



Alma Mater Studiorum-Università di Bologna Scuola di Ingegneria

OOP: linguaggi e piattaforme

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ENRICO DENTI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)



IL PUNTO DI VISTA PROCEDURALE

I linguaggi "classici", come il C, adottano il **punto di vista procedurale**

- l'enfasi è sull'**operazione da svolgere** (primo argomento)
- «chi» la svolge è in secondo piano (se c'è...)

Infatti, per svolgere l'operazione `operation` sul componente **comp**, si scrive tipicamente:

`operation(comp, parametri)`

COSA fare

CHI la fa

informazioni accessorie

Esempio: `fprintf(fout, "Hello!");`



IL PUNTO DI VISTA PROCEDURALE

L'approccio *procedurale*

- è naturale in un mondo semplice, dove c'è un solo ("ovvio") destinatario delle operazioni
 - architettura monolitica: "chi" svolge le operazioni è scontato
 - focus sull'algoritmo, ossia la *sequenza di azioni* da svolgere
- **mostra tutti i suoi limiti in presenza di *sistemi software***
 - architettura *multi-componente*: molte entità interagiscono fra loro
 - interazione più che / accanto a algoritmica: **il focus non è più solo sulle operazioni da svolgere, ma su CHI faccia COSA**
 - focus su **come distribuire le responsabilità** fra i componenti
 - conseguente necessità di **evidenziare A CHI CI SI RIVOLGE** per richiedere una certa operazione / un certo SERVIZIO



L'INVERSIONE DEL PUNTO DI VISTA

Nell'approccio a oggetti, *l'enfasi è sull'oggetto*

- una entità dotata di una propria identità
- con le sue proprietà
- e in grado di svolgere certi servizi (operazioni)

Questo porta a *invertire il punto di vista*:

- enfasi *non più sull'operazione svolta* (da qualcuno)
- ma sull'entità che la svolge → *l'oggetto*

Vi stupisce? NON DOVREBBE, visto che *lo fate continuamente!*

- ogni volta che fate "doppio clic" o "tap" su un'icona *vi concentrate sul "chi" deve fare qualcosa, non sull'operazione: "mandate un messaggio" all'entità e le chiedete di fare qualcosa (APRIRSI / ESEGUIRSI)*
- non aprite il programma/app, per poi scegliere l'operazione da menù!

IL PUNTO DI VISTA «A OGGETTI»

Nell'approccio a oggetti:

- si mette *in primo piano* **chi** svolge l'operazione
- l'operazione da svolgere *passa in secondo piano*

Per esprimere questo cambiamento di prospettiva:

- si adotta una *notazione sintattica che enfatizzi il cambiamento*
- si riutilizza a tal fine la *notazione puntata* già in uso per le **struct**, attribuendole però *nuovo significato*

Per chiedere al componente **comp** di svolgere **operation**, si scrive:

comp.operation (*parametri*)

a CHI mi
rivolgo

COSA gli
chiedo di fare

dettagli e info
per farla

NOTAZIONE PUNTATA

- Nei linguaggi OO, la notazione puntata indica anche la **selezione di un'operazione** fra quelle offerte da un'entità
- Si usa parlare di **metodo** per richiedere un servizio

Ad esempio, in JAVA la frase:

```
System.out.println("Hello!");
```

richiede al componente **System.out** di svolgere il servizio `println`
(**out** è a sua volta un componente dell'entità **System**)

Analogamente, in C# la frase:

```
System.Console.WriteLine("Hello!");
```

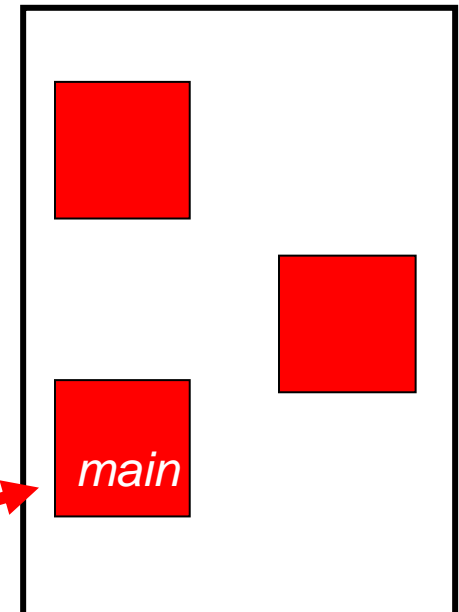
richiede al componente **System.Console** di svolgere il servizio `WriteLine`
(**Console** è a sua volta un componente nell'entità **System**)

