



# Alma Mater Studiorum-Università di Bologna Scuola di Ingegneria

---

## Stringhe di caratteri Un caso di studio: codice fiscale

*Corso di Laurea in Ingegneria Informatica*

Anno accademico 2021/2022

**Prof. ENRICO DENTI**

*Dipartimento di Informatica – Scienza e Ingegneria (DISI)*



# STRINGHE IN C

In C, le *stringhe* sono *array di caratteri*

- pezzi di memoria con dentro dei caratteri, *privi di identità collettiva* (come ogni array C..)
- liberamente modificabili in ogni singolo carattere

Conseguenze:

- una stringa C è un contenitore di caratteri, *non una specifica parola: in realtà, le stringhe... non esistono!*
  - una stringa “**punto**” può ad esempio diventare “**ponte**” sostituendo un paio di caratteri...
- rischio di errato uso dei puntatori, problemi in caso di "allungamento" della stringa
  - una stringa “**pianta**” potrebbe diventare “**piantina**”..



# STRINGHE IN C vs. STRINGHE COME OGGETTI

---

In Java & co., cambia il punto di vista:

- le stringhe *NON SONO PIÙ array di caratteri !*
  - un array di caratteri è un'altra cosa, DIVERSA da una stringa
- le stringhe sono *OGGETTI*, istanze della classe **String**
  - per scelta progettuale, i singoli caratteri NON SONO modificabili

Conseguenze:

- una stringa viene vista come *una specifica parola*
  - NON si può trasformare in un'altra, perché ogni altra parola è una stringa DIVERSA: le parole non si trasformano, esistono!
  - non ha senso dire che “**punto**” diventa “**ponte**” cambiando due caratteri, perché nel dizionario esistono sempre entrambe!
  - gli oggetti **String** non sono "contenitori di caratteri", sono *valori stringa* che rappresentano ciascuno una data parola



# STRINGHE: COSTRUZIONE

- *Concettualmente* gli oggetti stringa sono costruiti da un costruttore **String** che effettua la normale **new**
  - idealmente, quindi, dovremmo scrivere qualcosa del tipo:  
`String s = new String('c','i','a','o');`
  - ma sarebbe *impraticabile* (infatti, un tale costruttore non esiste)
- *Concretamente* si opta quindi per la *scrittura diretta di costanti stringa (literal) racchiuse fra virgolette*:
  - NON scriveremo `new String('c','i','a','o');`
  - MA semplicemente `"ciao";`

intendendosi che ciò sia *equivalente a effettuare una new ottimizzata* (che evita di ricreare stringhe già esistenti)



# STRINGHE vs. ARRAY DI CARATTERI

- Per questi motivi, le stringhe *non sono array di caratteri*, sono *oggetti concettualmente e praticamente diversi*
  - Java lo sottolinea *impedendo di applicare* alle stringhe i classici operatori degli array, come `[]`: per *selezionare il carattere i-esimo* si usa il metodo **`charAt`**
  - C#, Scala, Kotlin mantengono l'operatore `[]`, ma *solo in lettura*

## ESEMPIO JAVA

```
String s = "Nel mezzo del cammin";  
char ch = s.charAt(4);           // solo in lettura
```

## ESEMPIO C# (~Scala, ~Kotlin)

```
string s = "Nel mezzo del cammin";  
char ch = s[4];                 // solo in lettura
```

Scala: **var** ch : Char = **s**(4);

Kotlin: **var** ch : Char = **s**[4];



# STRINGHE vs. ARRAY DI CARATTERI

- Poiché gli oggetti stringa sono imm modificabili, i singoli caratteri sono *selezionabili solo in lettura*
- Non è possibile sovrascriverne uno con un altro

## CONTROESEMPIO JAVA

```
String s = "Nel mezzo del cammin";  
s.charAt(4) = 'Q'; // NO! VIETATO!
```

NO!

## CONTROESEMPIO C# (~Scala, ~Kotlin)

```
string s = "Nel mezzo del cammin";  
s[4] = 'Q'; // NO! VIETATO!
```

NO!

Scala: *s(4) = 'Q' ;*  
Error: *value update is not a member of String*

Kotlin: *s[4] = 'Q' ;*  
Error: *no set method providing array access (mmhhhh)...*



# JAVA: LA CLASSE `String`

---

- La classe `String` offre decine di metodi per
  - ricerche
  - confronti
  - conversioni
  - estrazioni di sottostringhe
  - etc.
- Coerentemente con l'approccio basato sull'idea di stringhe immutabili, *tutti i metodi restituiscono sempre nuovi valori, senza mai alterare l'originale.*
- In C#, Scala, Kotlin l'approccio è analogo: i metodi hanno nomi simili (ma non sempre identici) a quelli Java.



# LA CLASSE `String`: METODI

Costruiscono e restituiscono un nuovo oggetto **`String`** contenente quanto richiesto

```
char charAt(int index)
int length()
int compareTo(Object o)
int compareTo(String s)
int compareToIgnoreCase(String s)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String s)
int indexOf(int ch)
int indexOf(int ch, int from)
int indexOf(String s)
int indexOf(String s, int from)
int lastIndexOf(int ch)
int lastIndexOf(int ch, int from)
int lastIndexOf(String s)
int lastIndexOf(String s, int i)
```

```
String replace(char c1, char c2)
String toLowerCase()
String toLowerCase(Locale locale)
String toUpperCase()
String toUpperCase(Locale locale)
String trim()
String substring(int beginIndex)
String substring(int beginIndex,
                  int endIndex)

boolean endsWith(String s)
boolean startsWith(String p)
boolean startsWith(String p,
                    int offset)
```



# ESEMPIO in Java e C#

```
String s1 = "Nel mezzo del cammin";
String s2 = s1.replace('z', 'Z');
System.out.println( s1.equals(s2) );
System.out.println( s1.equalsIgnoreCase(s2) );
System.out.println( s2.toLowerCase() );
System.out.println( s1.startsWith("Del") );
System.out.println( s1.startsWith("Nel") );
```

Java

Invocazione di  
metodi su oggetti  
String

```
string s1 = "Nel mezzo del cammin";
string s2 = s1.Replace('z', 'Z');
System.Console.WriteLine( s1.Equals(s2) );
System.Console.WriteLine( s1.Equals(s2,
    System.StringComparison.InvariantCultureIgnoreCase) );
System.Console.WriteLine( s2.ToLower() );
System.Console.WriteLine( s1.StartsWith("Del") );
System.Console.WriteLine( s1.StartsWith("Nel") );
```

false  
true  
nel mezzo del cammin  
false  
true

C#: False  
e True

C#: il confronto *ignore case* è  
fatto sempre dal metodo **Equals**  
ma con opportuno argomento

C#



# ESEMPIO in Scala e Kotlin

```
var s1 = "Nel mezzo del cammin";  
var s2 = s1.replace('z', 'Z');  
println( s1.equals(s2) );  
println( s1.equalsIgnoreCase(s2) );  
println( s2.toLowerCase() );  
println( s1.startsWith("Del") );  
println( s1.startsWith("Nel") );
```

Scala

Scala: il confronto *ignore case* è fatto da un metodo apposito, come in Java

```
var s1 = "Nel mezzo del cammin";  
var s2 = s1.replace('z', 'Z');  
println( s1.equals(s2) );  
println( s1.equals(s2, true) );  
println( s2.toLowerCase() );  
println( s1.startsWith("Del") );  
println( s1.startsWith("Nel") );
```

```
false  
true  
nel mezzo del cammin  
false  
true
```

Kotlin: il confronto *ignore case* è fatto sempre dal metodo `equals` ma con opportuno flag boolean

Kotlin



# COSTANTI STRINGA e CONCATENAZIONE DI STRINGHE

- Le **costanti stringa** si denotano nel solito modo, con una sequenza di caratteri fra virgolette:  
**"ciao"**                      **" mondo\n"**
- Per ogni nuova costante stringa viene *creato implicitamente un nuovo oggetto di classe **String*** (in C#, `string`)
- La **concatenazione** è espressa dall'*operatore **+*** che *crea un nuovo oggetto stringa* inizializzato al valore risultante: nessuna delle stringhe iniziali viene modificata.

## ESEMPIO

```
String s1 = "ciao",  
        s2 = " mondo";  
  
String s3 = s1 + s2;
```

s1	ciao
s2	mondo
s3	ciao mondo

# CONCATENAZIONE DI STRINGHE

Se un riferimento a stringa viene ri-assegnato:

**s1 = s1 + s2;**

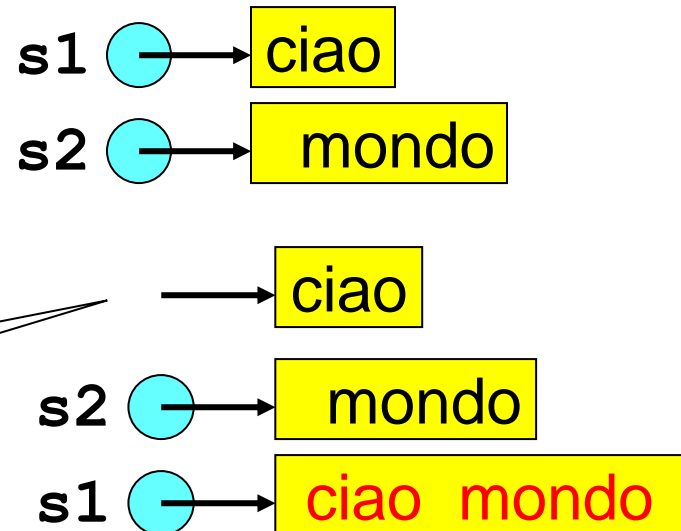
- il riferimento **s1** punta al *nuovo oggetto risultante*..
- *...ma l'oggetto precedente continua a esistere!*  
(se non più utile, sarà poi distrutto dal garbage collector)

## ESEMPIO

```
String s1 = "ciao",  
        s2 = " mondo";
```

**s1 = s1 + s2;**

Oggetto immutato,  
accessibile da altri ref



# CONCATENAZIONE DI STRINGHE

Ciò può essere confermato da un semplice test:

```
var s1 = "ciao";  
var s = s1;  
var s2 = " mondo";  
s1 = s1 + s2;  
System.out.println(s1);  
System.out.println(s);
```

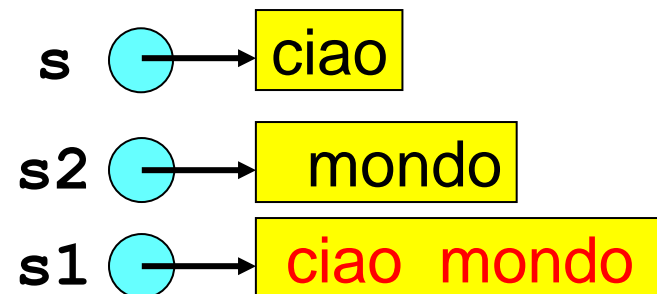
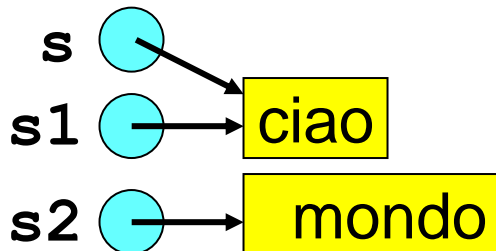
Java

~C#

~Scala

~Kotlin

```
ciao mondo  
ciao
```





# CONCATENAZIONE DI STRINGHE

La concatenazione di stringhe avviene:

- a *run-time*, se ci sono *variabili*

```
String s2 = "ciao" + s1;
```

- a *compile-time*, se ci sono solo *costanti*  
→ *non introduce inefficienze*

```
String s = "ciao" + " mondo\n"
```

Ciò è utile perché *una stringa non può eccedere la riga*

- stringhe più lunghe vanno *spezzate e concatenate* con +
- da Java 15, nonché in Scala e Kotlin, è però possibile definire del *text blocks multilinea* delimitati dal delimitatore `"""`
- C# offre un'idea simile, il *verbatim string literal*, introdotto da `@`



