della dichiarazione
- Ideali per oggetti “leggeri”
  - Sono allocate nello **stack** (a differenza degli oggetti che sono sempre allocati nello heap) e non richiedono GC

```
struct Vector
{
    private double vx, vy;

    public Vector(double vx, double vy)
    {
        this.vx = vx;
        this.vy = vy;
    }

    public double CalculateNorm()
    {
        return Math.Sqrt(vx*vx + vy*vy);
    }
}
```

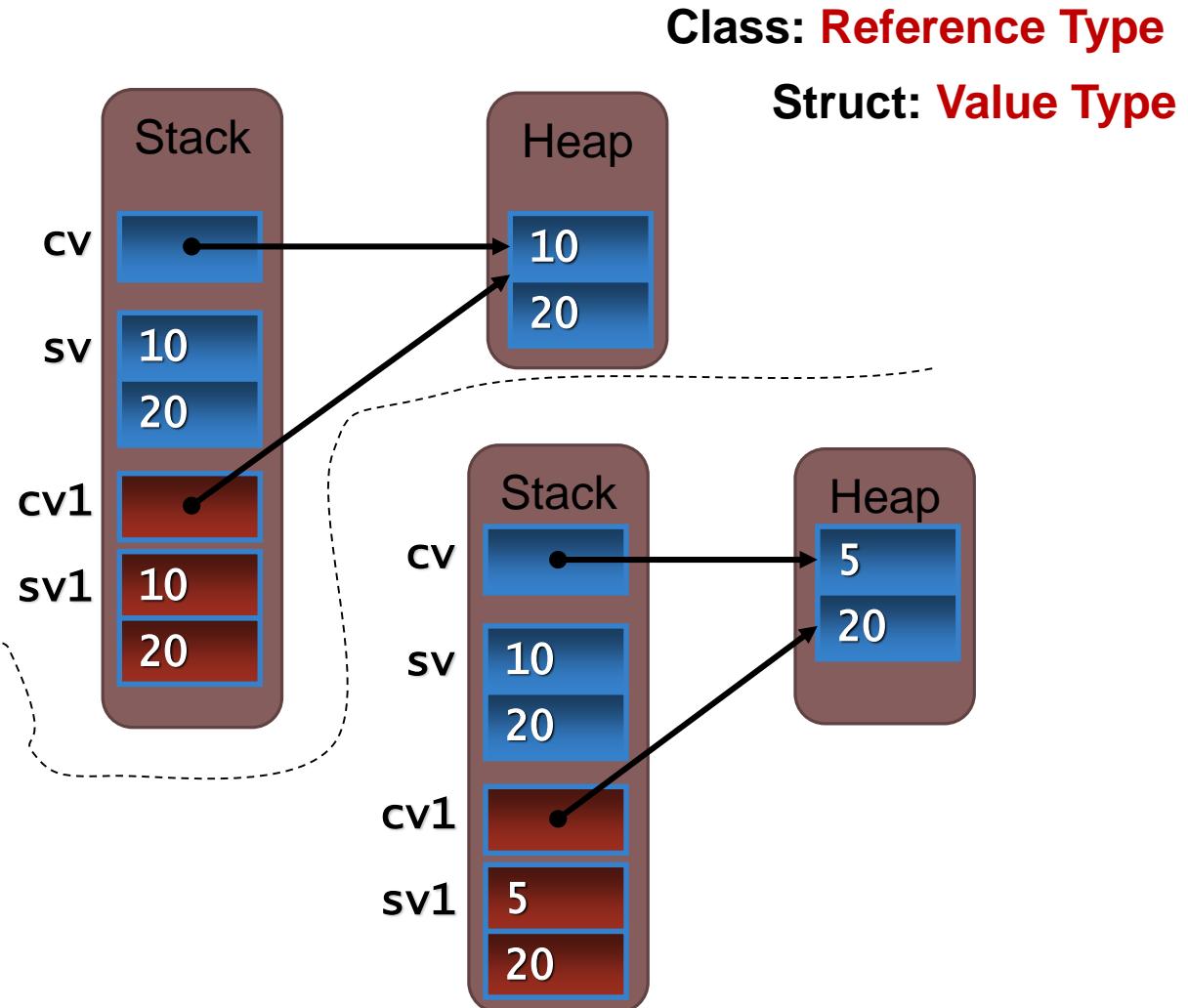
```
Vector P; //Alloca la struttura senza new: i campi
          //sono inizializzati ai valori di default

Vector P1 = new Vector(2,10);
           // Alloca la struttura chiamando un
           //costruttore con parametri

double n = P1.CalculateNorm();
```

# Struct vs. Class

```
class CVector  
{ public double vx, vy; ... }  
  
struct SVector  
{ public double vx, vy; ... }  
  
CVector cv = new CVector(10,20);  
  
SVector sv = new SVector(10,20);  
  
CVector cv1 = cv;  
  
SVector sv1 = sv;  
  
cv1.vx = 5;  
sv1.vx = 5;  
  
Console.WriteLine(sv.vx); // 10  
Console.WriteLine(cv.vx); // 5!
```



## PARTE 2

### Proprietà e Object Initializers

# Proprietà

Coniugano la semplicità dei campi con la flessibilità dei metodi

- Vi si accede come se fossero campi (es. `if (counter.Value > 42)`) ma in realtà sono metodi
  - **get**: chiamato quando viene letto il valore della proprietà.
  - **set**: chiamato quando viene assegnato un nuovo valore alla proprietà (la keyword «**value**» contiene il valore); se non esiste, la proprietà è read-only.
- Nei casi in cui in Java si implementano metodi *getter/setter*, in C# di solito si usano le *proprietà*.

```
public class Counter
{
    protected int value;
    ...
    public int GetValue()
    {
        return this.value;
    }
}
```



```
public class Counter
{
    protected int value;
    ...
    public int Value
    {
        get { return this.value; }
    }
}
```

## ESEMPIO

```
class Person
{
    private int age = 0;

    public int Age
    {
        get { return age; }
        set { age = value; }
    }
}

Person p = new Person();

// Esempio di chiamata set
p.Age = 42;

// Esempio di chiamata get
int x = p.Age;

// Esempio di chiamata get e set
p.Age += 1;
```

# Proprietà - Esempio con Dato Derivato

```
class Person
{
    private string name, surname;
    private DateTime dateOfBirth;

    public int Age // Proprietà di tipo int (read-only)
    {
        get
        {
            DateTime now = DateTime.Now;
            DateTime dn = dateOfBirth;

            int age = now.Year - dn.Year;

            if (now.Month < dn.Month || (now.Month == dn.Month && now.Day < dn.Day))
            {
                age--;
            }

            return age;
        }
    ...
}
```

```
static void Main()
{
    DateTime dob = ...
    Person p1 = new Person("Luca", "Bianchi", dob);
    Console.WriteLine(p1.Age);
}
```

# Proprietà automatiche

- Proprietà auto-implementate
  - Disponibili dalla versione 3.0 del linguaggio
- Permettono di rendere la dichiarazione delle proprietà più semplice e concisa.
- Utili per proprietà che semplicemente permettono l'accesso a un membro privato, senza implementare logiche specifiche nei metodi di accesso (get/set).
  - Il compilatore genera automaticamente un campo privato collegato alla proprietà

Codice Originale

```
class Persona
{
    public string Nome { get; set; }
    public string Cognome { get; set; }
    public DateTime DataDiNascita { get; private set; }
}
```



Codice «generato» dal Compilatore

```
class Persona
{
    private string nome;
    private string cognome;
    private DateTime dataDiNascita;

    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }

    public string Cognome
    {
        get { return cognome; }
        set { cognome = value; }
    }

    public DateTime DataDiNascita
    {
        get { return dataDiNascita; }
    }
}
```

# ESEMPIO 1-Properties

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** LezioneCSharp
- Toolbars:** Server Explorer, Toolbox
- Menu Bar:** FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEST, ANALYZE, TOOLS, EXTENSIONS, WINDOW, HELP
- Search Bar:** Search Visual Studio (Ctrl+Q)
- Code Editor:** MainProperties.cs (highlighted) and Person.cs (background). The MainProperties.cs code is as follows:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Properties
8  {
9      class MainProperties
10     {
11         static void Main(string[] args)
12         {
13             Person mario = new Person("Mario", "Rossi", new DateTime(1994, 1, 4));
14
15             Console.WriteLine(mario);
16             Console.ReadKey();
17
18             //giovanni.Age = 25;
19
20             //Console.WriteLine(mario);
21             //Console.ReadKey();
22
23             //Person luigi = new Person() {
24             //    FullName = "Luigi Bianchi",
25             //    BirthDate = new DateTime(1992, 1, 4)
26             //};
27
28             //Console.WriteLine(luigi);
29             //Console.ReadKey();
30         }
31     }
32 }
33
```

- Solution Explorer:** Shows the solution structure for 'LezioneCSharp' with 10 of 10 projects. The '1-Properties' project is selected, showing its contents: Properties, References, App.config, MainProperties.cs, Person.cs, 2-Indexers, 3-OperatorsOverloading, 4-ExtensionMethods, 5-DelegatesAndEvents, 6-NullablesAndOperators, 7-DynamicDispatch, 8-Enumerables, and 9-Iterators.
- Properties Explorer:** Shows the project properties for '1-Properties'.
- Task List:** Shows 'This item does not support previewing'.
- Status Bar:** c-sharp-lesson master

# Object Initializers

- Permettono di assegnare un valore a proprietà e campi pubblici di una classe al momento della creazione di una sua istanza
- Senza che vi sia un costruttore che esplicitamente accetta tali parametri

```
class Persona
{
    public string Nome { get; set; }
    public string Cognome { get; set; }
}
```

```
var p1 = new Persona();
p1.Nome = "Pluto";

var p2 = new Persona();
p2.Nome = "Mario";
p2.Cognome = "Rossi";
```

```
var p1 = new Persona() {
    Nome = "Pluto"
};

var p2 = new Persona() {
    Nome = "Mario",
    Cognome = "Rossi"
};
```



- Possono essere «emulati» in Java combinando: *Classi Anonime + Initializzatori*
- Ma funziona solo se:
  - Ci si riferisce a campi/metodi protetti o pubblici
  - La classe NON è **final**

```
public class Persona {
    private String nome;
    private String cognome;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setCognome(String cognome) {
        this.cognome = cognome;
    }
}
```

```
Persona p1 = new Persona() {
    setNome("Mario");
    SetCognome("Rossi")
};
```



# Collection Initializers

- Permettono di specificare gli elementi iniziali di una struttura dati
- Il compilatore chiama il metodo Add() per ciascun elemento.