APPLICAZIONI A OGGETTI: ARCHITETTURA

Una applicazione a oggetti è strutturata come *un insieme di entità*, di cui:

- alcune sono *statiche*, ossia esistono *prima* dell'inizio del programma e permangono *per tutta la sua durata*
 - librerie (prive di stato, es. libreria matematica)
 - moduli software statici (oggetti singoli)
 - definizioni di tipi

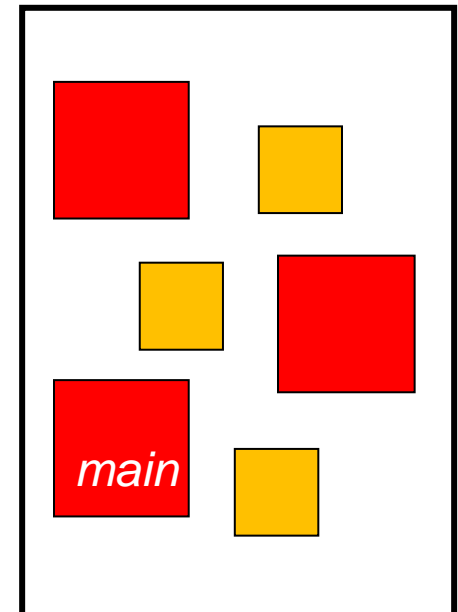
Poiché ogni applicazione deve avere un punto di partenza prestabilito, *una di tali entità statiche contiene il main*



APPLICAZIONI A OGGETTI: ARCHITETTURA

Una applicazione a oggetti è strutturata come *un insieme di entità*, di cui:

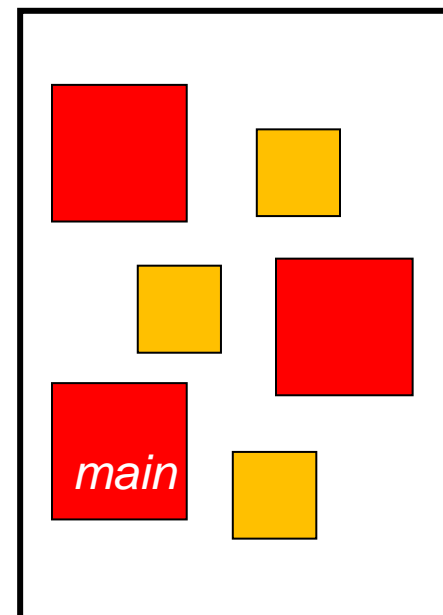
- alcune sono *statiche*, ossia esistono *prima* dell'inizio del programma e permangono *per tutta la sua durata*
- altre invece sono *dinamiche*, ossia vengono create durante l'esecuzione solo al momento del bisogno



APPLICAZIONI A OGGETTI: ARCHITETTURA

Una applicazione a oggetti è strutturata come *un insieme di entità*, di cui:

- alcune sono *statiche*, **CLASSI** o *oggetti singleton* prima dell'inizio del programma e restano in memoria per tutta la sua durata
- altre invece sono *dinamiche*, ossia **OGGETTI** dinamici, vengono create durante l'esecuzione del programma e vengono distrutte al termine dello stesso





JAVA & co.: IL LINGUAGGIO

- Java, C#, Scala, Kotlin sono linguaggi *progettati ex novo*, facendo tesoro delle esperienze (e degli errori) precedenti
- Di base simili a C e C++, ma *senza il requisito della piena compatibilità all'indietro*
- Obiettivo 1: sostituire meccanismi e costrutti linguistici poco chiari, sintatticamente contorti ed error-prone con *nuovi meccanismi e costrutti evoluti*
- Obiettivo 2: intercettare a compile-time quanti più errori possibile: *"se si compila, molto probabilmente è ok"*
 - sostituzione dei puntatori con *riferimenti*
 - dereferenzamento automatico
 - allocazione e deallocazione automatica della memoria (heap)
 - type system molto più stringente + *type inference* + controlli a run-time