# TEXT BLOCKS IN Java, Scala, Kotlin

In Java, un text block è delimitato da `"""`

- il delimitatore iniziale deve stare su riga a sé stante
- apposite *indent policy* stabiliscono come gestire l'indentazione del testo in base alla *posizione del delimitatore di chiusura*

```
/** Java15 text block feature
    INDENT POLICY
    Normally, one would format a text block in two ways:
    - first, position the left edge of the content to appear under the first "
      of the opening delimiter
    - second, place the closing delimiter on its own line to appear exactly under
      the opening delimiter.
    The resulting string will have no white space at the start of any line,
    and will not include the trailing blank line of the closing delimiter.

    However, because the trailing blank line is considered a determining line,
    moving it to the left has the effect of reducing the common white space prefix,
    and therefore reducing the the amount of white space that is stripped from the
    start of every line.
    In the extreme case, where the closing delimiter is moved all the way to the left,
    that reduces the common white space prefix to zero, effectively opting out of
    white space stripping.
    */
```

# TEXT BLOCKS: ESEMPIO

Francesca Michielin senza text blocks:

```
public static void test0(){
    String s = "E' la prima volta che mi capita" + "\n" +
               "Prima mi chiudevo in una scatola" + "\n" +
               "Sempre un po' distante dalle cose della vita" + "\n" +
               "Perché così profondamente non l'avevo mai sentita" + "\n" +
               "E poi ho sentito un'emozione accendersi veloce" + "\n" +
               "E farsi strada nel mio petto senza spegnere la voce" + "\n" +
               "E non sentire più tensione solo vita dentro di me" + "\n" +
               "Nessun grado di separazione" + "\n" +
               "Nessun tipo di esitazione" + "\n" +
               "Non c'è più nessuna divisione" + "\n" +
               "Tra di noi" + "\n" +
               "Siamo una sola direzione in questo universo" + "\n" +
               "Che si muove" + "\n" +
               "Non c'è nessun grado di separazione" + "\n" +
               "Davo meno spazio al cuore e più alla mente" + "\n" +
               "Sempre un passo indietro" + "\n" +
               "E l'anima in allerta" + "\n" +
               "E guardavo il mondo da una porta" + "\n" +
               "Mai completamente aperta" + "\n" +
               "E non da vicino" + "\n" +
               "E no non c'è alcuna esitazione" + "\n" +
               "Finalmente dentro di me" + "\n" +
               "Nessun grado di separazione" + "\n" +
               "Nessun..." + "\n";
    System.out.println("--STANDARD STRINGS--LENGTH:"+s.length());
    System.out.println(s);
}
```

761



# TEXT BLOCKS: ESEMPIO

Francesca Michielin senza text blocks:

```
public static void test0b(){
    String s = "E' la prima volta che mi capita" + System.lineSeparator() +
               "Prima mi chiudevo in una scatola" + System.lineSeparator() +
               "Sempre un po' distante dalle cose della vita" + System.lineSeparator() +
               "Perché così profondamente non l'avevo mai sentita" + System.lineSeparator() +
               "E poi ho sentito un'emozione accendersi veloce" + System.lineSeparator() +
               "E farsi strada nel mio petto senza spegnere la voce" + System.lineSeparator() +
               "E non sentire più tensione solo vita dentro di me" + System.lineSeparator() +
               "Nessun grado di separazione" + System.lineSeparator() +
               "Nessun tipo di esitazione" + System.lineSeparator() +
               "Non c'è più nessuna divisione" + System.lineSeparator() +
               "Tra di noi" + System.lineSeparator() +
               "Siamo una sola direzione in questo universo" + System.lineSeparator() +
               "Che si muove" + System.lineSeparator() +
               "Non c'è nessun grado di separazione" + System.lineSeparator() +
               "Davo meno spazio al cuore e più alla mente" + System.lineSeparator() +
               "Sempre un passo indietro" + System.lineSeparator() +
               "E l'anima in allerta" + System.lineSeparator() +
               "E guardavo il mondo da una porta" + System.lineSeparator() +
               "Mai completamente aperta" + System.lineSeparator() +
               "E non da vicino" + System.lineSeparator() +
               "E no non c'è alcuna esitazione" + System.lineSeparator() +
               "Finalmente dentro di me" + System.lineSeparator() +
               "Nessun grado di separazione" + System.lineSeparator() +
               "Nessun..." + System.lineSeparator();
    System.out.println("--STANDARD STRINGS--LENGTH:"+s.length());
    System.out.println(s);
}
```

In Windows 785 (=761+24),  
perché lineSeparator() è "\n\r"

# TEXT BLOCKS: ESEMPIO

Francesca Michielin con text blocks (cambia l'indentazione):

```
public static void test1(){
    String s = """
        E' la prima volta che mi capita
        Prima mi chiudevo in una scatola
        Sempre un po' distante dalle cose della vita
        Perché così profondamente non l'avevo mai sentita
        E poi ho sentito un'emozione accendersi veloce
        E farsi strada nel mio petto senza spegnere la voce
        E non sentire più tensione solo vita dentro di me
        Nessun grado di separazione
        Nessun tipo di esitazione
        Non c'è più nessuna divisione
        Tra di noi
        Siamo una sola direzione in questo universo
        Che si muove
        Non c'è nessun grado di separazione
        Davo meno spazio al cuore e più alla mente
        Sempre un passo indietro
        E l'anima in allerta
        E guardavo il mondo da una porta
        Mai completamente aperta
        E non da vicino
        E no non c'è alcuna esitazione
        Finalmente dentro di me
        Nessun grado di separazione
        Nessun...
        """;
    System.out.println("--TEXT BLOCK--LENGTH:" + s.length());
    System.out.println(s);
}
```

```
public static void test2(){
    String s = """
        E' la prima volta che mi capita
        Prima mi chiudevo in una scatola
        Sempre un po' distante dalle cose della vita
        Perché così profondamente non l'avevo mai sentita
        E poi ho sentito un'emozione accendersi veloce
        E farsi strada nel mio petto senza spegnere la voce
        E non sentire più tensione solo vita dentro di me
        Nessun grado di separazione
        Nessun tipo di esitazione
        Non c'è più nessuna divisione
        Tra di noi
        Siamo una sola direzione in questo universo
        Che si muove
        Non c'è nessun grado di separazione
        Davo meno spazio al cuore e più alla mente
        Sempre un passo indietro
        E l'anima in allerta
        E guardavo il mondo da una porta
        Mai completamente aperta
        E non da vicino
        E no non c'è alcuna esitazione
        Finalmente dentro di me
        Nessun grado di separazione
        Nessun...
        """;
    System.out.println("--TEXT BLOCK--INDENT--LENGTH:" + s.length());
    System.out.println(s);
}
```

```
E no non c'è alcuna esitazione
Finalmente dentro di me
Nessun grado di separazione
Nessun...
""";
```

# TEXT BLOCKS: ESEMPIO

Francesca Michielin con text blocks (cambia l'indentazione):

```
public static void test1(){
    String s = """
        E' la prima volta che mi capita
        Prima mi chiudevo in una scatola
        Sempre un po' distante dalle cose della vita
        Perché così profondamente non l'avevo mai sentita
        E poi ho sentito un'emozione accendersi veloce
        E farsi strada nel mio petto senza spegnere la voce
        E non sentire più tensione solo vita dentro di me
        Nessun grado di separazione
        Nessun tipo di esitazione
        Non c'è più nessuna divisione
        Tra di noi
        Siamo una sola direzione in questo universo
        Che si muove
        Non c'è nessun grado di separazione
        Davo meno spazio al cuore e più alla mente
        Sempre un passo indietro
        E l'anima in allerta
        E guardavo il mondo da una porta
        Mai completamente aperta
        E non da vicino
        E no non c'è alcuna esitazione
        Finalmente dentro di me
        Nessun grado di separazione
        Nessun...
        """;
    System.out.println("--TEXT BLOCK--LENGTH:" + s.length());
    System.out.println(s);
}
```

761

```
public static void test2(){
    String s = """
        E' la prima volta che mi capita
        Prima mi chiudevo in una scatola
        Sempre un po' distante dalle cose della vita
        Perché così profondamente non l'avevo mai sentita
        E poi ho sentito un'emozione accendersi veloce
        E farsi strada nel mio petto senza spegnere la voce
        E non sentire più tensione solo vita dentro di me
        Nessun grado di separazione
        Nessun tipo di esitazione
        Non c'è più nessuna divisione
        Tra di noi
        Siamo una sola direzione in questo universo
        Che si muove
        Non c'è nessun grado di separazione
        Davo meno spazio al cuore e più alla mente
        Sempre un passo indietro
        E l'anima in allerta
        E guardavo il mondo da una porta
        Mai completamente aperta
        E non da vicino
        E no non c'è alcuna esitazione
        Finalmente dentro di me
        Nessun grado di separazione
        Nessun...
        """;
    System.out.println("--TEXT BLOCK--INDENT--LENGTH:" + s.length());
    System.out.println(s);
}
```

785 (+24 tab)

857 (+24\*4 tab)

# TEXT BLOCKS in C

- In C#, il text block si chiama *verbatim string literal* ed è una normale stringa prefissata dal carattere @
  - ma occhio all'indentazione..

```
string s = @"Nel mezzo del cammin di nostra vita  
mi ritrovai per una selva oscura,  
che la diritta via era smarrita";
```

```
1  
2 public class Program  
3 {  
4     public static void Main()  
5     {  
6         string s = @"Nel mezzo del cammin di nostra vita  
7         mi ritrovai per una selva oscura,  
8         che la diritta via era smarrita";  
9         System.Console.WriteLine(s);  
10    }  
11 }
```

```
Nel mezzo del cammin di nostra vita  
mi ritrovai per una selva oscura,  
che la diritta via era smarrita
```



# UGUAGLIANZA FRA STRINGHE

Due stringhe si possono confrontare:

- con l'**operatore ==**
- con il metodo predefinito **equals** (in C#, `Equals`)

ma *la semantica di == cambia* fra Java e gli altri linguaggi

In Java:

- l'operatore **==** esprime un confronto *fra riferimenti*
- il metodo **equals** esprime un confronto *fra valori*

In C#, Scala, Kotlin:

ATTENZIONE: in C# normalmente `==` confronta riferimenti, è solo per le stringhe che confronta valori!!

- l'operatore **==** è un *alias per equals* → confronta *valori*
- per confrontare *riferimenti* (confronto di identità) si deve ricorrere a *metodi ausiliari* (in Kotlin, all'operatore `===`)



# UGUAGLIANZA FRA STRINGHE

## Uguaglianza fra riferimenti

- Java: **==**
- C#: **Object.ReferenceEquals**
- Scala: **eq**
- Kotlin: **===**

## Uguaglianza fra valori

- Java: **equals**
- C#: **==, Equals**
- Scala: **==, equals**
- Kotlin: **==, equals**

Attenzione: in C#, == confronta valori solo per le stringhe: per ogni altro oggetto, confronta riferimenti (come in Java)



# UGUAGLIANZA FRA STRINGHE: ESEMPI

```
public class Code {  
    public static void main(String[] args)  
    {  
        var s1 = "ciao mondo";  
        var s2 = "ciao mondo";  
        var s3 = "Ciao Mondo";  
        System.out.println(s1==s2);  
        System.out.println(s1.equals(s2));  
        var s4 = s3.toLowerCase();  
        System.out.println(s1==s4);  
        System.out.println(s1.equals(s4));  
    }  
}
```