```
var nums = new List<int>();  
nums.Add(0);  
nums.Add(1);  
nums.Add(2);  
nums.Add(3);
```



```
var nums = new List<int> { 0,1,2,3 };
```

- Disponibili a partire da Java 9 i seguenti metodi statici:
  - static <E> List<E> List.of()
  - static <E> List<E> List.of(E... elements)
  - static <E> Set<E> Set.of()
  - static <E> Set<E> Set.of(E... elements)
  - static <K, V> Map<K, V> Map.of()
  - static <K, V> Map<K, V> Map.of(K key, V val)
  - static <K, V> Map<K, V> Map.of(K k1, V v1,  
 K k2, V v2)
  - ...
- Restituiscono collezioni immutabili!



---

# PARTE 3

## Ereditarietà, Polimorfismo, Interfacce, Eccezioni, Dynamic Binding

# Ereditarietà

---

## ANALOGIE

- Eredità singola
  - riferimento a classe padre: «**super**» in Java, «**base**» in C#
- Polimorfismo con late binding
- Classi/metodi virtuali non estendibili/ridefinibili: «**final**» in Java, «**sealed**» in C#

## DIFFERENZE

- In Java tutti i metodi sono virtuali, mentre in C# (come in C++) sono virtuali solo i metodi definiti con la keyword «**virtual**».
- In C# nel ridefinire un metodo virtuale è necessario usare la keyword «**override**»
  - Analogie con l'annotazione Java `@Override`
- In C# è possibile ridefinire anche metodi non virtuali (con tutti i limiti del caso) tramite la keyword «**new**».
- In C# non si può cambiare visibilità nel ridefinire un metodo virtuale (es. da «**protected**» a «**public**»).

# Ereditarietà – Esempio

```
public class ExtendibleCounter{
    protected int value;

    public ExtendibleCounter(int initialValue){
        this.value = initialValue;
    }

    public void increment() {
        this.value++;
    }

    public int getValue() {
        return this.value;
    }
}

public class LimitCounter extends ExtendibleCounter{
    protected int limit;

    public LimitCounter(int initialValue,int limit){
        super(initialValue);
        this.limit = limit;
    }

    public boolean isOver() {
        return value==limit;
    }

    // Overriding del metodo increment()
    public void increment(){
        if (!this.isOver())
            super.increment(); // richiamo versione padre
    }
}
```



```
public class ExtendibleCounter
{
    protected int value;

    public ExtendibleCounter(int initialValue)
    {
        this.value = initialValue;
    }

    public virtual void Increment()
    {
        this.value++;
    }

    public int GetValue() {
        return this.value;
    }
}

public class LimitCounter : ExtendibleCounter
{
    protected int limit;

    public LimitCounter(int initialValue,int limit)
        : base(initialValue)
    {
        this.limit = limit;
    }

    public bool IsOver()
    {
        return value == limit;
    }

    // Overriding del metodo increment()
    public override void Increment()
    {
        if (!this.IsOver())
            base.Increment(); // richiamo versione base
    }
}
```



# Polimorfismo e metodi virtuali – Esempi

```
class Persona
{
    ...
    public void StampaDesc()
    {
        Console.WriteLine(
            "{0} {1}, anni {2}",
            Nome, Cognome, Età);
    }
}

class Studente : Persona
{
    ...
    public new void StampaDesc()
    {
        base.StampaDesc();
        Console.WriteLine(
            "Mat: {0}, Isc: {1}",
            Matricola, AnnoIscrizione);
    }
}
```

```
Persona A = new Persona("a", "A");
Studente B = new Studente("b", "B");
Persona C = B;
A.StampaDesc(); // Persona.StampaDesc()
B.StampaDesc(); // Studente.StampaDesc()
C.StampaDesc(); // Persona.StampaDesc()
```

```
class Persona
{
    ...
    public virtual void StampaDesc()
    {
        Console.WriteLine(
            "{0} {1}, anni {2}",
            Nome, Cognome, Età);
    }
}

class Studente : Persona
{
    ...
    public override void StampaDesc()
    {
        base.StampaDesc();
        Console.WriteLine(
            "Mat: {0}, Isc: {1}",
            Matricola, AnnoIscrizione);
    }
}
```

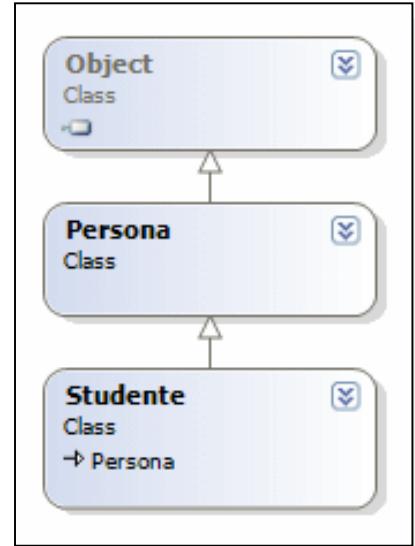
```
Persona A = new Persona("a", "A");
Studente B = new Studente("b", "B");
Persona C = B;
A.StampaDesc(); // Persona.StampaDesc()
B.StampaDesc(); // Studente.StampaDesc()
C.StampaDesc(); // Studente.StampaDesc()
```

# Polimorfismo (approfondimento)

L'ereditarietà fa sì che un oggetto possa appartenere a più di un tipo (la sua classe e la classe base)

## Conversione

- **Type cast** (sintassi simile al C)
  - Se la conversione non può essere eseguita → errore (eccezione)
- **Operatore as**
  - Se la conversione non può essere eseguita il risultato è **null**



## Controllo

- **Operatore is**
  - Permette di controllare se un oggetto appartiene a un determinato tipo

```
Persona A = new Persona("a", "A");
Studente B = new Studente("b", "B");
Persona C = B; // Corretto: non richiede cast
Studente D = (Studente)C; // Corretto ma richiede cast
Studente E = (Studente)A; // Genera errore a run-time
Studente F = C as Studente; // Corretto: F=C
Studente G = A as Studente; // Corretto: G=null

bool b1 = (A is Persona) && (B is Studente); // true
bool b2 = F is Studente; // true
bool b3 = A is Studente; // false
bool b4 = G is Studente; // false
bool b5 = B is Object; // true
```

# Classi Astratte

---

- È possibile dichiarare una classe come **abstract**
  - Classi astratte non possono essere istanziate (non si può creare un oggetto di tali classi)
  - Sono tipicamente utilizzate per fornire funzionalità comuni a una serie di classi derivate
  - Contengono tipicamente uno o più metodi **abstract** (ovvero metodi virtuali senza implementazione)

```
abstract class FormaGeometricaPiana
{
    public abstract int CalcolaPerimetro();
}

class Quadrato : FormaGeometricaPiana
{
    public override int CalcolaPerimetro()
    {
        return 0;
    }
}
```

# Interfacce

---

## ANALOGIE

- Stesso significato, sintassi e utilizzo simile.
  - Al posto della keyword «**implements**» (utilizzata in Java) si utilizza (in C#) la stessa notazione («**:**») utilizzata per l'ereditarietà
- Una classe può implementare più interfacce.
- I membri delle interfacce sono implicitamente «public»

## DIFFERENZE

- In C#, per convenzione, i nomi delle interfacce hanno il prefisso «**I**»
  - questo facilita la distinzione fra classe base e interfacce implementate dato che la sintassi è la stessa
  - In Java è possibile dichiarare costanti, classi innestate e metodi statici all'interno di un'interfaccia, in C# no
- In C# un'interfaccia, oltre a metodi, può contenere proprietà

# Interfacce (2)

```
public interface Device {  
    void switchOn();  
  
    void switchOff();  
  
    boolean isSwitchedOn();  
}  
  
public class Lamp implements Device {  
    public Lamp() { ... }  
  
    public void switchOn() {  
        this.switchedOn = true;  
    }  
  
    public void switchOff() {  
        this.switchedOn = false;  
    }  
  
    public boolean isSwitchedOn() {  
        return this.switchedOn;  
    }  
}
```



```
public interface IDevice  
{  
    void SwitchOn();  
    void SwitchOff();  
    bool IsSwitchedOn();  
}  
  
public class Lamp : IDevice  
{  
    public Lamp() { ... }  
    public void SwitchOn()  
    {  
        this.switchedOn = true;  
    }  
    public void SwitchOff()  
    {  
        this.switchedOn = false;  
    }  
    public bool IsSwitchedOn()  
    {  
        return this.switchedOn;  
    }  
}
```



# Eccezioni

## ANALOGIE

- Stessa sintassi:
  - «**try**»...«**catch**»...«**finally**», «**throw**»
- Gerarchia di ereditarietà delle eccezioni: derivano tutte da una classe «`System.Exception`»

## DIFFERENZE

- In C# non sono previste le «checked exceptions»
  - non vi è quindi un analogo della keyword «**throws**»
- In C# non sono ammesse istruzione di control-passing nel blocco **finally** (es. **break** o **return**).