JAVA & co.: IL LINGUAGGIO

- Java, C#, Scala, Kotlin sono linguaggi *progettati ex novo*, facendo tesoro delle esperienze (e degli errori) precedenti
- Di base simili a C e C++, ma *senza il requisito della piena compatibilità all'indietro*
- Obiettivo 1: sostituire meccanismi e costrutti linguistici poco chiari, sintatticamente contorti ed error-prone con *nuovi meccanismi e costrutti evoluti*
- Obiettivo 2: intercettare a compile-time quanti più errori possibile: "*se si compila*"
 - sostituzione dei puntatori
 - dereferenzamento automatico
 - allocazione e deallocazione automatica
 - type system molto più moderno

In Scala e Kotlin, ulteriori passi avanti:

- riduzione dell'uso di puntatori *null*
- *type inference* evoluta
- distinzione valori / variabili
- funzioni come *first-class entities*

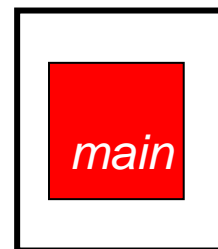


IL MAIN dal C a JAVA & co.

La più semplice applicazione a oggetti è costituita da **un singolo componente (*singleton*)**, che definisce **soltanto il `main`**

PRIMA DIFFERENZA RISPETTO AL C:

- in C, il main è *semplicemente scritto in un file, non è racchiuso in alcun costrutto linguistico*
- in Java e C# **il main dev'essere scritto dentro una classe pubblica** ed essere esso stesso ***pubblico*** (*protezione*)



```
public class MyProg {  
    ...  
    // il main (anch'esso pubblico)  
    ...  
}
```

IL MAIN dal C a JAVA & co.

Scala e Kotlin sono analoghi, con leggere differenze:

(segue)

- **in Scala** la qualifica *public* è predefinita → non va specificata inoltre, i componenti singleton si chiamano *object*
- idem **in Kotlin**, ma il componente *object* che contiene il main è definito in modo *implicito* → non va specificato



main

```
object MyProg {  
    ...  
    // il main Scala  
    ...  
}
```

```
// object implicito  
...  
// il main kotlin  
...
```



IL MAIN dal C a JAVA & co.

Proseguiamo con l'analisi comparativa:

SECONDA DIFFERENZA RISPETTO AL C:

- in C, il `main` può avere (argc/argv) o non avere argomenti
- in Java, il `main` ha sempre come unico argomento un *array di stringhe*
- in C#, il `Main` può avere come unico argomento un *array di stringhe* (segue)...

```
public class MyProg { // Java o C#  
    public static void main(String[] args) {  
        ...  
    }  
}
```

C#: `Main`

C#: `string[]` (opzionale)



IL MAIN dal C a JAVA & co.

Proseguiamo con l'analisi comparativa:

(segue)

- in Scala e Kotlin, il main è introdotto da una parola chiave (**def/fun**), e gli array si indicano con la parola chiave **Array** anziché **[]**

```
object MyProg { // Scala
    def main(args: Array[String])...
        ...
    }
}
```

Sintassi di tipo
postfissa

```
// Kotlin
fun main(args: Array<String>)...
    ...
}
```



IL MAIN dal C a JAVA & co.

Analisi comparativa, ultimo passo

TERZA DIFFERENZA RISPETTO AL C:

- in C, il `main` può avere tipo di ritorno `void` o `int`
 - in Java, il `main` ha sempre tipo di ritorno `void` (NON `int`)
 - in C#, il `Main` può avere tipo di ritorno `void` o `int`
- (segue)

```
public class MyProg {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

C#: `void` o `int`



IL MAIN dal C a JAVA & co.

Analisi comparativa, ultimo passo

(segue)

- in Scala e Kotlin, il main ha tipo di ritorno **Unit** (come dire **void**) ma la sintassi è meno familiare: il tipo di ritorno si scrive *in fondo*

```
object MyProg { // Scala
    def main(args: Array[String]) :Unit = {
        ...
    }
}
```

Obbligatorio

Importante!