Java

Output

true  
true  
false  
true

```
1 object Test{  
2     def main(args: Array[String]) : Unit = {  
3         var s1 = "ciao mondo";  
4         var s2 = "ciao mondo";  
5         var s3 = "Ciao Mondo";  
6         println(s1==s2);  
7         println(s1.equals(s2));  
8         println(s1.eq(s2));  
9         var s4 = s3.toLowerCase();  
10        println(s1==s4);  
11        println(s1.equals(s4));  
12        println(s1.eq(s4));  
13    }  
14 }
```

true  
true  
true  
true  
true  
false

Scala

```
fun main() {  
    var s1 = "ciao mondo";  
    var s2 = "ciao mondo";  
    var s3 = "Ciao Mondo";  
    println(s1==s2);  
    println(s1.equals(s2));  
    println(s1===s2);  
    var s4 = s3.toLowerCase();  
    println(s1==s4);  
    println(s1.equals(s4));  
    println(s1===s4);  
}
```

true  
true  
true  
true  
true  
false

C# PlayGround by Alen Vadassery

```
1 using System;  
2  
3 public class Program  
4 {  
5     public static void Main()  
6     {  
7         var s1 = "ciao mondo";  
8         var s2 = "ciao mondo";  
9         var s3 = "Ciao Mondo";  
        Console.WriteLine(s1==s2);  
        Console.WriteLine(s1.Equals(s2));  
        Console.WriteLine(Object.ReferenceEquals(s1,s2));  
        var s4 = s3.ToLower();  
        Console.WriteLine(s1==s4);  
        Console.WriteLine(s1.Equals(s4));  
        Console.WriteLine(Object.ReferenceEquals(s1,s4));  
    }  
}
```

C#

Kotlin



# RAPPRESENTAZIONE DI OGGETTI IN FORMA DI STRINGA

- Ogni classe dispone di un metodo **toString** (`ToString` in C#) con cui ottenere una *rappresentazione stampabile* di qualunque istanza della classe
  - è responsabilità del progettista definire un metodo **toString** che produca una stringa “significativa” per quella classe
  - in Java, Scala, Kotlin il metodo **toString** predefinito stampa un *identificativo alfanumerico* dell’oggetto **Counter@4abc9**
  - in C#, il metodo **ToString** predefinito stampa il *nome della classe* di cui l’oggetto è istanza **Counter**
- Tale stringa è *usata automaticamente* quando si chiede di «stampare» un oggetto via **println** (`WriteLine` in C#)





# ESEMPIO

```
public class Esempio5 {  
    public static void main(String args[]) {  
        Counter c = new Counter(10);  
        System.out.println(c);  
    }  
}
```

Java

C#, Scala, Kotlin:  
minime modifiche

Equivale a `c.toString()`, quindi  
stampa un *identificativo* dell'oggetto

Counter@4abc9

```
public class Esempio5bis {  
    public static void main(String args[]) {  
        System.out.println(new Counter(10));  
    }  
}
```

Java

Identica senza variabile di appoggio



# RIDEFINIZIONE DI `toString`

- La stampa di poco fa non vi piace?  
La ritenete poco significativa?
- Si può **ridefinire esplicitamente il metodo `toString`**  
*facendogli stampare ciò che si preferisce*

– in Java:

```
@Override  
public String toString()
```

Java: `@Override` è una  
annotazione per il compilatore

– in C#:

```
public override string ToString()
```

C#, Scala, Kotlin:  
`override` è una  
parola chiave

– in Scala:

```
override def toString() : String
```

– in Kotlin:

```
override fun toString() : String
```

Indicano che si  
sta sovrascrivendo  
una versione  
precedente  
(*design intent*)



# ESEMPIO: Counter CON toString

```
public class Counter {  
    ... // tutto quello che c'era già prima  
    public String toString() {  
        return "Counter di valore " + value; }  
}
```

Java

Lo stesso **Esempio5** di poco fa, *senza neanche bisogno di essere ricompilato*, ora stampa:

**Counter di valore 10**

C#, Scala, Kotlin:  
analogia modifica



# ESEMPIO: Frazione CON toString

```
public class Frazione {  
    ... // tutto quello che c'era già prima  
    public String toString() {  
        return this.getNum() + "/" + this.getDen(); }  
}
```

Java

Un piccolo main di prova:

```
Frazione frazione2 = new Frazione(1,4);  
System.out.println("Creata la frazione " + frazione2);  
  
Frazione frazione3 = new Frazione(-1,8);  
System.out.println("Creata la frazione " + frazione3);
```

Creata la frazione 1/4  
Creata la frazione -1/8

C#, Scala, Kotlin:  
analoga modifica

# E I TIPI PRIMITIVI..?

- Il metodo **toString** è definito *per ogni classe*, ossia per ogni tipo di *oggetto* – ma **non per i tipi primitivi**
  - perché (banalmente) questi ultimi non sono classi e quindi non c'è alcun posto in cui definirne «metodi»
- PROBLEMA: come convertire in stringa valori **primitivi** ?
  - non potendo definire *metodi*, non resta che scrivere funzioni «tradizionali» (ossia, statiche) poste in qualche «libreria»
  - ad esempio, si potrebbe inventare la seguente:

```
public class StringLibrary {  
    static String convert(boolean b)  
    static String convert(double d)  
    static String convert(float f)  
    static String convert(int i)  
    static String convert(long l)  
}
```

Java



# E I TIPI PRIMITIVI..?

- Se si facesse così, un cliente che dovesse convertire in stringa valori primitivi non dovrebbe far altro che invocare la giusta funzione di libreria:

```
int a = 35; double d = 3.14;
```

```
String sA = StringLibrary.convert(a);
```

```
String sD = StringLibrary.convert(d);
```

```
System.out.println("Primo valore: " + sA);
```

```
System.out.println("Secondo valore: " + sD);
```

Java

- Questo approccio funziona, ma costringe a definire una libreria ad hoc (e ricordarsela): rende tutto più complicato
- *Lesson learned: i tipi primitivi sono una continua sorgente di "rumore" e disuniformità*

# OLTRE I TIPI PRIMITIVI: EVERYTHING IS AN OBJECT

- In C#, Scala e Kotlin numeri, caratteri e boolean non sono più tipi primitivi: vengono promossi a vere *classi*
  - i valori numerici, caratteri e boolean sono perciò *veri oggetti* a cui si può *chiedere di fare qualcosa*
- Non è quindi più necessario trattarli diversamente: come ogni altro oggetto, **anch'essi possiedono una toString** che può essere invocata per convertirli in stringa!

```
public class StringLibrary {  
    static String convert(boolean b)  
    static String convert(double d)  
    static String convert(float f)  
    static String convert(int i)  
    static String convert(long l)  
}
```

**Metodo toString()** della  
corrispondente classe  
**Boolean, Char, Float,  
Double, Int, Long..**

# L'ESEMPIO NEGLI ALTRI LINGUAGGI

```
int a = 35; double d = 3.14;  
String sA = StringLibrary.convert(a);  
String sD = StringLibrary.convert(d);  
  
System.out.println("Primo valore: " + sA);  
System.out.println("Secondo valore: " + sD);
```

Java

Java: necessario  
usare funzioni  
statiche di libreria

```
var a = 35; val d = 3.14;  
val sA = a.toString();  
val sD = d.toString();  
// o anche, direttamente  
val sA = 35.toString();  
val sD = 3.14.toString();
```

Scala

Kotlin

C#, Scala, Kotlin: possibile usare  
il normale metodo `toString`  
della classe corrispondente,  
*anche direttamente sui valori!*

```
var sD = 3.14.ToString();  
System.Console.WriteLine(sD);  
  
var sF = 1.44F.ToString();  
System.Console.WriteLine(sF);
```

C#



# E IN JAVA...?

- In Java, l'idea della libreria accessoria funziona ma, come evidenziato, *non è una soluzione meravigliosa*
  - si rompe l'unitarietà del design: per lavorare con le stringhe servono *sia* la classe `String`, *sia* la libreria accessoria (`StringLibrary`)
  - ***molto seccante!***

```
public class StringLibrary {  
    static String convert(boolean b)  
    static String convert(double d)  
    static String convert(float f)  
    static String convert(int i)  
    static String convert(long l)  
}
```



- Bisogna quanto meno *ridurre la portata* del problema



# LA CLASSE `String` DI JAVA

## PARTE STATICA

- Per attenuare il problema si può *sfruttare la doppia natura del costrutto class* che, in Java e C# può far coesistere:
  - la definizione di tipo (non statica)
  - una *parte statica* di libreria tradizionale
- La libreria accessoria può essere quindi *incorporata* nella stessa classe `String` che definisce il tipo
  - unitarietà nella differenza ☺
  - MA ognuno mantiene il suo ruolo:
    - le funzioni statiche si invocano nella forma `String.nomefunzione` (come nella libreria matematica: `Math.sin(...)`, `Math.abs(...)`)
    - i metodi si invocano in stile OOP su uno specifico oggetto-stringa (ad esempio, per l'oggetto `s`, `s.toUpperCase()`, `s.length()`, etc.)





# LA CLASSE String DI JAVA

## VISIONE COMPLETA

```
char charAt(int index)
int length()
int compareTo(Object o)
int compareTo(String s)
int compareToIgnoreCase(String s)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String s)
int indexOf(int ch)
int indexOf(int ch, int from)
int indexOf(String s)
int indexOf(String s, int from)
int lastIndexOf(int ch)
int lastIndexOf(int ch, int from)
int lastIndexOf(String s)
int lastIndexOf(String s, int i)
String replace(char c1, char c2)
String toLowerCase()
String toLowerCase(Locale locale)
String toUpperCase()
String toUpperCase(Locale locale)
```

```
String trim()
boolean endsWith(String s)
boolean startsWith(String p)
boolean startsWith(String p,
                    int offset)
String substring(int beginIndex)
String substring(int beginIndex,
                int endIndex)
```

### Operazioni accessorie statiche

```
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(char[] data)
static String valueOf(char[] data,
                    int offset, int count)
static String valueOf(double d)
static String valueOf(float f)
static String valueOf(int i)
static String valueOf(long l)
static String valueOf(Object obj)
...
```

# JAVA: L'ESEMPIO RIVISTO

- Codice precedente, con nostra libreria separata:

```
int a = 35; double d = 3.14;  
String sA = StringLibrary.convert(a);  
String sD = StringLibrary.convert(d);  
  
System.out.println("Primo valore: " + sA);  
System.out.println("Secondo valore: " + sD);
```

Java

Java: necessario  
usare funzioni  
statiche di libreria

- Soluzione effettiva, con *funzioni statiche di String*

```
int a = 35; double d = 3.14;  
String sA = String.valueOf(a);  
String sD = String.valueOf(d);  
  
System.out.println("Primo valore: " + sA);  
System.out.println("Secondo valore: " + sD);
```

Java

Java: necessario  
usare funzioni  
statiche di **String**



# JAVA: ESEMPIO COMPLESSIVO

```
String s1 = "Nel mezzo del cammin";  
String s2 = s1.replace('z', 'Z');  
System.out.println( s1.equals(s2) );  
System.out.println( s1.equalsIgnoreCase(s2) );  
System.out.println( s2.toLowerCase() );  
System.out.println( s1.startsWith("Del") );  
System.out.println( s1.startsWith("Nel") );
```

Invocazione di metodi  
su oggetti **String**

```
String s3 = String.valueOf(2.14);  
String s4 = String.valueOf(1.44F);
```

In Java, i numeri sono valori  
primitivi: non si può invocare  
un metodo su di loro!