```
try
{
    //Statements which may throw an exception
    throw new Exception();
}
catch (Exception e)
{
    //Exception handling
}
finally
{
    //Clean-up code
}
```

# Eccezioni – Esempio

```
class DeviceIsOverException extends Exception{ }

...
// ...AbstractDevice
public void on() throws DeviceIsOverException{
    if (!this.on) {
        throw new DeviceIsOverException(this);
    }
}

// ...DeviceRow
public void allOn() throws DeviceIsOverException{
    for (Device d : this.list){
        try {
            d.on();
        } catch (DeviceIsOverException de) {
            // do some stuff!
        }
    }
}
```



```
class DeviceIsOverException : Exception {}

...
// ...AbstractDevice
public void on()
{
    if (!this.on)
    {
        throw new DeviceIsOverException(this);
    }
}

// ...DeviceRow
public void allOn()
{
    foreach (IDevice d in this.list)
    {
        try
        {
            d.on();
        } catch (DeviceIsOverException de)
        {
            // do some stuff!
        }
    }
}
```



# Dynamic late binding

## Tipo di dato «**dynamic**»

- Il compilatore assume che variabili di tale tipo supportino qualsiasi operazione (metodo, proprietà, operatore, ...)
- A compile-time viene codificata opportunamente l'operazione che deve essere eseguita sull'oggetto e i relativi argomenti; a runtime se l'operazione non è possibile viene generata un'eccezione.
- Consente di semplificare l'utilizzo di oggetti di tipo sconosciuto come quelli ottenuti da linguaggi dinamici o mediante reflection.

```
string str = "Prova dynamic"; // string è un normale tipo non dynamic
Console.WriteLine(str.ToUpper()); // il compilatore conosce questo metodo
// Console.WriteLine(str.MetodoInesistente()); // il compilatore qui darebbe errore

dynamic dyn = str; // dyn è una variabile di tipo dynamic
Console.WriteLine(dyn.ToUpper()); // compilabile e a runtime chiamerà string.ToUpper()
Console.WriteLine(dyn.MetodoInesistente()); // compilabile, ma a runtime darà eccezione
```

# Esempio: Dynamic Dispatch

```
class A
{
    public virtual string GetValue()
    {
        return "a";
    }
}
```

```
class B : A
{
    public override string GetValue()
    {
        return "b";
    }
}
```

```
public static void Print(A x)
{
    Console.WriteLine("A: " + x.GetValue());
}

public void Print(B x)
{
    Console.WriteLine("B: " + x.GetValue());
}
```

```
A b1 = new B();
dynamic b2 = new B();

Print(b1); // ?
Print(b2); // ?
```

# ESEMPIO 2-Dynamic Dispatch

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** FILE EDIT VIEW PROJECT BUILD DEBUG TEST ANALYZE TOOLS EXTENSIONS WINDOW HELP Search Visual Studio (Ctrl+Q) LezioneCSharp
- Solution Explorer:** Shows a solution named "LezioneCSharp" containing 10 projects: 0-Miscellaneous, 1-Properties, 2-DynamicDispatch, 3-Indexers, 4-OperatorsOverloading, 5-ExtensionMethods, 6-DelegatesAndEvents, 7-NullablesAndOperators, 8-Enumerables, and 9-Iterators.
- Toolbox:** Standard .NET development tools.
- Code Editor:** The file "MainDynamicDispatch.cs" is open, displaying C# code for dynamic dispatch. The code defines a namespace "LezioneCSharp" with classes A and B, and a static class MainDynamicDispatch with methods Print and Main.

```
1  using System;
2
3  namespace LezioneCSharp
4  {
5      class A
6      {
7          public virtual string GetValue()
8          {
9              return "a";
10         }
11     }
12
13     class B : A
14     {
15         public override string GetValue()
16         {
17             return "b";
18         }
19     }
20
21     static class MainDynamicDispatch
22     {
23         static void Print(A x)
24         {
25             Console.WriteLine("A: " + x.GetValue());
26         }
27
28         static void Print(B x)
29         {
30             Console.WriteLine("B: " + x.GetValue());
31         }
32
33         static void Main(string[] args)
34     }
35 }
```

- Status Bar:** Ready, Ln 4, Col 2, Ch 2, INS, ^ 0, # 56, c-sharp-lesson, master

# PARTE 4

## Indicizzatori, Overloading di Operatori, Metodi Estensione

# Indexers

- Un indexer permette di accedere a attributi interni di un oggetto come se fosse un array
  - Dichiarazione simile a quella delle proprietà (**get/set**)
  - Si utilizza la keyword **this** come nome della proprietà
  - È possibile utilizzare indexer mono- o multi-dimensionalni (come per gli array)
  - Gli indici possono avere tipo qualsiasi, non solo **int**

```
public class DomusController
{
    private IDevice[] devices;
    ...

    public IDevice GetDevice(int pos)
    {
        return this.devices[pos];
    }

    public void SetDevice(int pos, IDevice dev)
    {
        this.devices[pos] = dev;
    }
}

var dc = new DomusController(10);
dc.SetDevice(0, new Lamp());
dc.SetDevice(1, new TV());
dc.SetDevice(2, new Radio());
dc.GetDevice(1).SwitchOn();
```



Versione  
senza Indexer

```
public class DomusController
{
    private IDevice[] devices;
    ...

    public IDevice this[int pos]
    {
        get { return devices[pos]; }
        set { devices[pos] = value; }
    }
}
```

```
var dc = new DomusController(10);
dc[0] = new Lamp();
dc[1] = new TV();
dc[2] = new Radio();

dc[1].SwitchOn();
```

Versione  
con Indexer

# ESEMPIO

## 3-Indexers

The screenshot shows the Visual Studio IDE interface with the following details:

- Menu Bar:** FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEST, ANALYZE, TOOLS, EXTENSIONS, WINDOW, HELP.
- Search Bar:** Search Visual Studio (Ctrl+Q).
- Project Name:** LezioneCSharp.
- Solution Explorer:** Shows the solution structure with projects like 0-Miscellaneous, 1-Properties, 2-DynamicDispatch, 3-Indexers, 4-OperatorsOverloading, 5-ExtensionMethods, 6-DelegatesAndEvents, 7-NullablesAndOperators, 8-Enumerables, and 9-Iterators.
- Code Editor:** Displays the `Matrix.cs` file content. The code defines a `Matrix< TValue >` class with properties `Col` and `Row`, and an indexer `this[i, j]` that returns or sets the value at position `(i, j)`. It also overrides the `ToString` method to return a string representation of the matrix.
- Status Bar:** Shows the current file is `Matrix.cs`, line 1, column 1, character 1, and the status `INS`.

```
1  using System;
2  using System.Collections;
3  using System.Linq;
4
5  namespace Indexer
6  {
7      1 reference | 0 changes | 0 authors, 0 changes
8      class Matrix< TValue >
9      {
10          private TValue[][] _values;
11
12          0 references | 0 changes | 0 authors, 0 changes
13          public Matrix(int nRows, int nCols)
14          {
15              this.Cols = nCols;
16              this.Rows = nRows;
17              this._values = new TValue[nRows][];
18              for (int i = 0; i < nRows; i++)
19              {
19.5      this._values[i] = new TValue[nCols];
20              }
21
22              1 reference | 0 changes | 0 authors, 0 changes
23              public int Cols { get; }
24              1 reference | 0 changes | 0 authors, 0 changes
25              public int Rows { get; }
26
27              0 references | 0 changes | 0 authors, 0 changes
28              public TValue this[int i, int j]
29              {
30                  get { return this._values[i][j]; }
31                  set { this._values[i][j] = value; }
32              }
33
34              10 references | 0 changes | 0 authors, 0 changes
35              public override string ToString()
36              {
37                  return string.Join(
38                      "\n",
39                      this._values.Select(row => $" { string.Join(" " , row) } " ) );
40              }
41
42          }
43
44      }
45
46  }
```

# Overloading degli operatori

- I nuovi tipi possono ridefinire i principali operatori
  - Esempi di operatori ridefinibili: +, -, !, ~, ++, --, \*, /, %, &, |, ^, <<, >>, ==, !=, <, >, <=, >=
- Gli operatori con assegnamento (es. +=) non possono essere ridefiniti direttamente
  - vengono valutati usando il corrispondente operatore binario (es. +), che può essere ridefinito.
- Per ridefinire un operatore è necessario definire un metodo «**static**» il cui nome è la keyword «**operator**» seguita dal simbolo dell'operatore.
  - Si possono anche definire metodi statici da chiamare per type-cast impliciti ed esplicativi (es. «**explicit operator double (...)**» definisce un type cast esplicito a double).

```
public class Point3D
{
    private double x, y, z;

    public Point3D(double x, double y, double z)
    {
        this.x = x; this.y = y; this.z = z;
    }

    public static Point3D operator+(Point3D p1, Point3D p2)
    {
        // Esempio di operatore binario ridefinito
        return new Point3D(p1.x+p2.x, p1.y+p2.y, p1.z+p2.z);
    }

    public static Point3D operator-(Point3D p)
    {
        // Esempio di operatore unario ridefinito
        return new Point3D(-p.x, -p.y, -p.z);
    }

    public override string ToString()
    {
        return string.Format("[{0}, {1}, {2}]", x, y, z);
    }
}

var p1 = new Point3D(1, 2, 0);
var p2 = -p1; // Utilizzo operatore unario
var p3 = p1 + p2; // Utilizzo operatore binario
p1 += p3; // Utilizzo operatore binario con assegnamento

Console.WriteLine("{0} {1}", p1, p3);
```

# ESEMPIO

## 4-Operators Overloading

The screenshot shows a Visual Studio interface with the following details:

- Solution Explorer:** Shows the solution "LezioneCSharp" with multiple projects: 0-Miscellaneous, 1-Properties, 2-DynamicDispatch, 3-Indexers, and 4-OperatorsOverloading (which is currently selected).
- Code Editor:** Displays the file "MainOperatorsOverloading.cs" containing C# code for demonstrating operator overloading.
- Output Window:** Shows the message "This item does not support previewing".
- Status Bar:** Shows "Ln 1 Col 1 Ch 1 INS ↑ 0 ↻ 67 c-sharp-lesson master".

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace OperatorsOverloading
8 {
9
10    class MainOperatorsOverloading
11    {
12        static void Main(string[] args)
13        {
14            StrangeInt x = 7;
15            StrangeInt y = 3;
16
17            Console.WriteLine((x + y).ToString()); // 10?
18            Console.WriteLine((x - y).ToString()); // 4?
19            Console.WriteLine((x * y).ToString()); // 21?
20            Console.WriteLine((x / y).ToString()); // 2?
21
22            Console.ReadKey();
23            Console.WriteLine();
24
25            int r1 = x + y;
26            int r2 = x - y;
27            int r3 = x * y;
28            int r4 = x / y;
29
30            Console.WriteLine(r1);
31            Console.WriteLine(r2);
32            Console.WriteLine(r3);
33            Console.WriteLine(r4);
34
35            Console.ReadKey();
36        }
37    }
38 }
```

# Metodi di estensione

- Gli *extension method* consentono di aggiungere metodi ai tipi esistenti, senza creare un nuovo tipo derivato
- Es. Si vuole aggiungere un metodo personalizzato invocabile su tutti gli oggetti di tipo string senza creare un oggetto «MyString» che estenda da string.