```
// Kotlin
fun main(args: Array<String>) :Unit {
    ...
}
```

Facoltativo



IL CASO PIÙ SEMPLICE (C, Java, C#)

```
// file MyProg.c
int main(int argc, char* argv){
    int x=3, y=4; int z= x+y;
}
```



C

```
public class MyProg {
    public static void main(String[] args){
        int x=3, y=4; int z= x+y;
    }
}
```



Java

```
public class MyProg {
    public static void Main(string[] args){
        int x=3, y=4; int z= x+y;
    }
}
```



C#

IL CASO PIÙ SEMPLICE (Java, Scala, Kotlin)

```
public class MyProg {  
    public static void main(String[] args) {  
        int x=3, y=4; int z= x+y;  
    }  
}
```

NOVITÀ

- in Scala e Kotlin, le variabili sono introdotte dalla parola chiave **var** (se sono valori imm modificabili, **val**)

Java

```
object MyProg {  
    def main(args: Array[String]):Unit = {  
        var x:Int = 3, y:Int = 4, z:Int = x+y;  
    }  
}
```

val, se non più modificati

Scala

```
fun main(args: Array<String>):Unit {  
    var x:Int = 3, y:Int = 4, z:Int = x+y;  
}
```

val, se non più modificati

Kotlin



CONVENZIONI SUI NOMI DEI FILE

Si applicano le seguenti convenzioni di naming:

- in Java, il file deve chiamarsi *esattamente come la classe* e avere estensione **.java**
- in C#, il file dovrebbe chiamarsi *come la classe* e avere estensione **.cs**
- in Scala, il file deve chiamarsi *esattamente come la classe o object* e avere estensione **.scala**
- in Kotlin, il file deve chiamarsi *esattamente come la classe o object* e avere estensione **.kt**

NB: «esattamente come la classe» significa *maiuscole/minuscole comprese*, senza eccezioni.



COMPILAZIONE (C, Java, C#)

COMPILAZIONE C

```
C:> cc MyProg.c
```

produce MyProg.exe

L'EXE ottenuto è eseguibile *sul sistema operativo*

COMPILAZIONE Java

```
C:> javac MyProg.java
```

produce MyProg.class

Il file ottenuto è eseguibile *sull'infrastruttura Java*

NB: dev'essere installato il JDK e dev'essere nel PATH

COMPILAZIONE C#

```
C:> csc MyProg.cs
```

produce MyProg.exe

Il file ottenuto è un EXE eseguibile *sull'infrastruttura .NET*

NB: dev'essere installato il .NET Framework e dev'essere nel PATH

Non sono
la stessa
cosa!



COMPILAZIONE (Scala, Kotlin)

Scala e Kotlin sono costruiti per funzionare *sulla stessa infrastruttura base di Java*: la Java Virtual Machine (JVM)

Pertanto:

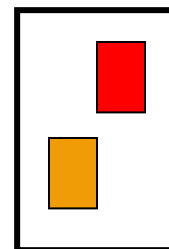
- la compilazione si lancia coi rispettivi compilatori
 - `javac` per Java
 - `scalac` e `kotlinc` rispettivamente per Scala e Kotlin
- ma *il risultato è comunque costituito da file `.class`*
 - perché Kotlin e Scala sono basati sulla piattaforma Java (con cui tra l'altro sono *interoperabili*)

NB: entrambi questi linguaggi possono compilare anche per altre piattaforme, la JVM non è l'unica opzione

UN ESEMPIO CON DUE ENTITÀ

Un programma costituito da *due entità*:

- la nostra **Esempio1**, che definisce il `main`
- una **classe fornita dall'infrastruttura**



Obiettivo:

- stampare a video la classica frase di benvenuto sfruttando il servizio di stampa fornito dal **"sistema"**

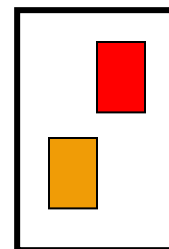
In Java e C#, classe **System** rappresenta *il sistema sottostante*, con tutti i suoi sotto-componenti e i relativi servizi

- uno di tali componenti è il *dispositivo standard di uscita* (chiamato *out* in Java e *Console* in C#)
- tale componente offre, fra gli altri, il *servizio di stampa* (chiamato *println* in Java e *WriteLine* in C#)

UN ESEMPIO CON DUE ENTITÀ

Un programma costituito da *due entità*:

- la nostra **Esempio1**, che definisce il `main`
- una **classe fornita dall'infrastruttura**



Obiettivo:

- stampare a video la classica frase di benvenuto sfruttando il servizio di stampa fornito dal **"sistema"**

In Scala e Kotlin, la suddivisione interna dei servizi è diversa, ma il risultato è analogo

- in Scala, il *dispositivo di uscita* si chiama **Console** (come in C#) e offre il *servizio di stampa* (chiamato però `println` come in Java)
- in Kotlin, il *dispositivo di uscita* è nel componente **kotlin.io** e il *servizio di stampa* si chiama anche lì `println` come in Java



UN ESEMPIO CON DUE ENTITÀ: C, Java, C#