Per questo, **valueOf** può essere solo una  
"classica" funzione statica di libreria

... che però ha senso mettere nella parte statica  
della classe **String** per *attinenza concettuale*



# JAVA: ESEMPIO COMPLESSIVO

```
String s1 = "Nel mezzo del cammin";  
String s2 = s1.replace('z', 'Z');  
System.out.println( s1.equals(s2) );  
System.out.println( s1.equalsIgnoreCase(s2) );  
System.out.println( s2.toLowerCase() );  
System.out.println( s1.startsWith("Del") );  
System.out.println( s1.startsWith("Nel") );  
  
String s3 = String.valueOf(2.14);  
String s4 = String.valueOf(1.44F);  
System.out.println(s3);  
System.out.println(s4);
```

Invocazione di metodi  
su oggetti `String`

```
false  
true  
nel mezzo del cammin  
false  
true  
2.14  
1.44
```



# JAVA: FORMATTAZIONI `printf`-like

- Le operazioni accessorie statiche offrono anche la **funzione `format`**, che consente di formattare dati *in modo molto simile alla `printf` del C*
  - facilita la conversione di programmi C
  - supporta la stampa di dati in colonne (tabelle varie)

```
static String format(String format, ...)
```

- È una funzione a ***numero variabile di argomenti***
  - il primo è la stringa di formato  
template: **`%`**[argument\_index\$][flags][width][.precision]**conversion**  
esempi: **`%s`**, **`%d`**, **`%10s`**, **`%-15s`**, **`%5.2d`**, ...
  - i successivi, gli oggetti da stampare



# JAVA: FORMATTAZIONI printf-like

- Ci sono *decine e decine* di specifiche e varianti
  - per i dettagli vedere la classe `java.util.Formatter`

Conversion	Argument Category	Description
'b', 'B'	general	If the argument <i>arg</i> is null, then the result is "false". If <i>arg</i> is a boolean or
'h', 'H'	general	The result is obtained by invoking <code>Integer.toHexString(arg.hashCode())</code> .
's', 'S'	general	If <i>arg</i> implements <code>Formattable</code> , then <code>arg.formatTo</code> is invoked. Otherwise, t
'c', 'C'	character	The result is a Unicode character
'd'	integral	The result is formatted as a decimal integer
'o'	integral	The result is formatted as an octal integer
'x', 'X'	integral	The result is formatted as a hexadecimal integer
'e', 'E'	floating point	The result is formatted as a decimal number in computerized scientific notat
'f'	floating point	The result is formatted as a decimal number
'g', 'G'	floating point	The result is formatted using computerized scientific notation or decimal for
'a', 'A'	floating point	The result is formatted as a hexadecimal floating-point number with a signif argument category.
't', 'T'	date/time	Prefix for date and time conversion characters. See Date/Time Conversions.
'%'	percent	The result is a literal '%' ('\\u0025')
'n'	line separator	The result is the platform-specific line separator

Flag	General	Character	Integral	Floating Point	Date/Time	Description
','	y	y	y	y	y	The result will be left-justified.
'#'	y <sup>2</sup>	-	y <sup>3</sup>	y	-	The result should use a conversion-dependent alternate form
','	-	-	y <sup>4</sup>	y	-	The result will always include a sign
' '	-	-	y <sup>4</sup>	y	-	The result will include a leading space for positive values
'0'	-	-	y	y	-	The result will be zero-padded
'.'	-	-	y <sup>2</sup>	y <sup>5</sup>	-	The result will include locale-specific grouping separators
'('	-	-	y <sup>4</sup>	y <sup>5</sup>	-	The result will enclose negative numbers in parentheses

'n'	the platform-specific line separator as returned by <code>System.lineSeparator()</code> .
-----	---





# JAVA: FORMATTAZIONI printf-like

- Esempio

```
public static void main(String[] args) {  
    var s1 = String.format("%-20s canta la canzone %-35s %-4s",  
        "Francesca Michielin", "Nessun grado di separazione", "|");  
    System.out.println(s1);  
    var s2 = String.format("%-20s canta la canzone %-35s %-4s",  
        "Renato Zero", "I migliori anni della nostra vita", "|");  
    System.out.println(s2);  
}
```

----- Java Run -----

```
Francesca Michielin  canta la canzone Nessun grado di separazione      |  
Renato Zero         canta la canzone I migliori anni della nostra vita  |
```

- da Java 15, la stessa funzionalità è offerta anche sotto forma di *metodo* della classe String, tramite il nuovo metodo ***formatted***

***s.formatted(a,b,c)*** equivale a ***String.format(s,a,b,c)***

```
var s1 = "%-20s canta la canzone %-35s %-4s".formatted(  
    "Francesca Michielin", "Nessun grado di separazione", "|");  
System.out.println(s1);  
var s2 = "%-20s canta la canzone %-35s %-4s".formatted(  
    "Renato Zero", "I migliori anni della nostra vita", "|");  
System.out.println(s2);
```

# FORMATTAZIONI NEGLI ALTRI LINGUAGGI

- C# offre la funzione statica `string.Format`, che supporta anche interpolazione e specifiche di allineamento:

```
public static void Main()
{
    var s1 = string.Format("{0,-20} canta la canzone {1,-35}{2,-4} \n ", "Francesca Michielin", "Nessun grado di separazione", "|");
    System.Console.WriteLine(s1);
    var s2 = string.Format("{0,-20} canta la canzone {1,-35}{2,-4} \n ", "Renato Zero", "I migliori anni della nostra vita", "|");
    System.Console.WriteLine(s2);
}
```

```
Francesca Michielin  canta la canzone Nessun grado di separazione  |
Renato Zero         canta la canzone I migliori anni della nostra vita  |
```

- Scala e Kotlin sono analoghi, ma `format` è un metodo della classe `String` e il formato è il target:

```
def main(args: Array[String]) : Unit = {
    val formato = "%1$-20s canta la canzone %2$-35s %3$-4s"
    var s1 = formato.format("Francesca Michielin", "Nessun grado di separazione", "|");
    var s2 = formato.format("Renato Zero", "I migliori anni della nostra vita", "|");
    println(s1);
    println(s2);
}
```

```
Francesca Michielin  canta la canzone Nessun grado di separazione  |
Renato Zero         canta la canzone I migliori anni della nostra vita  |
```

# UN APPROCCIO ALTERNATIVO: INTERPOLATORI in Scala

- Scala introduce anche gli **interpolatori**
  - l'interpolatore **f** funge da prefisso per stringhe printf-like
  - l'interpolatore **s** funge da prefisso per stringhe semplici con variabili

```
2 def main(args: Array[String]) : Unit = {  
3   var cantante = "Francesca Michielin";  
4   var canzone = "Nessun grado di separazione"  
5   var barra = "|"  
6   // ----- come prima -----  
7   val formato = "%1$-20s canta |la canzone %2$-35s %3$-4s"  
8   var s1 = formato.format(cantante, canzone, barra);  
9   println(s1);  
10  // ----- con interpolatore f -----  
11  println(f"$cantante%-20s canta la canzone $canzone%-35s $barra%-4s")  
12 }
```

```
Francesca Michielin  canta la canzone Nessun grado di separazione |  
Francesca Michielin  canta la canzone Nessun grado di separazione |
```

Da Scala 3, promosso ad  
approccio preferito  
rispetto alla concatena-  
zione col + (deprecata)

```
// ----- con interpolatore f -----  
println(f"$cantante%-20s canta la canzone $canzone%-35s $barra%-4s")  
// ----- con interpolatore s -----  
println(s"$cantante canta la canzone $canzone $barra")  
}
```

```
Francesca Michielin  canta la canzone Nessun grado di separazione |  
Francesca Michielin  canta la canzone Nessun grado di separazione |
```



# UN PROBLEMA DI PRESTAZIONI

## DA `String` A `StringBuilder`

---

- La **concatenazione di stringhe con l'operatore `+`** è comoda, leggibile e facile da ricordare...
- ... MA producendo ogni volta nuove stringhe ***non è efficiente se l'operazione è ripetuta***, come in un ciclo
- Per questo Java, C#, Scala, Kotlin offrono la classe **`StringBuilder`** che è un ***contenitore per stringhe***
  - non è un *valore stringa* (immodificabile), è un *contenitore* (modificabile)
  - a differenza di `String`, è progettata per *rendere molto efficiente la modifica* della stringa contenuta
  - metodo più importante: **`append`** (in C#: **`Append`**)



# ESEMPI A CONFRONTO (1/3)

Concatenazione semplicistica con l'operatore + di `String`:

```
String s = "";  
for(int i=0; i<1000; i++)  
    s += "La pioggia agli irti colli";  
System.out.println(s);
```

Java

A ogni iterazione, l'operatore +

- genera una nuova stringa
- ricopia al suo interno il contenuto delle due da concatenare
- abbandona la precedente al garbage collector



# ESEMPI A CONFRONTO (2/3)

---

Concatenazione smart con `StringBuilder`:

```
StringBuilder sb = new StringBuilder();  
for(int i=0; i<1000; i++)  
    sb.append("La pioggia agli irti colli");  
System.out.println(s);
```

Java

Stavolta, a ogni iterazione il metodo **append**

- appende al buffer interno il contenuto della nuova stringa
- (solo se necessario allarga dinamicamente il buffer)



# ESEMPI A CONFRONTO (3/3)

Sarà davvero più efficiente?

- *Misuriamo e confrontiamo i tempi di esecuzione (approx)*
- Sfruttiamo la funzione statica `System.currentTimeMillis`

```
var t0 = System.currentTimeMillis();  
// codice del ciclo  
var t1 = System.currentTimeMillis();  
System.out.println(t1-t0);
```

Java

Scala

Kotlin

Java

Output

```
1000 StringBuilder appends: 1  
1000 string concatenations: 14
```

```
1000 StringBuilder appends: 0  
1000 string concatenations: 15
```

C#

Probabile ottenere  
tempi diversi con  
JDK e PC diversi

Kotlin

```
1000 StringBuilder append: 0  
1000 string concatenation: 38
```

```
1000 StringBuilder append: 3  
1000 string concatenation: 186
```

Scala



# OLTRE `StringBuilder` : `StringJoiner`

- Un uso tipico di `StringBuilder` è in un ciclo, per concatenare insieme di stringhe *separandole con qualcosa*
  - ESEMPIO: concatenare parole separandole con virgole
- Usare `StringBuilder` è possibile, ma costringe:
  - o a tenersi la virgola *anche dopo l'ultimo termine* (come qui sotto)
  - o a complicare l'algoritmo per trattare diversamente l'ultimo termine

```
public static void main (String[] args){  
    StringBuilder sb = new StringBuilder();  
    for(int i=0; i<args.length; i++) {  
        sb.append(args[i]); sb.append(", ");  
    }  
    System.out.println(sb);  
}
```

Java

Virgola sgradevole  
dopo l'ultimo termine

alfa, beta, gamma, delta,





# OLTRE `StringBuilder` : `StringJoiner`

---

- Perché complicarsi la vita?
- Concatenare stringhe in un ciclo, *separandole con una stringa a scelta*, ***è un'esigenza tipica di molte situazioni***  
→ c'è una classe già pronta per farlo: ***StringJoiner***
- Java: la classe **StringJoiner**
  - è un contenitore-concatenatore di stringhe
  - funziona sostanzialmente come `StringBuilder`, ma offre *metodi particolarmente comodi per concatenare stringhe in modo flessibile*
  - il costruttore riceve la *stringa di separazione* desiderata (volendo, anche un prefisso iniziale e un suffisso finale)
  - metodo chiave: **add**



# OLTRE StringBuilder : StringJoiner

- Con **StringJoiner**, concatenare insiemi di stringhe separandole con una stringa data è *una banalità*
  - il costruttore riceve la *stringa di separazione* desiderata (volendo, anche un prefisso iniziale e un suffisso finale)
  - si aggiungono via via le singole stringhe con il metodo **add**

```
public static void main (String[] args){  
    StringJoiner sj = new StringJoiner(" , ");  
    for(int i=0; i<args.length; i++) {  
        sj.add(args[i]);  
    }  
    System.out.println(sj);  
}
```

Java

Nessuna virgola o  
spazio extra dopo  
l'ultimo termine

**alfa , beta , gamma , delta**



# OLTRE StringBuilder : StringJoiner

- Con **StringJoiner**, concatenare insiemi di stringhe separandole con una stringa data è *una banalità*
  - il costruttore riceve la *stringa di separazione* desiderata (volendo, anche un prefisso iniziale e un suffisso finale)
  - si aggiungono via via le singole stringhe con il metodo **add**

```
public static void main (String[] args){  
    StringJoiner sj = new StringJoiner(", ", "(", ")");  
    for(int i=0; i<args.length; i++) {  
        sj.add(args[i]);  
    }  
    System.out.println(sj);  
}
```

Prefisso  
iniziale

Java

Suffisso  
finale

(alfa, beta, gamma, delta)



# O ANCORA PIÙ SEMPLICEMENTE..

- Il caso semplice in cui **StringJoiner** è usato solo col separatore (senza prefisso e suffisso) è sostituibile più brevemente con la funzione statica **String.join**

```
public static void main (String[] args){  
    var result = String.join("", " , args) ;  
    System.out.println(result) ;  
}
```

Java

alfa, beta, gamma, delta

- Questo approccio è seguito anche in C#:

```
public static void Main()  
{  
    string[] frasi = {"La pioggia", "agli", "irti", "colli"};  
    var result = System.String.Join("", frasi);  
    System.Console.WriteLine(result);  
}
```

C#

# O ANCORA PIÙ SEMPLICEMENTE..

- Scala e Kotlin introducono invece un apposito metodo (**mkString** / **joinToString**) nella classe array:

```
object Prova{  
  def main(args: Array[String]) : Unit = {  
    var frasi = Array("La pioggia", "agli", "irti", "colli");  
    var result = frasi.mkString(",");  
    println(result);  
  }  
}
```

```
object Prova{  
  def main(args: Array[String]) : Unit = {  
    var frasi = Array("La pioggia", "agli", "irti", "colli");  
    var result = frasi.mkString("(", ", ", ", ", ")");  
    println(result);  
  }  
}
```

Scala

```
fun main() {  
  var frasi = arrayOf("La pioggia", "agli", "irti", "colli");  
  var result = frasi.joinToString(",");  
  println(result);  
}
```

```
fun main() {  
  var frasi = arrayOf("La pioggia", "agli", "irti", "colli");  
  var result = frasi.joinToString(", ", "(", ", ")");  
  println(result);  
}
```

Kotlin

Occhio all'ordine degli argomenti!