- Parola chiave «**this**» sul 1° argomento
  - ... di un metodo **statico** in una classe **statica**
  - Tale metodo **statico** può essere chiamato come se fosse un metodo **d'istanza** del tipo del primo parametro.

## Utilizzo Classico

```
string t = "Hello, World!";
int n1 = MyExtensions.CountOccurencies(t, 'l');
int n2 = MyExtensions.CountOccurencies("TESTO", 'T');
```

```
static class MyExtensions
{
    public static int CountOccurencies(this string text, char element)
    {
        int occurencies = 0;

        foreach (var e in text)
        {
            if (e == element)
            {
                occurencies++;
            }
        }

        return occurencies;
    }
}
```

## Utilizzo dei metodi di estensione

```
string t = "Hello";
int n1 = t.CountOccurencies('l');
int n2 = "TESTO".CountOccurencies('T');
```

# ESEMPIO

## 5-Extension Methods

The screenshot shows a Visual Studio interface with the following details:

- File Menu:** FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEST, ANALYZE, TOOLS, EXTENSIONS, WINDOW, HELP.
- Search Bar:** Search Visual Studio (Ctrl+Q).
- Solution Explorer:** Shows a solution named "LezioneCSharp" containing 10 projects: 0-Miscellaneous, 1-Properties, 2-DynamicDispatch, 3-Indexers, 4-OperatorsOverloading, 5-ExtensionMethods (selected), 6-DelegatesAndEvents, 7-NullablesAndOperators, 8-Enumerables, and 9-Iterators.
- Properties Window:** Shows "5-ExtensionMethods Project Properties" under "Misc".
- Code Editor:** Displays "Complex.cs" file with the following code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ExtensionMethods
8 {
9     sealed class Complex
10    {
11        public double Re { get; }
12        public double Im { get; }
13
14        public Complex(double re, double im)
15        {
16            this.Re = re;
17            this.Im = im;
18        }
19
20        public static Complex operator +(Complex x, Complex y)
21        {
22            return new Complex(x.Re + y.Re, x.Im + y.Im);
23        }
24
25        public static Complex operator -(Complex x, Complex y)
26        {
27            return new Complex(x.Re - y.Re, x.Im - y.Im);
28        }
29
30        public static Complex operator *(Complex x, Complex y)
31        {
32            return new Complex(x.Re * y.Re - x.Im * y.Im, x.Im * y.Re + x.Re * y.Im);
33        }
34    }
}
```

The code editor also shows a status bar with "Ln 1 Col 1 Ch 1 INS ↑ 0 ⌂ 67 c-sharp-lesson master".

---

## PARTE 5

### Generici, Collezioni, Delegates, Lambda, LINQ (cenni)

# Tipi generici

---

## ANALOGIE

- Polimorfismo parametrico con una sintassi piuttosto simile.
- Possibile vincolare un parametro di tipo a essere un sottotipo di un tipo più specifico di `Object`.
- Introdotti in C# dalla versione 2.0

## DIFFERENZE

- L'implementazione è piuttosto diversa
  - Java utilizza «type erasure» durante la compilazione (la JVM non sa nulla dei tipi generici e i parametri di tipo diventano tutti `Object`)
  - in C# sono compilati come appositi tipi generici e gestiti dal runtime «reification»: questo consente una maggior efficienza in particolare quando il parametro di tipo è un tipo valore (es. `int` o `float`).
- Java supporta le «wildcards» (es. `Vector<?>`), mentre in C# non sono disponibili
- In C# è possibile vincolare un parametro di tipo ad avere un costruttore pubblico senza parametri o a essere un tipo riferimento o un tipo valore.

# Tipi generici – Esempi

```
public class Pair<X, Y>{  
    private X first;  
    private Y second;  
  
    public Pair(X first, Y second){  
        this.first = first;  
        this.second = second;  
    }  
  
    public X getFirst() { return this.first; }  
  
    public Y getSecond() { return this.second; }  
  
    public String toString(){  
        return "<" + first + "," + second + ">";  
    }  
}
```

```
JAVA  
Vector<Pair<String, Integer>> v = new Vector<>();  
v.addElement(new Pair<>("Prova",1));  
v.addElement(new Pair<>("di",2));  
v.addElement(new Pair<>("Vettore",3));  
  
for (Pair<String, Integer> pair : v){  
    System.out.println(pair);  
}
```

```
public class Pair<X, Y>  
{  
    private X first;  
    private Y second;  
  
    public Pair(X first, Y second)  
    {  
        this.first = first;  
        this.second = second;  
    }  
  
    public X GetFirst() { return this.first; }  
  
    public Y GetSecond() { return this.second; }  
  
    public override string ToString()  
    {  
        return "<" + first + "," + second + ">";  
    }  
}
```

```
C#  
var v = new List<Pair<string, int>>();  
v.Add(new Pair<string, int>("Prova",1));  
v.Add(new Pair<string, int>("di",2));  
v.Add(new Pair<string, int>("Vettore",3));  
  
foreach (var pair in v)  
{  
    Console.WriteLine(pair);  
}
```

# Collezioni e iteratori

## ANALOGIE

- Ampia libreria di strutture dati e relativi algoritmi
  - namespace `System.Collections` in C# e package `java.util` in Java
- Interfacce e classi astratte per consentire di manipolare dati in modo uniforme indipendentemente dall'implementazione della specifica struttura dati
- Iterazione su qualsiasi oggetto la cui classe implementi un'apposita interfaccia
  - `IEnumerable<X>` in Java e `IEnumerable<X>` in C#

## DIFFERENZE

- In C# alle strutture dati non generiche basate su «object» sono state affiancate classi ad hoc per tipi generici (namespace `System.Collections.Generic`)
  - mentre in Java le strutture dati non-generiche esistenti sono state adattate all'utilizzo con i tipi generici.
- In C# è possibile implementare gli iteratori in modo molto semplice scrivendo un metodo che utilizza l'istruzione «**`yield return`**»
- In C# si può creare una collezione inizializzandone gli elementi in modo simile a quanto avviene per gli array

# Collezioni e iteratori – Esempio 1

```
List<Integer> nums = new ArrayList<>(  
    Arrays.asList(0,1,2,3,4,5,6,7,8,9));  
  
for (int i = nums.size() - 1; i >= 0; i--) {  
    if (nums.get(i) % 2 == 1){  
        nums.remove(i);  
    }  
}  
  
Map<Integer, String> map = new HashMap<>();  
for (int n : nums){  
    // Associa  $n^3$  (come testo) ad ogni numero n  
    map.put(n, Integer.toString(n*n*n));  
}  
  
Random rnd = new Random();  
for (int i = 0; i < 20; i++){  
    int n = rnd.nextInt(10);  
    String str = map.get(n);  
    if (str != null){  
        System.out.printf("%d ==> %s\n", n, str);  
    } else {  
        System.out.printf("%d non in cache.\n", n);  
    }  
}
```



```
var nums = new List<int> { 0,1,2,3,4,5,6,7,8,9 };  
  
for (var i = nums.Count - 1; i >= 0; i--)  
{  
    if (nums[i] % 2 == 1)  
    {  
        nums.RemoveAt(i);  
    }  
}  
  
var map = new Dictionary<int, string>();  
foreach (int n in nums)  
{ // Associa  $n^3$  (come testo) ad ogni numero n  
    map.Add(n, (n*n*n).ToString());  
}  
  
var rnd = new Random();  
for (int i = 0; i < 20; i++)  
{  
    var n = rnd.Next(10);  
    string str;  
    if (map.TryGetValue(n, out str))  
    {  
        Console.WriteLine("{0} ==> {1}", n, str);  
    }  
    else  
    {  
        Console.WriteLine("{0} non in cache.", n);  
    }  
}
```



# Collezioni e iteratori – Esempio 2

```
class Range implements Iterable<Integer>{
    final private int start;
    final private int stop;

    public Range(int start, int stop){
        this.start = start;
        this.stop = stop;
    }

    public java.util.Iterator<Integer> iterator(){
        return new RangeIterator(this.start, this.stop);
    }
}

class RangeIterator implements Iterator<Integer>{
    private int cur, stop;

    public RangeIterator(int start, int stop) {
        this.current = start;
        this.stop = stop;
    }

    public Integer next() { return this.cur++; }

    public boolean hasNext() { return cur <= stop; }

    public void remove() { }

    ...
    for (int i : new Range(5,12)){
        System.out.println(i); // 5 6 7 8 9 10 11 12
    }
}
```