```
<include stdio.h>
```

```
int main(int argc, char* argv[]) {  
    printf("%s", argv[1]);  
}
```

C

In Java e C#, `args[0]` non è il nome del programma: è già il primo argomento

```
public class Esempio1 {  
    public static void main(String[] args) {  
        System.out.println(args[0]);  
    }  
}
```

Java

```
public class Esempio1 {  
    public static void Main(string[] args) {  
        System.Console.WriteLine(args[0]);  
    }  
}
```

C#



UN ESEMPIO CON DUE ENTITÀ: Java, Scala, Kotlin

```
public class Esempio1 {  
    public static void main(String[] args) {  
        System.out.println(args[0]);  
    }  
}
```

Anche in Scala e Kotlin, `args[0]` è già il primo argomento

Java

```
object Esempio1 {  
    def main(args: Array[String]): Unit = {  
        Console.println(args(0));  
    }  
}
```

Scala: parentesi tonde per accedere agli array

In Scala, `println` è un metodo dell'oggetto `Console`.
Si può scrivere più brevemente anche solo `println`.

Scala

```
public fun main(args: Array<String>): Unit {  
    kotlin.io.println(args[0]);  
}
```

In Kotlin, `println` è un metodo dell'oggetto `kotlin.io`.
Si può scrivere più brevemente anche solo `println`.

Kotlin



UN ESEMPIO CON DUE ENTITÀ: Java, Scala, Kotlin

```
public class Esempio1 {  
    public static void main(String[] args) {  
        System.out.println(args[0]);  
    }  
}
```

Anche in Scala e Kotlin, `args[0]`
è già il primo argomento

Java

```
object Esempio1 {  
    def main(args: Array[String]):Unit = {  
        println(args(0)); // Console non si scrive  
    }  
}
```

In pratica, si scrive sempre solo `println`

Scala

```
public fun main(args: Array<String>):Unit {  
    println(args[0]); // kotlin.io non si scrive  
}
```

In pratica, si scrive sempre solo `println`

Kotlin



COMPILAZIONE: C, Java, C#

COMPILAZIONE C

C:> cc Esempio1.c *produce Esempio1.exe*

L'EXE ottenuto è eseguibile sul sistema operativo

COMPILAZIONE Java

C:> javac Esempio1.java *produce Esempio1.class*

Il file ottenuto è eseguibile sull'infrastruttura Java

*La compilazione di programmi Scala o Kotlin è analoga, usando i rispettivi compilatori **scalac** e **kotlinc***

*Il risultato sono comunque dei file di tipo **.class***

COMPILAZIONE C#

C:> csc Esempio1.cs *produce Esempio1.exe*

Il file ottenuto è un EXE eseguibile sull'infrastruttura .NET

*Non sono
la stessa
cosa!*



COMPILAZIONE: Java, Scala, Kotlin

COMPILAZIONE Java

```
C:> javac Esempio1.java
```

*produce **Esempio1.class***

Il file ottenuto è eseguibile sull'infrastruttura Java

COMPILAZIONE Scala

```
C:> scalac Esempio1.scala
```

*produce **Esempio1.class**
ed **Esempio1\$.class***

Il file ottenuto è eseguibile sull'infrastruttura Java (+ librerie Scala)

COMPILAZIONE Kotlin

```
C:> kotlinc Esempio1.kt
```

*produce **Esempio1Kt.class***

Il file ottenuto è eseguibile sull'infrastruttura Java (+ librerie Kotlin)

ESECUZIONE: C, Java, C#