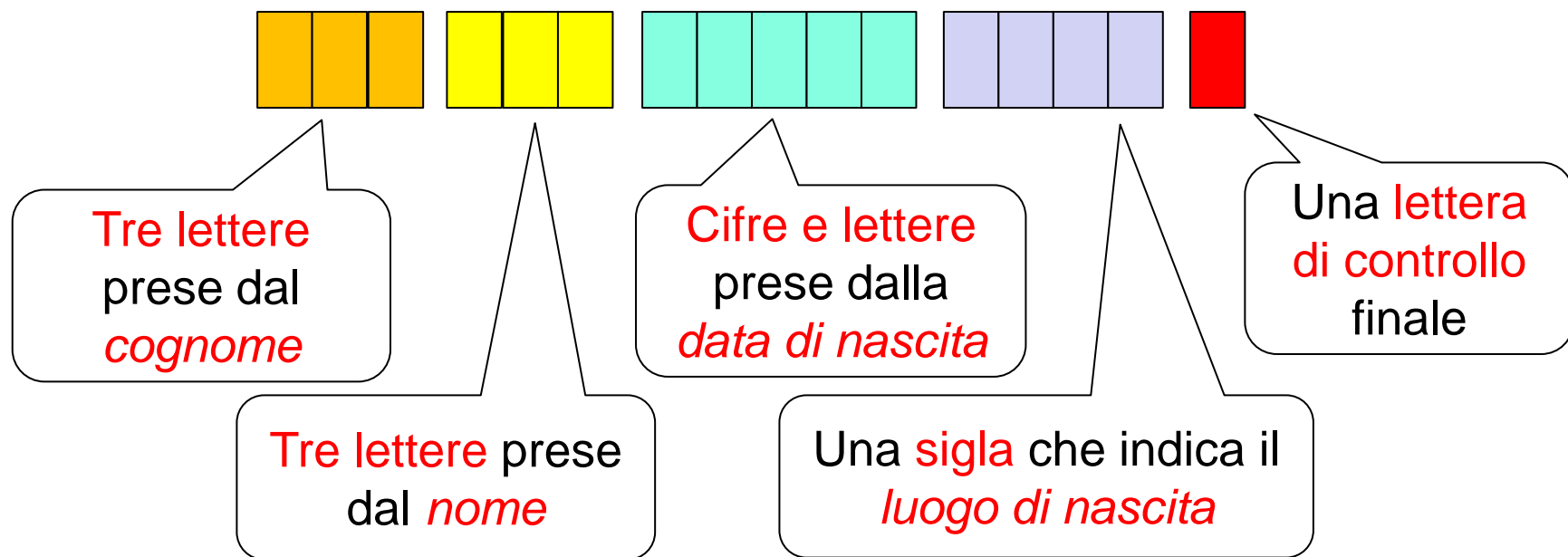
In Kotlin è diverso da Java e Scala

# UN PROBLEMA CONCRETO: calcolo e verifica del CODICE FISCALE

# SAPETE COS'È IL CODICE FISCALE?

- Il codice fiscale serve a *identificare in modo univoco ogni persona residente* in Italia
- .. o anche non residente, se deve pagare tasse 😊

Il codice è fatto da 16 lettere e cifre, divise in 5 gruppi:



# COME SI SCELGONO ?

Dato che il codice serve a identificare una persona:

- bisogna *cercare di non generare due codici uguali*
- è importante *accorgersi di eventuali errori*  
(lettere scambiate di posto, lettere sbagliate, ecc.)

- Per questo, è la legge (D.M. 23/12/1976) a stabilire *come si scelgono* quelle lettere e cifre.
- IDEE:
  - scegliere *lettere diverse* nei *nomi corti* e nei *nomi lunghi*
  - *distinguere* in qualche modo *uomini* e *donne*
- Inoltre, la lettera finale è il risultato di un *calcolo* su tutte le lettere e i numeri precedenti.

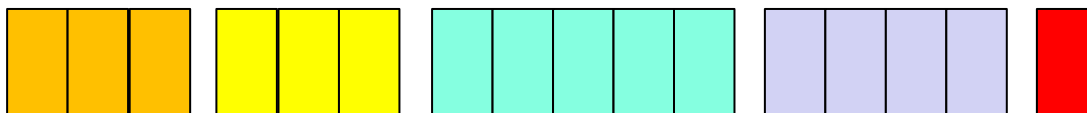




# LE LETTERE DEL COGNOME

Per il **cognome** si prendono:

- innanzitutto, **le prime tre consonanti**
- se non bastano (cioè sono meno di tre), **le vocali**
- se non bastano ancora, *si mette una "X" alla fine*



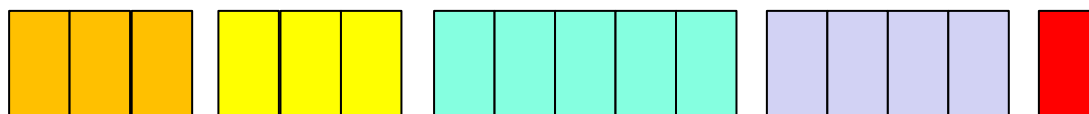
Esempi

- **ROSSI** → **prime tre consonanti** → **R S S**
- **AIELLO** → **ha solo due consonanti, serve una vocale**  
→ **L L A**
- **RE** → **ha solo una consonante e solo una vocale**  
si aggiunge una X → **R E X**

# LE LETTERE DEL NOME

Per il **nome** si procede QUASI come per il cognome:

- se il nome è lungo (cioè **ha quattro o più consonanti**) si prendono **la prima, la terza e la quarta**
- se il nome è corto (cioè **ha tre consonanti o meno**) si fa come per il cognome (**si prendono le prime tre**)
- se non bastano (cioè sono meno di tre), **le vocali**
- se non bastano ancora, *si mette una "X" alla fine*

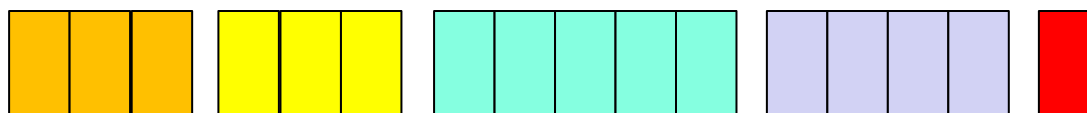


- ALBERTO → 1<sup>a</sup>, 3<sup>a</sup>, 4<sup>a</sup> consonante → L R T
- ENRICO → prime tre consonanti → N R C
- ELISA → due consonanti + vocale → L S E

# LE LETTERE DEL NOME

Perché questo?

- perché così *nomi simili* producono *codici diversi*
- importante, come detto, per non "confondere" una persona con un'altra dal nome simile

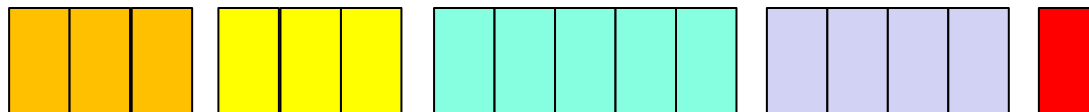


- ANGELO → prime tre consonanti → N G L
  - ANGELINO → 1<sup>a</sup>, 3<sup>a</sup>, 4<sup>a</sup> consonante → N L N
  - PIERLUIGI → 1<sup>a</sup>, 3<sup>a</sup>, 4<sup>a</sup> consonante → P L G
  - PIERLUCA → 1<sup>a</sup>, 3<sup>a</sup>, 4<sup>a</sup> consonante → P L C
- se avessimo preso le prime tre, sarebbero risultati identici!

# LA DATA DI NASCITA

Il **terzo blocco** rappresenta la **data di nascita**:

- **l'anno** su due cifre
- **il mese** codificato da una lettera *scelta fra dodici che non si confondano fra loro (e con le cifre)*
- **il giorno** su due cifre, *aumentato di 40 per le donne* in modo da distinguere facilmente uomini e donne:
  - gli **uomini** hanno giorni compresi fra **01** e **31**
  - le **donne** hanno giorni compresi fra **41** e **71**



- uomo nato il 19/03/98 → 98 C 19
- donna nata il 19/03/98 → 98 C 59



# LA DATA DI NASCITA

Che lettere si usano per il mese?

- **NON le prime dodici** dell'alfabeto, perché alcune sono troppo simili fra loro (E, F) e altre assomigliano a numeri (I/1, O/0)
- ma bensì **queste dodici**, stabilite dalla legge:

**A, B, C, D, E, H, L, M, P, R, S, T**

secondo questa corrispondenza:

• gennaio	= A	luglio	= L
• febbraio	= B	agosto	= M
• marzo	= C	settembre	= P
• aprile	= D	ottobre	= R
• maggio	= E	novembre	= S
• giugno	= H	dicembre	= T

# LA SIGLA DEL LUOGO DI NASCITA

Il quarto blocco rappresenta il luogo di nascita:

- per i nati in Italia, il **comune** di nascita
- per i nati all'estero, lo **Stato estero** di nascita

Esiste una *lunga tabella* che li elenca tutti:

- i **comuni italiani** hanno sigle comprese fra **A001** e **V999**  
*Attualmente sono usate quelle fino a M431*
- gli **stati esteri** hanno sigle comprese fra **Z100** e **Z999**

*Curiosità: la prima cifra indica il continente*

- 1 = Europa
- 2 = Asia
- 3 = Africa
- 4 = America del nord
- 5 = America centrale
- 6 = America del sud
- 7 = Oceania
- 8 = terre polari
- 9 = terre polari

Z800	DIPENDENZE CANADESI	TERRE POLARI ARTICHE
Z801	DIPENDENZE NORVEGESI ARTICHE	TERRE POLARI ARTICHE
Z802	DIPENDENZE RUSSE	TERRE POLARI ARTICHE
Z900	DIPENDENZE AUSTRALIANE	TERRE POLARI ANTARTICHE
Z901	DIPENDENZE BRITANNICHE	TERRE POLARI ANTARTICHE
Z902	DIPENDENZE FRANCESI	TERRE POLARI ANTARTICHE
Z903	DIPENDENZE NEOZELANDESI	TERRE POLARI ANTARTICHE
Z904	DIPENDENZE NORVEGESI ANTARTICHE	TERRE POLARI ANTARTICHE
Z905	DIPENDENZE STATUNITENSI	TERRE POLARI ANTARTICHE
Z906	DIPENDENZE SUDAFRICANE	TERRE POLARI ANTARTICHE

# LE SIGLE DEI COMUNI

## Sigle dei comuni: criteri

- le sigle furono attribuite *originariamente* in ordine alfabetico da *A001 = Abano Terme (PD)* fino a *M206 = Zuri (CA)*
- la lista però è in continua evoluzione (cambi di nome, fusioni di comuni): i nuovi comuni sono aggiunti *in fondo alla lista*, i vecchi sono depennati
- attualmente (Febbraio 2022) l'ultimo è *M432 = Miliscemi (TP)*
- sono presenti (ovviamente!) anche comuni che *non sono più italiani* (Fiume, Zara) ma lo sono stati in passato: la Storia non si cancella!