```
public class Range : IEnumerable<int>
{
    readonly private int start;
    readonly private int stop;

    public Range(int start, int stop)
    {
        this.start = start;
        this.stop = stop;
    }

    public IEnumerator<int> GetEnumerator()
    {
        for (int cur = start; cur <= stop; cur++)
        {
            yield return cur;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    ...
    foreach (int i in new Range(5, 12))
    {
        Console.WriteLine(i); // 5 6 7 8 9 10 11 12
    }
}
```



# Delegates

- Simili ai puntatori a funzione del C/C++
  - Sono Orientati agli oggetti
  - Sono Type-safe
- Analoghi alle interfacce funzionali di Java >= 8
- Caratteristiche principali
  - Permettono di passare un metodo come parametro o di assegnarlo a una variabile
  - Una volta che ad una variabile di tipo delegate è stato assegnato un metodo, questa si comporta esattamente come tale metodo
- Sono alla base degli eventi C#

```
class Program
{
    delegate int Operation(int a, int b);

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

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

    static void Main()
    {
        Operation op = Add;

        int res1 = op(3, 5);

        op = Sub;

        int res2 = op(2, 4);
    }
}
```

# Delegates – Altro Esempio

```
delegate void ProgressUpdate(int perc);

class MyExecutor
{
    public void Execute(ProgressUpdate callBack)
    {
        for (int i = 0; i < MAX; i++)
        {
            // do some stuff

            if (i % 100 == 0)
            {
                callBack(i);
            }
        }
    }
}
```

```
void PrintExecutorProgressStatus(int perc)
{
    Console.Write("...{0} ", perc);
}

void IgnoreProgressStatus(int perc) { }

static void Main()
{
    MyExecutor e1 = new MyExecutor();
    e1.Execute(PrintExecutorProgressStatus);

    MyExecutor e2 = new MyExecutor();
    e2.Execute(IgnoreProgressStatus);
}
```

# Metodi anonimi (pre Lambda Expressions)

```
delegate void ProgressUpdate(int perc);

class MyExecutor
{
    public void Execute(ProgressUpdate callBack)
    {
        for (int i = 0; i < MAX; i++)
        {
            // do some stuff

            if (i % 100 == 0)
            {
                callBack(i);
            }
        }
    }
}
```

- Permettono di passare direttamente un “blocco di codice” a un parametro delegato (**closure-like functions**)
- Eliminano la necessità di dichiarare un metodo separato per poi poterlo passare al delegate
- La keyword **delegate** sostituisce il nome del metodo (che è appunto “anonimo”) ed è seguita dalla dichiarazione degli eventuali parametri del metodo

```
static void Main()
{
    MyExecutor e1 = new MyExecutor();
    e1.Execute(delegate(int p) {
        Console.WriteLine("...{0} ", perc);
    });

    MyExecutor e2 = new MyExecutor();
    e2.Execute(delegate(int p) { });
}
```

# Espressioni Lambda

- Funzioni anonime che contengono istruzioni C# e possono essere utilizzate per creare delegate anonimi
- Disponibili dalla versione 3.0 del linguaggio
  - Come estensione dei delegates
- Sintatticamente molto simili alle lambda di Java >= 8
  - Operativamente, sono del tutto analoghe a Java
- Contengono l'operatore =>
  - La parte a sinistra dell'operatore => indica i parametri della funzione
  - La parte a destra contiene un'espressione o un insieme di istruzioni racchiuso fra parentesi graffe.

```
//with parameters
n => n == 2
(a, b) => a + b
(a, b) => {a++; return a+b; }

//With explicitly typed
parameters
(int a, int b) => a + b

//No parameters
() => return 0;
```

# Espressioni Lambda – Esempi

```
delegate int TipoFunzione(int parametro);  
delegate int TipoFunzione2(int p1, int p2);
```

```
TipoFunzione f = x => x * x;  
int a = f(5); // a = 25  
  
TipoFunzione2 f2 = (x, y) => x + y;  
int b = f2(3, 7); // b = 10
```

```
TipoFunzione f = delegate(int x) {  
    return x * x;  
};  
  
int a = f(5); // a = 25  
  
TipoFunzione2 f2 = delegate(int x, int y) {  
    return x + y;  
};  
  
int b = f2(3, 7); // b = 10
```

```
e.Execute(p => Console.WriteLine("...{0} ", p));
```

```
e.Execute(delegate(int p) {  
    Console.WriteLine("...{0} ", p);  
});
```

# ESEMPIO

## 6-Delegates AndEvents

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** LezioneCSharp
- Toolbars:** Standard, Debug, Start, Task List.
- Code Editor:** Person.cs (highlighted) and MainDelegatesAndEvents.cs (background). The Person.cs code is as follows:

```
1 using System;
2
3 namespace Delegates
4 {
5     delegate void ValueEditAction<in TType>(TType value);
6
7     class Person
8     {
9         private string _name;
10        private string _surname;
11        private int _age;
12
13        public Person(string name, string surname, int age)
14        {
15            this._name = name;
16            this._surname = surname;
17            this._age = age;
18        }
19
20        public string Name
21        {
22            get { return this._name; }
23            set
24            {
25                this._name = value;
26                if (this.NameChanged != null)
27                {
28                    this.NameChanged(value);
29                }
30            }
31        }
32
33        public string Surname
34        {
35            get { return this._surname; }
36            set
37        }
38    }
39}
```

- Solution Explorer:** Shows a solution named 'LezioneCSharp' containing 10 projects. The '6-DelegatesAndEvents' project is selected.
- Properties:** Shows '6-DelegatesAndEvents' Project Properties.
- Task List:** Shows 'This item does not support previewing'.
- Status Bar:** Ln 1, Col 1, Ch 1, INS, ↑ 0, ↲ 67, c-sharp-lesson, master, master

# Nullable types

- Una variabile di tipo `Nullable<T>` può assumere tutti i valori di `T` oppure il valore `null` (es. `Nullable<bool>` può assumere tre valori: `true`, `false`, `null`).
  - `T` deve essere un value type
- La sintassi `T?` (es. `bool?`, `int?`, ...) equivale a `Nullable<T>` ed è preferibile perché più intuitiva e leggibile.
- Utili in tutti i casi in cui un value type può avere valore non-definito (caso tipico: quando si interagisce con un database).
- Metodi `GetValueOrDefault()`, proprietà `HasValue(bool)` e `Value(T)`.
- L'operatore `??` (null-coalescing) consente di specificare un valore di default da restituire nel caso in cui la variabile sia `null`.

```
int? v1 = null; // equivale a Nullable<int> v1 = null;
int? v2 = 42;   // equivale a Nullable<int> v2 = 42;

Console.WriteLine("V1: {0}", v1!=null ? v1.ToString() : "Sconosciuto"); // oppure ..., v1.HasValue ?
Console.WriteLine("V2: {0}", v2!=null ? v2.ToString() : "Sconosciuto");
Console.WriteLine(v1.GetValueOrDefault()); // 0 (default di int)
Console.WriteLine(v2.GetValueOrDefault()); // 42
Console.WriteLine(v1 ?? 3); // 3
Console.WriteLine(v2 ?? 3); // 42
Console.WriteLine((v1 + 3) == null); // True (null + 3 = null)
Console.WriteLine((v2 + 58) == null); // False (42 + 58 = 100)
```

# ESEMPIO

## 7-NullablesAnd Operators

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** LezioneCSharp
- Toolbars:** Standard, Debug, Start
- Solution Explorer:** Shows the solution structure for "LezioneCSharp" with 10 projects, including "7-NullablesAndOperators".
- Properties Explorer:** Shows the properties for the selected project.
- Task List:** Shows the task list for the current file.
- Output Window:** Shows the output "Ready".
- Code Editor:** Displays the C# code for "MainNullablesAndOperators.cs".

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace NullablesAndOperators
8  {
9      class A
10     {
11         public string SomeProperty { get; set; }
12     }
13
14
15     class MainNullablesAndOperators
16     {
17         static void Main(string[] args)
18         {
19             int? v1 = null; // equivale a Nullable<int> v1 = null;
20             Nullable<int> v2 = 42; // equivale a Nullable<int> v2 = 42;
21
22             Console.WriteLine("V1: {0}", v1 != null ? v1.ToString() : "Sconosciuto"); // oppure ..., v1.HasValue ? Console.WriteLine("V2: {0}", v2!=null ? v2.ToString() : "Sconosciuto");
23             Console.WriteLine(v1.GetValueOrDefault()); // 0 (default di int)
24             Console.WriteLine(v2.GetValueOrDefault()); // 42
25             Console.WriteLine(v1 ?? 3); // 3
26             Console.WriteLine(v2 ?? 3); // 42
27
28             Console.WriteLine((v1 + 3) == null); // True (null + 3 = null)
29             Console.WriteLine((v2 + 58) == null); // False (42 + 58 = 100)
30
31             Console.ReadLine();
32
33             A example = new A();
34             string v3 = example?.SomeProperty ?? "default string";
35             Console.WriteLine(v3);
36         }
37     }
38 }
```

# Tipi anonimi

---

- Creazione di oggetti con proprietà read-only senza dover esplicitamente definirne il tipo.
- Si utilizza «`new`» con un «object initializer»
- Possono contenere solo proprietà read-only, non altri membri
- Di solito in abbinamento con la keyword «`var`» (il nome del tipo non è noto al programmatore: è deciso internamente dal compilatore)
- Tipicamente utilizzati nella clausola «`select`» di un query LINQ (si veda oltre)

```
var a = new { Nome = "Rossi", Anno = 1980 };
Console.WriteLine(a.Nome);
Console.WriteLine(a.Anno);

var data = DateTime.Today;
// Se il nome di una proprietà del nuovo tipo
// è omesso, il compilatore usa il nome della
// proprietà che lo inizializza.
var b = new { Anno = data.Year, data.Month };
Console.WriteLine(b.Anno);
Console.WriteLine(b.Month);
```

# C# `IEnumerable` VS Java8 `Stream`

---

`IEnumerable` ≈ `Iterable + Stream`

C#    Java 8

## Classe `«System.Linq.Enumerable»`

- Fornisce molti *extension methods* che implementano molte funzioni *high-order* per manipolare flussi di dati (potenzialmente illimitati)
- Permette di costruire *pipeline* di elaborazione analogamente a `Collection.stream()` di Java 8
- Anche gli array implementano `IEnumerable`!
- Nomenclatura più vicina al mondo SQL che al mondo funzionale
  - `Enumerable.Select` ≈ `Stream::map`
  - `Enumerable.Aggregate` ≈ `Stream::reduce | Stream::collect`
  - `Enumerable.Where` ≈ `Stream::filter`
  - ...

# Language-Integrated Query – LINQ (1/2)

- Integrazione all'interno del C# di un linguaggio di interrogazione simile a SQL
  - Introdotto per semplificare la gestione delle collezioni di dati
- Basato su:
  - Variabili implicitamente tipizzate e inizializzatori di oggetti
  - Tipi generici, metodi di estensione e tipi anonimi
  - Espressioni lambda
  - Apposite keyword e sintassi per scrivere le query
- Cosa si può interrogare?
  - Database
  - Strutture dati in memoria, collections, array, ...
  - Documenti XML
  - Ogni altra fonte dati che implementi un «provider LINQ»

```
// 1. Fonte dati: array in memoria
int[] numbers = new int[7] {0,1,2,3,4,5,6};

// 2. Creazione della query
var numQuery = from num in numbers
               where (num % 2) == 0
               orderby num descending
               select num;

// 3. Esecuzione della query
foreach (int num in numQuery)
{
    Console.WriteLine(num);
}
```

```
// 1. Fonte dati: file xml
var xml = XElement.Load(@"C:\Temp\CD.xml");

// 2. Creazione della query
var xmlQuery = from cd in xml.Elements()
               where cd.Element("COUNTRY").Value == "UK"
               group cd by cd.Element("YEAR").Value into g
               orderby g.Key
               select new
               {
                   Anno = g.Key,
                   Numero = g.Count(),
               };

// 3. Esecuzione della query
foreach (var r in xmlQuery)
{
    Console.WriteLine("{0}: {1} CD", r.Anno, r.Numero);
}
```

# Language-Integrated Query – LINQ (2/2)

## Query operators

- Metodi che forniscono le funzionalità del LINQ (es. selezione, raggruppamento, ...)
- Implementati come metodi di estensione generici che agiscono su interfacce (`IEnumerable<T>` e `IQueryable<T>`)
- La sintassi «`from ... where ... select`» viene trasformata dal compilatore in chiamate a tali metodi
  - il programmatore può anche utilizzarli direttamente
- N.B. L'esecuzione delle operazioni che restituiscono una sequenza di elementi non avviene al momento della chiamata del metodo, ma è posticipata al momento in cui si enumera il risultato.

```
// sintassi «query expression»
var numQuery = from num in numbers
                where (num % 2) == 0
                orderby num descending
                select num;
```



```
// sintassi «method-based»
var numQuery = numbers
                .Where(num => (num % 2) == 0)
                .OrderByDescending(num => num)
                .Select(num => num);
```

# LINQ – Altro Esempio

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

for (var i = nums.Count - 1; i >= 0; i--)
{
    if (nums[i] % 2 == 1)
    {
        nums.RemoveAt(i);
    }
}

var map = new Dictionary<int, string>();

foreach (var n in nums)
{ // Associa n^3 (come testo) ad ogni numero n
    map.Add(n, (n*n*n).ToString());
}
```



```
var nums = Enumerable.Range(0, 10);
var map = nums.Where(n => n % 2 == 0)
              .ToDictionary(n => n, n => (n * n * n).ToString());
```

# ESEMPIO 8-Enumerables

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** Search Visual Studio (Ctrl+Q), LezioneCSharp
- Solution Explorer:** Solution 'LezioneCSharp' (10 of 10 projects) containing 0-Miscellaneous, 1-Properties, 2-DynamicDispatch, 3-Indexers, 4-OperatorsOverloading, 5-ExtensionMethods, 6-DelegatesAndEvents, 7-NullablesAndOperators, 8-Enumerables (selected), 9-Iterators, App.config, MainEnumerablesExamples.cs.
- Toolbox:** Standard .NET components.
- Code Editor:** File: MainEnumerablesExamples.cs (8-Enumerables)

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace EnumerableExamples
6  {
7      static readonly string[] books = new string[]
8      {
9          "VB .NET Tutorials for dummies",
10         "Learn C ++ the hard way",
11         "MVC tutorials for .Net developers",
12         "Java Official Reference",
13         "Java Stream Tutorial",
14         "#Tutorials -- The handbook",
15         "Python 3 - The ultimate guide",
16         "Why Python 2 won't die",
17         "IronPython tutorial",
18         "Effective Java"
19     };
20
21
22     static void Main(string[] args)
23     {
24
25         var bookInfo = books
26             .Where(title => title.Contains("tutorial"))
27             .OrderBy(title => title.Length)
28             .Select(title => new
29             {
30                 Title = title,
31                 Length = title.Length,
32                 NumOfWords = title.Split(' ').Length
33             });
34
35
36
37         foreach (var book in bookInfo)
38         {
39
40             Console.WriteLine("Press enter...");
41             Console.ReadKey();
42
43             foreach (var book in bookInfo)
44             {
45
46                 if (book.Title == bookInfo[0].Title)
47                     break;
48
49             }
50
51         }
52
53     }
54
55 }
```
- Status Bar:** Ready, Ln 1, Col 1, Ch 1, INS, ^ 0, 67, c-sharp-lesson, master

# Custom stream: Iteratori + metodi estensione

---

C# supporta gli iteratori a livello di linguaggio tramite la sintassi «**yield return**»

- Applicabile in metodi aventi come tipo di ritorno `IEnumerable<T>` o `IEnumerator<T>`
  - Si usa «**yield return**» al posto di «**return**»
- Lo stato del metodo «sopravvive» alla «**yield return**»
  - Il compilatore genera un o `IEnumerator<T>` dietro le quinte
- Usato in metodi estensione, permette di estendere a piacere l'insieme di operazioni high-order possibili sugli enumerabili
  - In Java, sarebbe come poter aggiungere metodi all'interfaccia Stream
- LINQ è implementato combinando iteratori e metodi estensione
  - Si veda la classe «`System.Linq.Enumerable`»

# Esempio: custom map & filter in C#

```
public static IEnumerable<TOut> Map<TIn, TOut>(this IEnumerable<TIn> sequence, Func<TIn, TOut> mapping)
{
    foreach (var element in sequence)
    {
        yield return mapping(element);
    }
}
```

```
public static IEnumerable<TAny> Filter<TAny>(this IEnumerable<TAny> sequence, Predicate<TAny> predicate)
{
    foreach (var element in sequence)
    {
        if (predicate(element))
        {
            yield return element;
        }
    }
}
```

# ESEMPIO

## 9-Iterators

The screenshot shows the Visual Studio IDE interface with the following details:

- Title Bar:** FILE EDIT VIEW PROJECT BUILD DEBUG TEST ANALYZE TOOLS EXTENSIONS WINDOW HELP Search Visual Studio (Ctrl+Q)
- Solution Explorer:** Shows a solution named "LezioneCSharp" containing 10 projects, with the "9-Iterators" project selected.
- Main Editor:** The file "MainIterators.cs" is open, displaying C# code for iterator methods. The code includes:
  - Imports: using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;
  - Namespace: namespace Iterators
  - Method: static class MyEnumerable
  - Method: public static IEnumerable<Tuple<int, TAny>> Indexed<TAny>(this IEnumerable<TAny> sequence)
  - Method: public static IEnumerable<TAny> EvenPositions<TAny>(this IEnumerable<TAny> sequence)
  - Method: public static void ForEach<TAny>(this IEnumerable<TAny> sequence, Action<TAny> consumer)
  - Method: public static I<sub>n</sub>umerable<TOut> Map<TIn, TOut>(this IEnumerable<TIn> sequence, Func<TIn, TOut> mapping)
- Status Bar:** ShowsLn 8 Col 2 Ch 2 INS ↑ 0 ↩ 68 c-sharp-lesson master

---

## PARTE 6

### File I/O (cenni) e Reflection

# File I/O e serializzazione oggetti

## ANALOGIE

- Classi «stream» per gestire l'input/output in modo unificato su file e altri meccanismi (es. rete, memoria, ...)
- Decoratori per leggere/scrivere agevolmente i principali tipi di dati
  - `DataOutputStream`-`DataInputStream` in Java, `BinaryWriter`-`BinaryReader` in C#
- Meccanismi di serializzazione per rendere gli oggetti persistenti

## DIFFERENZE

- In Java le classi serializzabili implementano l'interfaccia «`Serializable`» ed è possibile specificare cosa non serializzare tramite la keyword «**transient**»
- In C# le classi serializzabili sono marcate con l'attributo `[Serializable]` ed è possibile specificare cosa non serializzare tramite l'attributo `[NonSerialized]`.
  - Inoltre è possibile implementare l'interfaccia «`ISerializable`» per personalizzare la serializzazione (analogamente all'implementazione dei metodi «`readObject`» e «`writeObject`» in Java)
- C# supporta un meccanismo di «version tolerant serialization» per aggiungere nuovi campi alle classi serializzate senza rompere la compatibilità con oggetti salvati da classi della versione precedente.