Dettaglio	Codice Nazionale	Codice Istat	Codice Catastale	Sigla Provincia	Denominazione	Soppresso
<a href="#">i</a>	M149		F7AA	ZA	Zara	SI
<a href="#">i</a>	D620		F2AA	FU	Fiume	SI

## ESEMPI

- |                   |        |         |        |
|-------------------|--------|---------|--------|
| • Reggio Emilia   | → H223 | Bologna | → A944 |
| • Albania         | → Z100 | Cina    | → Z210 |
| • Marocco         | → Z330 | USA     | → Z404 |
| • Rep. dominicana | → Z505 | Brasile | → Z602 |

# SIETE CURIOSI ?

Trovate tutto sul sito dell'Agenzia delle Entrate!

Ti trovi in: [Home](#) / [Servizi](#) / [Ricerca codici tributo](#) / Tabelle codici Uffici finanziari, Regioni, Province e Comuni

## Tabelle codici Uffici finanziari, Regioni, Province e Comuni

[← Pagina Precedente](#)

- ▶ [Tabella dei codici delle Regioni e delle Province Autonome](#)
- ▶ [Tabella delle Province](#)
- ▶ [Tabella dei codici degli Enti Territoriali emittenti prestiti obbligazionari](#)
- ▶ [Tabella dei codici degli Uffici Finanziari e delle Direzioni Regionali](#)
- ▶ [Tabella dei codici dei Comuni](#)
- ▶ [Tabella dei Comuni per il pagamento dell'imposta di scopo](#)
- ▶ [Tabella dei Comuni per il pagamento della IMIS](#)
- ▶ [Tabella dei Comuni per il pagamento dell'imposta/contributo di solidariet ](#)

### Tabella dei codici dei Comuni

[← Pagina Precedente](#)

Che inizia per: [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [Z](#) **BOLO** [Cerca](#)

Codice Ente	Denominazione
<b>A944</b>	<b>BOLOGNA</b>
A945	BOLOGNANO
A946	BOLOGNETTA
A947	BOLOGNOLA
A948	BOLOTANA





# SIETE CURIOSI ?

Volendo, potete anche scaricare un archivio completo:

<https://www.agenziaentrate.gov.it/portale/web/guest/schede/fabbricatiterreni/archivio-comuni-e-stati-esteri/consultazione-archivio-comuni-stati-esteri>

Ti trovi in: [Home](#) / [Schede informative e servizi](#) / [Fabbricati e terreni](#) / [Archivio Comuni e Stati esteri](#) / Consultazione Archivio Comuni e Stati esteri

## ARCHIVIO COMUNI E STATI ESTERI

### INFORMAZIONI

Che cos'è

### SERVIZI

Consultazione Archivio Comuni e Stati esteri

Guida al servizio

## Consultazione Archivio Comuni e Stati esteri

Il **Codice dei Comuni d'Italia e degli Stati esteri** è stato istituito per rendere possibile l'espressione in forma abbreviata delle denominazioni dei Comuni d'Italia. Nato in ambito catastale, il codice è stato adottato per la codifica amministrativa dei Comuni e utilizzato, tra l'altro, anche nella composizione del Codice Fiscale.

Il codice è stato istituito attribuendolo ai Comuni in ordine alfabetico, partendo dal gruppo A001 e proseguendo in sequenza numerica sino ad A999 cui segue il gruppo B001 sino a B999, e così via. In fase di impianto, sono stati utilizzati i gruppi alfanumerici da A001 a M206; la codifica dei Comuni istituiti successivamente ha rispettato l'ordine sequenziale dei codici, ma non l'ordine alfabetico della denominazione.

Per le persone nate all'estero, la necessità di prevedere una analoga codifica del luogo di nascita da inserire nella composizione del codice fiscale, ha portato alla definizione di una specifica codifica, limitata all'attribuzione di un codice univoco per ogni Stato estero, senza ulteriore livello di dettaglio se non, in taluni casi, riguardo territori e dipendenze amministrati da Stati sovrani. Per gli Stati esteri la codifica è composta dalla lettera Z e da tre cifre (la prima è riferita al continente).

Il **Codice dei Comuni d'Italia e degli Stati esteri** si può consultare facilmente tramite la ricerca puntuale per provincia, per denominazione, per codice, per ufficio di competenza (nel caso degli Stati esteri, per continente, denominazione, codice Stato). Le informazioni sono disponibili anche in forma di elenco riepilogativo, che descrive le situazioni attuali (Comuni attuali, elenco delle variazioni, elenco degli Stati esteri).

[Accedi al servizio](#)

**i** Stai per scaricare dal sito dell'Agenzia delle Entrate un file firmato digitalmente o verificabile tramite hash. La firma digitale o l'hash del file ne garantiscono la provenienza e l'integrità. Le istruzioni per la verifica del file sono disponibili in questa pagina: [Verifica firma software](#)

- [Scarica l'Archivio Comuni aggiornato al 04/01/2022 – exe \(file zip autoestraente\)](#)
- [Scarica l'Archivio Comuni aggiornato al 04/01/2022 - link alternativo – exe \(file zip autoestraente\)](#)

# SIETE CURIOSI ?

Esempio: stati esteri attuali

1	lice Nazior	Denominazione	Stato Sov	Continente				
2	Z100	ALBANIA		220	Z714	NIUE	NZ	OCEANIA
3	Z101	ANDORRA		221	Z715	NORFOLK (ISOLE E ISOLE DEL MAR DEI CORAL	AUS	OCEANIA
4	Z102	AUSTRIA		222	Z716	NUOVA CALEDONIA	F	OCEANIA
5	Z103	BELGIO		223	Z719	NUOVA ZELANDA		OCEANIA
6	Z104	BULGARIA		224	Z721	ISOLE CILENE (PASQUA E SALA Y GOMEZ)	RCH	OCEANIA
7	Z106	STATO CITTA' DEL VATICANO		225	Z722	ISOLE PITCAIRN	GB	OCEANIA
8	Z107	DANIMARCA		226	Z723	POLINESIA FRANCESE (ISOLE)	F	OCEANIA
9	Z108	ISOLE FAER OER	DK	227	Z724	ISOLE SALOMONE		OCEANIA
10	Z109	FINLANDIA		228	Z725	SAMOA AMERICANE (ISOLE)	USA	OCEANIA
11	Z110	FRANCIA		229	Z726	SAMOA		OCEANIA
12	Z112	GERMANIA		230	Z727	TOKELAU O ISOLE DELL'UNIONE	NZ	OCEANIA
13	Z113	GIBILTERRA	GB	231	Z728	TONGA		OCEANIA
14	Z114	REGNO UNITO		232	Z729	ISOLE WALLIS E FUTUNA	F	OCEANIA
15	Z115	GRECIA		233	Z730	PAPUA NUOVA GUINEA		OCEANIA
16	Z116	IRLANDA		234	Z731	KIRIBATI		OCEANIA
17	Z117	ISLANDA		235	Z732	TUVALU		OCEANIA
18	Z119	LIECHTENSTEIN		236	Z733	VANUATU		OCEANIA
19	Z120	LUSSEMBURGO		237	Z734	PALAU		OCEANIA
20	Z121	MALTA		238	Z735	STATI FEDERATI DI MICRONESIA		OCEANIA
21	Z122	ISOLA DI MAN	GB	239	Z800	DIPENDENZE CANADESI		TERRE POLARI ARTICHE
22	Z123	MONACO		240	Z801	DIPENDENZE NORVEGESI ARTICHE		TERRE POLARI ARTICHE
23	Z124	NORMANNE (ISOLE) O ISOLE DEL CANALE	GB	241	Z802	DIPENDENZE RUSSE		TERRE POLARI ARTICHE
24	Z125	NORVEGIA		242	Z900	DIPENDENZE AUSTRALIANE		TERRE POLARI ANTARTICHE
				243	Z901	DIPENDENZE BRITANNICHE		TERRE POLARI ANTARTICHE
				244	Z902	DIPENDENZE FRANCESI		TERRE POLARI ANTARTICHE
				245	Z903	DIPENDENZE NEOZELANDESI		TERRE POLARI ANTARTICHE
				246	Z904	DIPENDENZE NORVEGESI ANTARTICHE		TERRE POLARI ANTARTICHE
				247	Z905	DIPENDENZE STATUNITENSI		TERRE POLARI ANTARTICHE
				248	Z906	DIPENDENZE SUDAFRICANE		TERRE POLARI ANTARTICHE
				249	Z907	SUD SUDAN		AFRICA



# SIETE CURIOSI ?

Ad esempio, in provincia di Bologna, dal 2014 ad oggi:

**Dettaglio Variazione Comuni**

<b>Tipo Variazione</b>	Costituzione
<b>Data Variazione</b>	01/01/2016
<b>Estremi Provvedimento</b>	Legge Regionale del 23 novembre 2015 n.19 Parte Prima del B.U.R.E.R n303
<b>Contenuto Provvedimento</b>	costituzione del comune di Alto Reno Terme mediante la fusione dei comuni di Granaglione e Porretta Terme
<b>Note VCT:</b>	

Indietro

**Dettaglio Variazione**

Codice Nazionale	Provincia	Denominazione Italiana	Denominazione Estera	Variazione
M369	BO	ALTO RENO TERME		Costituito in data 01-GEN-16
A558	BO	PORRETTA TERME		Aggregato
E135	BO	GRANAGLIONE		Aggregato



# SIETE CURIOSI ?

Ad esempio, in provincia di Bologna, dal 2014 ad oggi:

Ti trovi in: [Home](#) / [Schede informative e servizi](#) / [Fabbricati e terreni](#) / [Archivio Comuni e Stati esteri](#) / [Consultazione Archivio Comuni e Stati Esteri](#) / [Ricerca Comuni](#) / [Lista Comuni](#)

## Consultazione Archivio Comuni e Stati Esteri

Ricerca Comuni con denominazione 'Bazzano' (1 elemento)

Uffici competenti e variaz. amministrative	Codice Nazionale	Codice Istat	Codice Catastale	Sigla Provincia	Denominazione	Soppresso	Tariffe d'estimo e dati statistici
	A726		H1AF	BO	Bazzano	SI	

Variazioni

Aggregazione

Data Variazione: 01/01/2014

Codice Nazionale	Provincia	Denominazione Italiana	Denominazione Estera	Variazione
A726	BO	BAZZANO		Aggregato a VALSAMOGGIA(M320)

# ESEMPI COMPLETI

ROSSI MARIO, nato il 12/6/76 a Bologna

R S S   M R A   7 6 H 1 2   A 9 4 4   ?

Ancora non lo sappiamo

VERDI LUCIA, nata il 25/12/98 a Reggio Emilia

V R D   L C U   9 8 T 6 5   H 2 2 3   ?

Ancora non lo sappiamo

..e tu?

    ?



# LA LETTERA FINALE DI CONTROLLO

Questa lettera è diversa dalle altre:

- non deriva da un singolo dato della persona
- ma è il *risultato di un calcolo su tutte le lettere e le cifre precedenti*, con lo scopo di *evidenziare errori*

## APPROCCIO

- si *attribuisce un valore* a ogni lettera o cifra
- si *sommano* tali valori
- si prende la *lettera "corrispondente"* al risultato

## ATTENZIONE, PERÒ:

- il valore di ogni lettera o cifra *non è 1,2,3,4..* come ci si potrebbe aspettare, perché *non ci si accorgerebbe di eventuali scambi di posizione* (la somma è commutativa!)

# IL PROBLEMA

Perché non dare semplicemente dei valori alle lettere ?

- se stabilissimo che A=1, B=2, C=3, ecc., la somma su un codice corretto verrebbe *identica* alla somma su codici *simili MA sbagliati !*

ESEMPIO: Verdi Lucia (*per comodità solo cognome e nome*)

- Codice corretto: V R D L C U
- Codice sbagliato: V D R L C U
- Codice sbagliato: V R D L U C

MA con la semplice somma il risultato sarebbe *identico*

- in qualsiasi modo si attribuisse un valore alle lettere,
- *gli scambi di posizione (l'errore più comune!) non verrebbero rilevati*  
→ NON VA BENE!