# File I/O – Esempio

```
import java.io.*;  
  
public class UseDataStream {  
    private static final String P = "/temp/prova.bin";  
  
    public static void main(String[] args) throws IOException  
    {  
        FileOutputStream f = new FileOutputStream(P);  
        DataOutputStream ds = new DataOutputStream(f);  
        ds.writeBoolean(true);  
        ds.writeInt(10000);  
        ds.writeDouble(5.2);  
        ds.close();  
  
        FileInputStream f2 = new FileInputStream(P);  
        DataInputStream ds2 = new DataInputStream(f2);  
        System.out.println(ds2.readBoolean());  
        System.out.println(ds2.readInt());  
        System.out.println(ds2.readDouble());  
        ds2.close();  
    }  
}
```



```
using System.IO;  
  
public class UseDataStream  
{  
  
    private static readonly string P = @"C:\temp\prova.bin";  
  
    public static void Main()  
    {  
        var f = new FileStream(P, FileMode.Create);  
        var ds = new BinaryWriter(f);  
        ds.Write(true);  
        ds.Write(10000);  
        ds.Write(5.2);  
        ds.Close();  
  
        var f2 = new FileStream(P, FileMode.Open);  
        var ds2 = new BinaryReader(f2);  
        System.Console.WriteLine(ds2.ReadBoolean());  
        System.Console.WriteLine(ds2.ReadInt32());  
        System.Console.WriteLine(ds2.ReadDouble());  
        ds2.Close();  
    }  
}
```



# Serializzazione oggetti – Esempio

```
public class CPersona implements Serializable{
    private String nome;
    private int annoNascita;
    transient private String cachedToString = null;

    public String toString(){
        if (this.cachedToString == null){
            cachedToString = nome + ":" + annoNascita;
        }
        return this.cachedToString;
    }

    private static final String STR = "p.bin";
    public static void main(String[] args) throws Exception{
        ObjectOutputStream sOut = new ObjectOutputStream(
                new FileOutputStream(STR));
        CPersona p = new CPersona("Rossi", 1960);
        System.out.println("1) "+p); // cache vuota
        System.out.println("2) "+p); // cache non vuota
        sOut.writeObject(new CPersona("Rossi", 1960));
        sOut.close();

        System.out.println("Ri-carico l'oggetto... ");
        ObjectInputStream sIn = new ObjectInputStream(
                new FileInputStream(STR));
        CPersona q = (CPersona)sIn.readObject();
        System.out.println("1) "+q); // cache vuota
        System.out.println("2) "+q); // cache non vuota
        sIn.close();
    }
}
```



```
[Serializable]
public class CPersona
{
    private String nome;
    private int annoNascita;

    [NonSerialized]
    private String cachedToString;
    ...

    override public string ToString()
    {
        if (this.cachedToString == null)
        { cachedToString = nome + ":" + annoNascita; }
        return this.cachedToString;
    }

    private static readonly string STR = "p.bin";
    public static void Main()
    {
        var bf = new BinaryFormatter();
        var fOut = new FileStream(STR, FileMode.Create);
        CPersona p = new CPersona("Rossi", 1960);
        Console.WriteLine("1) " + p); // cache vuota
        Console.WriteLine("2) " + p); // cache non vuota
        bf.Serialize(fOut, new CPersona("Rossi", 1960));
        fOut.Close();
        Console.WriteLine("Ri-carico l'oggetto... ");
        var fIn = new FileStream(STR, FileMode.Open);
        CPersona q = (CPersona)bf.Deserialize(fIn);
        Console.WriteLine("1) " + q); // cache vuota
        Console.WriteLine("2) " + q); // cache non vuota
        fIn.Close();
    }
}
```



# Reflection

---

## ANALOGIE

- Entrambi i linguaggi forniscono un meccanismo di «reflection», ossia la possibilità di identificare campi e metodi di una classe a runtime, di invocarne i metodi, costruirne nuove istanze, etc.
- Un'apposita classe rappresenta i tipi
  - «`Class<T>`» in Java e «`Type`» in C#
  - Ogni istanza di tale classe è un tipo (es. una certa interfaccia o una certa classe).
  - Altre classi rappresentano metodi, costruttori, etc.

## DIFFERENZE

- In C# la reflection avviene a livello di «assembly», mentre in Java a livello di «class»
- In Java l'istanza di `Class<T>` corrispondente a un determinato tipo si ottiene con «`.class`» (es. `«String.class»`), mentre in C# con «`typeof()`» (es. `«typeof(string)»`)

# Reflection – Esempio

```
import java.io.*;
import java.lang.reflect.*;

class DynamicExecution{
    final static String Q_CLASS = "Nome classe: ";
    final static String Q METH = "Nome metodo: ";
    final static String L_OK = "OK: result = ";
    final static String E_RET = "Errore tipo ritorno";

    public static void main(String[] s) throws Exception{
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));
        System.out.println(Q_CLASS);
        String className = reader.readLine();
        System.out.println(Q METH);
        String methName = reader.readLine();
        Class<?> cl = Class.forName(className);
        Constructor<?> cns = cl.getConstructor();

        Method met = cl.getDeclaredMethod(methName);
        Class<?> rt = met.getReturnType();
        if (!rt.isAssignableFrom(String.class)) {
            throw new NoSuchMethodException(E_RET);
        }

        Object o = cns.newInstance();
        String result = (String)met.invoke(o);
        System.out.println(L_OK + result);
    }
}
```



```
using System;
using System.Reflection;

class DynamicExecution
{
    readonly static string Q_CLASS = "Nome classe: ";
    readonly static string Q METH = "Nome metodo: ";
    readonly static string L_OK = "OK: result = ";
    readonly static string E_RET = "Errore tipo di ritorno";

    public static void Main()
    {
        Console.WriteLine(Q_CLASS);
        string className = Console.ReadLine();
        Console.WriteLine(Q METH);
        string methName = Console.ReadLine();
        Type cl = Type.GetType(className);
        ConstructorInfo cns =
            cl.GetConstructor(Type.EmptyTypes);
        MethodInfo met = cl.GetMethod(methName);
        Type rt = met.ReturnType;
        if (!rt.IsAssignableFrom(typeof(string)))
        {
            throw new InvalidOperationException(E_RET);
        }

        object o = cns.Invoke(null);
        string result = (string)met.Invoke(o, null);
        Console.WriteLine(L_OK + result);
    }
}
```



# PARTE 7

## Thread e Concorrenza, TPL (cenni)

# Thread e concorrenza

## ANALOGIE

- Supporto a livello di linguaggio con keyword per definire blocchi di codice come sezioni critiche
  - «**synchronized**» in Java, «**lock**» in C#
  - keyword «**volatile**» per proibire ottimizzazioni del compilatore che comporterebbero problemi in caso di accessi concorrenti.
- Possibilità di definire interi metodi come regioni critiche
  - keyword «**synchronized**» in Java, attributo `[MethodImpl(MethodImplOptions.Synchronized)]` in C#
- Ricche librerie di classi con costrutti e strutture dati per semplificare lo sviluppo di applicazioni concorrenti
  - es. `java.util.concurrent` in Java, `System.Threading` e `System.Collections.Concurrent` in C#

## DIFFERENZE

- I thread di Java accettano `Runnables`, i thread di .NET accettano **delegates**
- In Java ogni classe eredita `wait()`, `notify()` and `notifyAll()` da `Object`
- In C# i metodi equivalenti sono `Wait()`, `Pulse()` and `PulseAll()` della classe «`System.Threading.Monitor`»

# Thread e concorrenza – Esempio 1

```
class Worker extends Thread {  
    private int from, count;  
    public Worker(int from, int count){  
        this.from = from;  
        this.count = count;  
    }  
  
    public void run() {  
        for (int i = from; i < from+count; i++) {  
            if (isPrime(i)) {  
                System.out.printf("%d\n", i);  
            }  
        }  
    }  
  
    private static boolean isPrime(int num) {  
        int sq = (int)Math.sqrt(num);  
        for (int i = 2; i <= sq; i++) {  
            if ((num % i) == 0) {  
                return false;  
            }  
        }  
        return true;  
    }  
}  
  
final int n = 1000;  
final int t = 4;  
int from = 2;  
int count = (n-from) / t;  
for (int i=0; i < t-1; i++) {  
    new Worker(from, count).start();  
    from += count;  
}  
new Worker(from, n-from).start();
```



```
class Worker {  
    private int from, count;  
    public Worker(int from, int count){  
        this.from = from;  
        this.count = count;  
    }  
  
    public void Run() {  
        for (int i = from; i < from+count; i++) {  
            if (IsPrime(i)) {  
                Console.WriteLine("{0}\n", i);  
            }  
        }  
    }  
  
    private static bool IsPrime(int num){  
        int sq = (int)Math.Sqrt(num);  
        for (int i = 2; i <= sq; i++){  
            if ((num % i) == 0){  
                return false;  
            }  
        }  
        return true;  
    }  
}  
  
const int n = 1000;  
const int t = 4;  
int from = 2;  
int count = (n - from) / t;  
for (int i = 0; i < t-1; i++) {  
    new Thread(new Worker(from, count).Run).Start();  
    from += count;  
}  
new Thread(new Worker(from, n-from).Run).Start();
```



# Thread e concorrenza – Esempio 2

```
class Worker extends Thread
{
    private Object lockCS;
    private String name;
    public Worker(String name, Object lockCS)
    {
        lockCS = lockCS;
        this.name = name;
    }
    public void run()
    {
        for (int i=0;i<1000;i++)
        {
            System.out.printf("%s-1\n", name);
            synchronized(lockCS)
            {
                System.out.printf("%s-2\n", name);
                System.out.printf("%s-3\n", name);
            }
        }
    }
}
```

 JAVA

```
public class TestCS
{
    public static void main(String[] args)
    {
        Object lockCS = new Object();
        Worker a = new Worker("A", lockCS);
        Worker b = new Worker("B", lockCS);
        a.start();
        b.start();
    }
}
```

```
class Worker
{
    private object lockCS;
    private string name;
    public Worker(string name, object lockCS)
    {
        this.lockCS = lockCS;
        this.name = name;
    }
    public void Run()
    {
        for (int i=0;i<1000;i++)
        {
            Console.WriteLine("{0}-1\n", name);
            lock(lockCS)
            {
                Console.WriteLine("{0}-2\n", name);
                Console.WriteLine("{0}-3\n", name);
            }
        }
    }
}
```

 C#

```
public class TestCS
{
    public static void Main()
    {
        object lockCS = new object();
        Worker a = new Worker("A", lockCS);
        Worker b = new Worker("B", lockCS);
        new Thread(a.Run).Start();
        new Thread(b.Run).Start();
    }
}
```

# Task Parallel Library e metodi asincroni

---

## Task<T>

- Rappresenta un'operazione asincrona che restituirà un risultato di tipo T: contiene informazioni sullo stato dell'operazione e il suo risultato quando è completata
- Il parallelismo basato su «Task» è simile a quello basato su thread ma più efficiente e scalabile
- **TPL** (Task Parallel Library)
  - libreria di classi .NET per semplificare lo sviluppo di applicazioni concorrenti

## Metodi asincroni: keyword «**async**» e «**await**»

- Un metodo che restituisce un Task può essere marcato come asincrono con «**async**».
- Ogni volta che un metodo asincrono si mette in attesa del completamento di un (sub) task (con la keyword «**await**»), il controllo torna al chiamante fino a quanto il (sub) task non è completato.

# Task Parallel Library e metodi asincroni

```
class MyClass
{
    private HttpClient client = new HttpClient();
    private string s1 = "http://www.google.com";
    private string s2 = "http://www.unibo.it";

    public void Work()
    { // Crea task, continuazione e torna subito
        Task<string> t1 = client.GetStringAsync(s1);
        t1.ContinueWith(Work2);
    }

    public void Work2(Task<string> t1)
    {
        Console.WriteLine("T1:{0}", t1.Result.Length);
        // Crea un secondo task e attende che termini
        Task<string> t2 = client.GetStringAsync(s2);
        Console.WriteLine("T2:{0}", t2.Result.Length);
    }
}
```



```
class Program
{
    public static void Main()
    {
        MyClass p = new MyClass();
        p.Work();

        while (true)
        {
            Console.Write(".");
            Thread.Sleep(30);
        }
    }
}
```

```
class MyClass
{
    private HttpClient client = new HttpClient();
    private string s1 = "http://www.google.com";
    private string s2 = "http://www.unibo.it";

    public async void Work()
    {
        string str1 = await client.GetStringAsync(s1);
        Console.WriteLine("T1: {0}", str1.Length);
        string str2 = await client.GetStringAsync(s2);
        Console.WriteLine("T2: {0}", str2.Length);
    }
}
```

# LINQ e TPL: un esempio

```
class Worker {
    private int from, count;
    public Worker(int from, int count)
    { this.from = from; this.count = count; }

    public void Run() {
        for (int i = from; i < from+count; i++) {
            if (IsPrime(i))
                Console.WriteLine("{0}\n", i);
        }
    }
}

private static bool IsPrime(int num) {
    int sq = (int)Math.Sqrt(num);
    for (int i = 2; i <= sq; i++) {
        if ((num % i) == 0)
            return false;
    }
    return true;
}

const int n = 1000;
const int t = 4;
int from = 2;
int count = (n - from) / t;
for (int i = 0; i < t-1; i++) {
    new Thread(new Worker(from, count).Run).Start();
    from += count;
}
new Thread(new Worker(from, n-from).Run).Start();
```

□ Qui sotto un'implementazione con LINQ e TPL dell'esempio a sinistra:

- In IsPrime, il metodo di estensione LINQ All(), applicato a una sequenza di int, restituisce true se tutti gli elementi soddisfano il predicato.
- Partitioner.Create suddivide i numeri in sequenze affidate a task diversi da Parallel.ForEach.

```
private static bool IsPrime(int num)
{
    return Enumerable.Range(2, (int)Math.Sqrt(num))
                    .All(i => num % i != 0);
}

...
var part = Partitioner.Create(2, 1000);
Parallel.ForEach(part, p =>
{
    for (int i = p.Item1; i < p.Item2; i++)
    {
        if (IsPrime(i))
        {
            Console.WriteLine("{0}\n", j);
        }
    }
});
```

---

## PARTE 8

### Metadati, Annotazioni e Documentazione

# Metadati/annotazioni

---

## ANALOGIE

- Sia Java che C# consentono di annotare il codice con metadati per estendere le funzionalità del linguaggio di programmazione e le potenzialità delle applicazioni a runtime
  - in C# si utilizzano gli «attributi», in Java le «annotazioni».
- In entrambi i casi alcuni tipi di annotazioni sono già disponibili ed è possibile definire nuovi tipi di annotazioni
- Un esempio fra i molteplici impieghi di tali metadati
  - in entrambi i linguaggi sono utilizzati per annotare le classi e i metodi di test nei più diffusi framework di «unit testing»

## DIFFERENZE

- La sintassi è differente: `[Attribute]` in C# e `@Annotation` in Java)
- Il C# supporta anche alcune direttive per il compilatore che si possono inserire nel codice sorgente in modo simile al C/C++ (es. `#pragma`, `#region...#endregion`, ...)

# Metadati/annotazioni – Esempio

```
class A
{
    protected void m(int x) {}

class B extends A
{
    @Override
    protected void m(int x) {}

    @Override
    protected void m(string x) {} // comp. error

    @Deprecated
    public void old() {} // comp. warning se usato
}

...
static void deprec()
{
    new B().old(); // genera il warning
}

@SuppressWarnings("deprecation")
static void noDeprec()
{
    new B().old(); // non genera il warning
}
```



```
class A
{
    protected virtual void m(int x) {}

class B : A
{
    override // N.B. in C# è una keyword obbligatoria
    protected void m(int x) {}

    override
    protected void m(string x) {} // comp. error

    [Obsolete]
    public void Old() {} // comp. warning se usato
}

...
static void deprec()
{
    new B().Old(); // genera il warning
}

#pragma warning disable 0612, 0618
static void noDeprec()
{
    new B().Old(); // non genera il warning
}
#pragma warning restore 0612, 0618
```



# Metadati/annotazioni per Unit Testing

```
import static org.junit.Assert.*;
import org.junit.Test;

class CounterTest
{
    @Test
    public void test1()
    {
        Counter c = new Counter();
        c.increment();
        c.increment();
        assertTrue("increment() non funziona",
                   c.getValue() == 2);
    }

    @Test
    public void test2()
    {
        Counter c = new Counter();
        assertTrue("Il valore non è inizialmente zero",
                   c.getValue() == 0);
    }
}
```



```
using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]
public class CounterTest
{
    [TestMethod]
    public void Test1()
    {
        Counter c = new Counter();
        c.Increment();
        c.Increment();
        Assert.IsTrue(c.GetValue() == 2,
                      "Increment() non funziona");
    }

    [TestMethod]
    public void Test2()
    {
        Counter c = new Counter();
        Assert.IsTrue(c.GetValue() == 0,
                      "Il valore non è inizialmente zero");
    }
}
```



# Generazione di documentazione dai commenti

## ANALOGIE

- Sia Java che C# forniscono un meccanismo per estrarre commenti appositamente formattati dal codice sorgente e inserirli in automatico in un documento separato.
- Questi commenti sono tipicamente utilizzati per generare la documentazione del codice sorgente
  - API Reference

## DIFFERENZE

- In Java i commenti sono delimitati da `/** ... */`
  - il contenuto è in formato HTML con speciali tag (@param, @return, ...).
  - Il tool «**javadoc**» produce una documentazione HTML a partire da tali commenti.
- In C# i commenti sono delimitati da `/** ... */` o preceduti da `///` su ogni riga (caso più comune)
  - il contenuto è in formato XML con appositi tag predefiniti (`<param>`, `<returns>`, ...).
  - Il compilatore C# produce un documento XML a partire da tali commenti, che può poi essere utilizzato dallo stesso ambiente di sviluppo (es. per visualizzare tooltip nell'editor dei sorgenti con la descrizione di classi e metodi), o da appositi tool per produrre documentazione.

# Generazione di documentazione dai commenti

```
/**  
 * Class level summary documentation...  
 * <p>Longer comments...  
 * <p>other comments...  
 * <p>other comments...  
 */  
class SomeClass  
{  
    /** The class constructor. */  
    public SomeClass() { }  
  
    /** Desc. for SomeMethod.  
     * @param s Description for s  
     * @see String Reference to a type or  
     * member */  
    public void SomeMethod(String s) { }  
  
    /** Some other method.  
     * @return Return results descr.  
     * @link #SomeMethod(string)  
     * Notice the use of the @link to  
     * reference a specific method */  
    public int SomeOtherMethod() { return 0; }  
  
    /**  
     * The entry point for the application.  
     *  
     * @param args A list of command line  
     * arguments */  
    public static void main(String[] args)  
    {  
    }
```

```
/// <summary>  
/// Class level summary documentation...</summary>  
/// <remarks>Longer comments...  
/// <para>other comments...</para>  
/// <para>other comments...</para>  
/// </remarks>  
public class SomeClass  
{  
    /// <summary>The class constructor.</summary>  
    public SomeClass() { }  
  
    /// <summary>Desc. for SomeMethod.</summary>  
    /// <param name="s">Description for s</param>  
    /// <seealso cref="String">Reference to a type or  
    /// member.</seealso>  
    public void SomeMethod(string s) { }  
  
    /// <summary>Some other method.</summary>  
    /// <returns>Return results descr.</returns>  
    /// <seealso cref="SomeMethod(string)">  
    /// Notice the use of the cref attribute to  
    /// reference a specific method </seealso>  
    public int SomeOtherMethod() { return 0; }  
  
    /// <summary>  
    /// The entry point for the application.  
    /// </summary>  
    /// <param name="args"> A list of command line  
    /// arguments</param>  
    public static void Main(string[] args)  
    {  
    }
```



# Riferimenti

## MSDN - Visual C#

- <https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx>
- <https://msdn.microsoft.com/en-us/library/618ayhy6.aspx>