# LA SOLUZIONE

**OBIETTIVO:** accorgersi degli scambi di posizione

- se si parte dall'idea di "sommare i valori delle lettere", essendo la somma intrinsecamente commutativa...
- l'unico modo per avere *risultati diversi* nel codice corretto e nei codici sbagliati è *dare valore alle lettere tenendo conto della loro posizione*

La legge stabilisce perciò che:

- a ogni lettera sia attribuito un valore fra 0 e 25
- le lettere *di posto pari* (2<sup>a</sup>, 4<sup>a</sup>, 6<sup>a</sup>) seguano l'ordine
- le lettere *di posto dispari* (1<sup>a</sup>, 3<sup>a</sup>, 5<sup>a</sup>) invece NO

In questo modo, se due lettere si scambiano, *il loro valore nella somma cambia* e quindi il risultato viene diverso: l'errore viene rilevato.



# VALORE DELLE LETTERE

## VALORI USATI

- a ogni lettera dell'*alfabeto inglese* (da A a Z) è attribuito un valore fra 0 e 25, ma *non necessariamente seguendo l'ordine alfabetico*

## VALORE DELLE LETTERE

- alle lettere *di posto pari* (2<sup>a</sup>, 4<sup>a</sup>, 6<sup>a</sup>) si attribuisce un valore *seguendo l'ordine alfabetico*: A = 0, B = 1, C = 2, ... fino a Z = 25
- alle lettere *di posto dispari* (1<sup>a</sup>, 3<sup>a</sup>, 5<sup>a</sup>) si attribuisce invece un valore seguendo una *speciale tabella*:

A	B	C	D	E	F	G	H	I	J	K	L	M
1	0	5	7	9	13	15	17	19	21	2	4	18
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
20	11	3	6	8	12	14	16	10	22	25	24	23



# METODO DI CALCOLO

## PASSO PRELIMINARE:

*ogni cifra fra 0 e 9 è sostituita da una lettera fra A e J*

Adesso il codice è composto di sole *lettere*.

## PROCEDIMENTO:

1. a ogni lettera è attribuito un valore fra 0 e 25  
a seconda della posizione (pari o dispari)
2. si somma tutto
3. *si divide il risultato per 26 e si prende il resto,*  
*che è sicuramente un numero fra 0 e 25*
4. *si prende la lettera corrispondente a tale resto,* secondo il normale  
ordine alfabetico (A=0, B=1, ecc.)

# ESEMPIO 1

ROSSI MARIO, nato il 12/6/76 a Bologna

R S S M R A 7 6 H 1 2 A 9 4 4 ?

Passo preliminare: sostituzione cifre 0-9 con lettere A-J

R S S M R A H G H B C A J E E

Attribuzione valori alle lettere:

R	S	S	M	R	A	H	G	H	B	C	A	J	E	E
	18		12		0		6		1		0		4	
8		12		8		17		17		5		21		9

Somma: 138  $\rightarrow$   $138/26 = 5$  con resto 8  $\rightarrow$  Lettera finale: I

R S S M R A 7 6 H 1 2 A 9 4 4 I

## ESEMPIO 2

VERDI LUCIA, nata il 25/12/98 a Reggio Emilia

VRD LCU 98T65 H223 ?

Passo preliminare: sostituzione cifre 0-9 con lettere A-J

V R D L C U J I T G F H C C D

Attribuzione valori alle lettere:

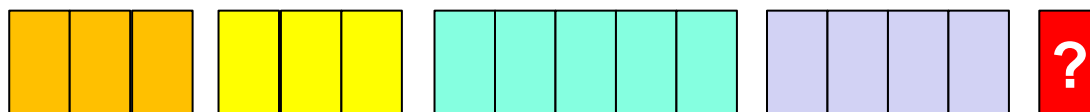
V	R	D	L	C	U	J	I	T	G	F	H	C	C	D
	17		11		20		8		6		7		2	
10		7		5		21		14		13		5		7

Somma: 153  $\rightarrow 153/26 = 5$  con resto 23  $\rightarrow$  Lettera finale: X

VRD LCU 98T65 H223 X

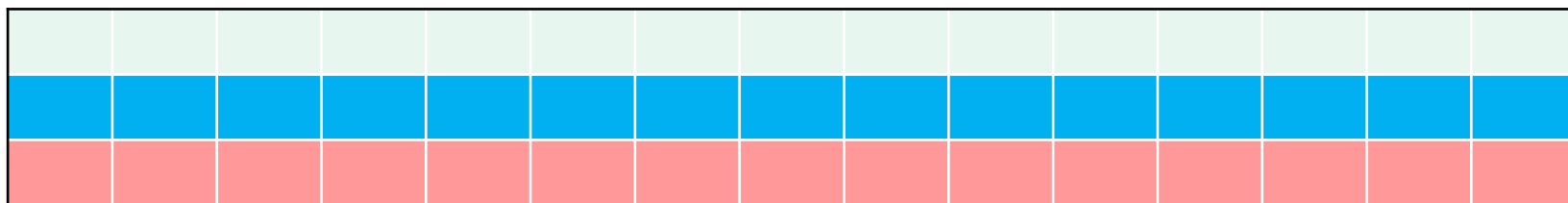
# ... E TU ? 😊

cognome nome, nato/a il \_\_\_\_\_ a \_\_\_\_\_

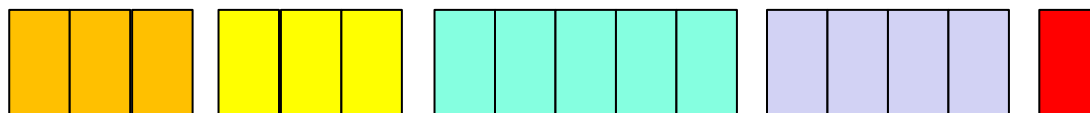


Passo preliminare: sostituzione cifre 0-9 con lettere A-J

Attribuzione valori alle lettere:



Somma: \_\_\_\_\_  $\rightarrow$  \_\_\_\_\_/26 = \_ con resto \_\_\_\_\_  $\rightarrow$  Lettera finale: \_



# UN PROBLEMA: OMOCODIA

Nonostante gli sforzi, possono risultare *codici identici* se due persone:

- hanno "quasi" lo stesso cognome e nome
- sono nate nello stesso giorno/mese/anno (su due cifre..) e luogo
- sono dello stesso sesso

**Improbabile? NO!**

- si stima ci siano già oltre 34.000 casi + 1000/1400 l'anno (stima)
- in ulteriore aumento, in particolare riguardo a persone straniere
  - giorno di nascita a volte incerto → attribuito il 1° gennaio
  - in certe zone del mondo, certi cognomi/nomi sono estremamente frequenti

**Argh!! E allora...?**

- come si fa ad accorgersene?
- cosa si fa se quel codice esiste già??

IL SOLO MODO è controllare  
sull'*Anagrafe Tributaria*

Si cambiano *entrambi* !

# OMOCODIA: SOLUZIONE

Come si risolve l'omocodia?

- si dà a ENTRAMBI un NUOVO CODICE *diverso dallo standard*
- *Trasparenza: entrambi devono sapere di avere un omocodico*
- il nuovo codice è ottenuto *sostituendo una o più cifre numeriche del codice originale con lettere a partire da destra*: (DM 23/12/76)  
0/L 1/M 2/N 3/P 4/Q 5/R 6/S 7/T 8/U 9/V
- questo schema ammette 128 varianti → 128 omocodici
- dopo la sostituzione, *il carattere finale di controllo va ricalcolato*

È a causa del rischio di omocodia che, per legge, *solo l'Agenzia delle Entrate può generare un codice fiscale*

- l'uso di un codice fiscale "fai da te" può creare *gravissimi problemi* (scambio di persona, errate fatturazioni, versamenti non riconosciuti..)
- *per questo, la generazione di codici fiscali "fai da te" è punita per legge*
- si può invece liberamente *verificare la correttezza formale di un codice*



# DUPLICE CODIFICA

---

Più raramente si presenta invece il problema opposto:

- Una località che era frazione di un comune diviene Comune autonomo  
→ le viene assegnato un nuovo codice comune
- Conseguente necessità di *rigenerare il codice fiscale degli abitanti*  
→ due codici fiscali diversi fanno riferimento alla stessa persona!





# CODICE FISCALE: VERIFICA

---

- Dal 2010 esiste un *servizio online dell'Agenzia delle Entrate* che verifica in modo ufficiale se un certo codice esiste ed è corretto
- Tale servizio è stato introdotto per legge:
  - Decreto legge n. 78 del 2010, convertito con modificazioni nella legge 122/2010: articolo 38, comma 6

*"Data la valenza del codice fiscale quale elemento identificativo di ogni soggetto, l'Amministrazione finanziaria rende disponibile a chiunque, con servizio di libero accesso, la possibilità di verificare, mediante i dati disponibili in Anagrafe Tributaria, l'esistenza e la corrispondenza tra il codice fiscale e i dati anagrafici"*

# AGENZIA DELLE ENTRATE

Cittadini Imprese Professionisti Intermediari Enti e Pa L'Agenzia

Area riservata

## Servizi Online

### Verifica codice fiscale

Verifica codice fiscale di persona fisica o di soggetto diverso da persona fisica

Verifica e corrispondenza tra il codice fiscale e i dati anagrafici di una persona fisica

Verifica e corrispondenza tra il codice fiscale e la denominazione di un soggetto diverso da persona fisica

Ti trovi in: [Home](#) / [Servizi](#) / Servizio di verifica del codice fiscale

## Servizio di verifica del codice fiscale

Il servizio permette di verificare l'esistenza e la corrispondenza tra i codici fiscali e quelli registrati in Anagrafe tributaria. Questa consultazione con [articolo 38, comma 6, primo paragrafo del dl 78/2010 - pdf](#) convertito in legge.

### Verifica codice fiscale di persona fisica o di soggetto diverso da persona fisica

Per eseguire la verifica occorre inserire il codice fiscale del soggetto. Se il soggetto è correttamente registrato in Anagrafe tributaria, il messaggio di risposta è "Codice fiscale valido". Il servizio verifica anche la validità di un codice fiscale provvisorio numerico attribuito a una persona fisica.

Il servizio non consente di verificare la validità di una partita Iva.

### Verifica e corrispondenza tra il codice fiscale e i dati anagrafici di una persona fisica

Il servizio verifica la validità e la corrispondenza tra un codice fiscale e i dati anagrafici completi di una persona fisica, attraverso il collegamento con l'Anagrafe tributaria. Per eseguire la verifica occorre indicare negli appositi campi il codice fiscale e i dati anagrafici completi del soggetto.

La verifica di corrispondenza viene effettuata sulla coincidenza di tutti i caratteri inseriti con quelli registrati in Anagrafe tributaria. Se vi è corrispondenza il messaggio di risposta è "Dati validi". L'applicazione verifica anche la validità di un codice fiscale provvisorio numerico e i dati anagrafici completi di una persona fisica.

Controlla se un certo codice *esiste ed è valido* (ma non rivela di chi sia)

Controlla se un certo codice *corrisponde a una data persona*

## Link correlati

- [Richiesta tessera sanitaria/codice fiscale](#)
- [Verifica partita Iva](#)



# AGENZIA DELLE ENTRATE

Ti trovi in: [Home](#) / [Servizi](#) / [Servizio di verifica del codice fiscale](#) / Verifica

## Verifica codice fiscale di persona fisica

Tutti i campi sono obbligatori

Codice fiscale:

Inserisci nel campo "Codice di sicurezza" i caratteri che vedi o che senti.



[audio](#)

[altra immagine](#)

[altro audio](#)

Codice di sicurezza:

Invia

Ripulisci

## Verifica e corrispondenza tra il codice fiscale e i dati anagrafici di una persona fisica

Tutti i campi sono obbligatori

Codice fiscale:

Dati anagrafici

Cognome:

Nome:

Data di nascita (gg/mm/aaaa):



Provincia di nascita:

Comune o stato estero di nascita:

Sesso:

☐ Maschio ☐ Femmina

Inserisci nel campo "Codice di sicurezza" i caratteri che vedi o che senti.



[audio](#)

[altra immagine](#)

[altro audio](#)

Codice di sicurezza:

Invia

Ripulisci



# OMOCODIA: ESEMPIO IN DETTAGLIO

Nel caso del nostro Mario Rossi:

- il suo codice "base" è, come sappiamo, **RSSMRA76H12A944I**
- in caso di omocodia, i codici alternativi si ottengono sostituendo una o più cifre numeriche *con lettere* a partire da destra

0/L    1/M    2/N    3/P    4/Q    5/R    6/S    7/T    8/U    9/V

Pertanto:

- 1<sup>a</sup> variante omocodica: **sostituzione del 1° carattere '4' con 'Q'**  
**RSSMRA76H12A944I → RSSMRA76H12A94Q?**
- 2<sup>a</sup> variante omocodica: **sostituzione del 2° carattere '4' con 'Q'**  
**RSSMRA76H12A944I → RSSMRA76H12A9Q4?**

Alla fine, la lettera finale di controllo va ricalcolata:

- 1<sup>a</sup> variante omocodica: **RSSMRA76H12A94QF**
- 2<sup>a</sup> variante omocodica: **RSSMRA76H12A9Q4U**

# OMOCODIA: TEST

Codice base del "nostro" Mario Rossi:

- **RSSMRA76H12A944I**

Alcune sue varianti omocodiche:

1. **RSSMRA76H12A94QF**
2. **RSSMRA76H12A9Q4U**
3. **RSSMRA76H12A9QQR**

Ottimo, vengono diversi anche i  
codici di controllo.  
Quindi, tutto bene...

Un altro Mario Rossi, nato a Milano il 1° gennaio 1990:

- **RSSMRA90A01F205Z**

Alcune sue varianti omocodiche:

- |                            |                            |
|----------------------------|----------------------------|
| 1. <b>RSSMRA90A01F20RU</b> | 5. <b>RSSMRA90AL1F205K</b> |
| 2. <b>RSSMRA90A01F2L5K</b> | 6. <b>RSSMRA9LA01F205K</b> |
| 3. <b>RSSMRA90A01FN05O</b> |                            |
| 4. <b>RSSMRAV0A01F205O</b> |                            |

...tutto bene *davvero*? È normale?

# CARATTERE DI CONTROLLO: LIMITI

Il carattere di controllo:

- è ottenuto dalla *somma pesata* dei caratteri precedenti
- grazie al diverso peso dei caratteri di posto pari rispetto a quelli di posto dispari, *rileva le inversioni di posizione*
- ma **non** gli scambi fra caratteri di posto pari (dispari) *fra loro* e **neppure** i cambiamenti che causino variazioni *nulle mod 26*

Nel nostro caso:

RSSMRA90AL1F205K

RSSMRA90A01F2L5K

RSSMRA9LA01F205K

RSSMRA90A01FN05O

RSSMRAV0A01F205O

- in tutti questi omocodici, uno **0** di posizione pari è rimpiazzato da **L**
- cambia la sostituzione, ma *non la somma !*

Qui invece:

- un **2** di posizione dispari è sostituito da **N**  
→ la somma aumenta di 15
- un **9** di posizione dispari è sostituito da **V**  
→ la somma diminuisce di 11

*Le due somme differiscono di 26... mod 26 = 0*

# CODICE FISCALE: PROGETTO (1)

- Scelta di fondo: fare un *componente software singleton*

- deve solo fare calcoli, come la libreria matematica:  
non ha bisogno di uno stato interno

In Java e C#, metodi statici  
In Scala e Kotlin, un object

- Interfaccia esterna:

- un metodo pubblico

**calcolaCodiceFiscale**

In Java e C#,  
statico

- argomenti: cognome, nome,  
giorno mese e anno di nascita,  
comune di nascita, sesso
    - valore restituito: il codice fiscale "base"

- un metodo pubblico

**verificaCodiceFiscale**

In Java e C#,  
statico

- argomenti: gli stessi di cui sopra + *il codice fiscale da verificare*
    - valore restituito: l'esito della verifica (boolean)



# CODICE FISCALE: PROGETTO (2)

- Organizzazione interna

In Java e C#,  
tutti statici

- **regola aurea: un metodo (privato) per ogni calcolo**
- `calcolaCognome`: prende in ingresso il *cognome* e restituisce tre lettere
- `calcolaNome`: prende in ingresso il *nome* e restituisce tre lettere
- `calcolaAnno`: prende in ingresso l'*anno* e restituisce le ultime due cifre (come stringa)
- `calcolaMese`: prende in ingresso il *mese* e restituisce il corrispondente carattere
- `calcolaGiornoSesso`: prende in ingresso il *giorno* e il  *Sesso* e restituisce due cifre (sotto forma di stringa)
- `calcolaComune`: prende in ingresso il *comune* o lo *stato estero* di nascita e restituisce il relativo codice alfanumerico
- `calcolaCarControllo`: prende in ingresso *tutto il codice* finora calcolato e restituisce il carattere di controllo corrispondente.





# CODICE FISCALE: PROGETTO (3)

- In questa organizzazione:
  - i metodi pubblici costituiscono la *"facciata"* del componente
  - lavorano coordinando il lavoro di altri metodi (non visibili)
- Questo tipo di struttura si dice **FAÇADE**
- È uno dei (molti) **pattern di progettazione**
  - per **pattern** si intende uno *schema tipico, un modello di soluzione standard* adatto a un preciso set di situazioni
  - rappresenta una *buona pratica, una soluzione di provato valore con precisi pro & contro*
  - ne incontreremo molti altri





# CODICE FISCALE: PROGETTO (4)

Come fare per..

- ..verificare se un carattere è vocale o consonante?
  - metodo `boolean isConsonante(char c)`
  - modo stupido: controllare ogni consonante.. (catena di `if`..) → ORRENDO  
modo appena meno stupido: controllare ogni vocale (sono meno..) → BAH
  - modo più furbo: sfruttare le stringhe e i relativi metodi!  
*un carattere è una vocale solo se appartiene alla stringa "AEIOU" 😊😊*  
(occhio a minuscole/maiuscole...) [dubbio: ... la Y? → Leggete la legge! 😊]
- ..ottenere il peso di ogni carattere, o il carattere del mese ?
  - modo stupido: catena di `if`.. → INGUARDABILE e INEFFICIENTE  
modo appena meno stupido: uno `switch` → MOLTO "BOVINO"...
  - modo più furbo: sfruttare le stringhe e i relativi metodi!  
*→ nella stringa "BAKPLCQDREVOSFTGUHMINJWZYX" ogni carattere ha la posizione uguale al peso che gli spetta (per i posti dispari) 😊😊*



# IMPLEMENTAZIONE

---

*Come potrei togliervi la soddisfazione di farla voi...? ☺*

**Però, la vostra soluzione dovrà  
superare il *mio* collaudo ☺**

Piano di collaudo:

- verificare tutte le funzioni
- progettando, per ciascuna di esse, uno o più casi significativi
- particolare attenzione ai *casi limite*:  
*non si collauda solo il caso ovvio e banale!*



# PIANO DI COLLAUDO

```
assert(CodiceFiscale.calcolaCodiceFiscale(
    "Mario", "Rossi", 12, 6, 1976, "M", "Bologna").equals("RSSMRA76H12A944I"));
assert(CodiceFiscale.calcolaCodiceFiscale(
    "Mario", "Rossi", 1, 1, 1990, "M", "Milano").equals("RSSMRA90A01F205Z"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 12, 6, 1976, "M", "Bologna", "RSSMRA76H12A94QF"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 12, 6, 1976, "M", "Bologna", "RSSMRA76H12A9Q4U"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 1, 1, 1990, "M", "Milano", "RSSMRA90A01F20RU"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 1, 1, 1990, "M", "Milano", "RSSMRA90A01F2L5K"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 1, 1, 1990, "M", "Milano", "RSSMRA90A01FN05O"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 1, 1, 1990, "M", "Milano", "RSSMRA90A0MF205R"));
assert(CodiceFiscale.verificaCodiceFiscale(
    "Mario", "Rossi", 1, 1, 1990, "M", "Milano", "RSSMRA90AL1F205K"));
...
```

calcola il codice "base"

omocodie



# QUALCHE «DRITTA» PER I VARI LINGUAGGI...

- Occhio alle parentesi quadre / tonde
  - Java, C# e Kotlin usano, come il C, le quadre per accedere agli array; Scala invece utilizza le parentesi tonde
  - Java non permette le quadre sulle stringhe (necessario usare `charAt`); gli altri linguaggi invece le consentono (C# e Kotlin le esigono)
- Conversioni carattere/numero e viceversa
  - Java e C# consentono, come il C, un'aritmetica mix fra caratteri e numeri (esempio: `char ch2 = ch1 + 'A' - '0'`)  
al contrario, Scala e Kotlin non la consentono → necessarie *conversioni esplicite* tramite i metodi `toInt`, `toChar`, etc.
- Operatore condizionale ternario
  - Java e C# prevedono, come il C, l'operatore condizionale ternario `? :` (esempio: `int absValue = x >= 0 ? x : -x`)  
Scala e Kotlin lo rinominano invece come *if/else expression*  
→ `var absValue : Int = if (x >= 0) x else -x`



# QUALCHE «DRITTA» PER I VARI LINGUAGGI...

- Cicli

- Java e C# consentono, come il C, cicli `for` «stile `while`», in cui è possibile specificare qualunque condizione e l'indice è controllato *esplicitamente*; al contrario, Scala e Kotlin hanno cicli `for` che iterano su un range di valori, con indice controllato *implicitamente*:

Scala: `for(i <- intervallo)`      Kotlin: `for(i in intervallo)`

→ in presenza di condizioni, spesso è necessario riformularli come *while*

- in Scala l'espressione del `for` può includere o escludere l'estremo superiore destro dell'intervallo (`to` vs. `until`; esistono anche altre keyword..):

`0 to N`    *da 0 a N incluso*

`0 until N`    *da 0 a N escluso*

- in Scala infine il ciclo `for` può anche prevedere *condizioni di filtro* con `if`, che permettono in molti casi di evitare di riscrivere il ciclo come `while`

→ ESEMPI:

`for(i <- 0 to 10)`

*da 0 a 10 incluso*

`for(i <- 0 to 10 if i%2==1)`

*solo 1,3,5,7,9*



# QUALCHE «DRITTA» PER I VARI LINGUAGGI...

- Main

- Java, C# e Scala prevedono che il main sia *all'interno* della classe (o, in Scala, dell'object); al contrario, Kotlin di default si aspetta che sia a top level, ossia all'esterno di tutti gli object: se così non è, occorre specificarlo con l'apposita annotazione `@JvmStatic`:

```
public object Code {
```

```
...
```

```
@JvmStatic
```

```
fun main(args: Array<String>)
```

```
...
```

```
}
```

Non necessaria se il main  
Kotlin è posto al top level

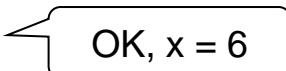
- se il main è in una classe o object *separato* da altre che usa, i compilatori e i linker Java, C# e Scala recuperano gli altri pezzi automaticamente, anche compilando una sola classe o object per volta;  
in Kotlin ciò avviene solo lanciando il compilatore *contemporaneamente* su *tutte* le classi o object che interessano




# QUALCHE «DRITTA» PER I VARI LINGUAGGI...

- Semicolon insertion

- In Java e C#, come in C, gli «a capo» nel testo non hanno rilevanza  
Scala e Kotlin invece li considerano *equivalenti a un punto e virgola*  
→ andare a capo «malamente» può cambiare il significato o causare errori

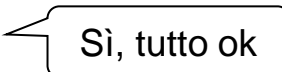
var x = 1 + 5;  OK, x = 6

var x = 1  
+5;  NO, x = 1 ed errore sulla frase seguente

var x = 1 +  
5;  OK, x = 6

- Operatori di incremento e decremento

- Java, C#, Kotlin mantengono gli operatori di incremento del C, ++ e --  
Scala invece li elimina → necessario sostituirli con assegnamenti ibridi:

var x = 3; x += 1;  Sì, tutto ok

var x = 3; x++;  NO: operatore inesistente