```
C:> Esemplio1 alfa beta gamma  
alfa
```

C

L'EXE ottenuto è eseguito direttamente sul sistema operativo

```
C:> java Esemplio1 alfa beta gamma  
alfa
```

Java

Il file ottenuto è eseguito sull'infrastruttura Java

OSSERVA: occorre invocare esplicitamente l'interprete Java (strato-ponte)

```
C:> Esemplio1 alfa beta gamma  
alfa
```

C#

Il file ottenuto è eseguito sull'infrastruttura .NET

NOTA: sembra uguale al primo.. ma non funziona se sulla macchina non è installato il Microsoft .NET Framework



ESECUZIONE: Java, Scala, Kotlin

```
C:> java Esempio1 alfa beta gamma  
alfa
```

Java

Il file ottenuto è eseguito sull'infrastruttura Java

OSSERVA: occorre invocare esplicitamente l'interprete Java (strato-ponte)

```
C:> scala Esempio1 alfa beta gamma  
alfa
```

Scala

Il file ottenuto è eseguito sull'infrastruttura Java (+ librerie Scala)

OSSERVA: occorre invocare esplicitamente l'interprete Scala (Java..)

```
C:> kotlin Esempio1Kt alfa beta gamma  
alfa
```

Kotlin

Il file ottenuto è eseguito sull'infrastruttura Java (+ librerie Kotlin)

OSSERVA: occorre invocare esplicitamente l'interprete Kotlin (Java..)

ESECUZIONE: Java, Scala, Kotlin

```
C:> java Esempio1 alfa beta gamma  
alfa
```

Java

Il file ottenuto è eseguito sull'infrastruttura Java

OSSERVA: occorre invocare esplicitamente l'interprete Java (strato-ponte)

```
C:> scala Esempio1 alfa beta gamma  
alfa
```

Scala

Il file ottenuto è ese

OSSERVA: occorre

Sono fondamentalmente degli alias: sotto,
richiamano lo stesso strato-ponte **java**,
con opportuni parametri e librerie

ala)
(Java..)

```
C:> kotlin Esempio1kt alfa beta gamma  
alfa
```

Kotlin

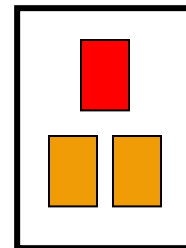
Il file ottenuto è eseguito sull'infrastruttura Java (+ librerie Kotlin)

OSSERVA: occorre invocare esplicitamente l'interprete Kotlin (Java..)

UN TERZO ESEMPIO (1/4)

Un programma costituito da *tre classi*:

- la nostra **Esempio2**, che definisce il **main**
- **due entità di infrastruttura**
 - una per stampare, come prima
 - una per fare calcoli: il componente «libreria matematica»
- Chi è e come si chiama la libreria matematica?
 - in Java è l'entità (classe): **Math**
 - in C# è l'entità (classe): **System.Math**
 - in Scala è l'entità: **scala.math**
 - in Kotlin è l'entità: **kotlin.math**
- Cosa offre?
 - Costanti (e , π)
 - Decine di funzioni utili



NB: in C#, Scala e Kotlin, Math (o math) è un sotto-componente di qualcos'altro (System o altra entità «top level»); in Java no.



UN TERZO ESEMPIO (2/4)

```
<include stdio.h>
<include math.h>
int main() {
    printf("%f", sin(M_PI/3) );
}
```

C

```
public class Esempio2 {
    public static void main(String[] args) {
        System.out.println( Math.sin(Math.PI/3) );
    }
}
```

Java

```
public class Esempio2 {
    public static void Main(string[] args) {
        System.Console.WriteLine(
            System.Math.Sin(System.Math.PI/3) );
    }
}
```

C#

UN TERZO ESEMPIO (3/4)

```
public class Esempio2 {  
    public static void main(String[] args) {  
        System.out.println( Math.sin(Math.PI/3) );  
    }  
}
```

Java

```
object Esempio2 {  
    def main(args: Array[String]):Unit = {  
        println(scala.math.sin(scala.math.Pi/3))  
    }  
}
```

Scala

Poiché Scala e Kotlin sono interoperabili con Java, è anche possibile scrivere **Math.sin(Math.PI/3)**, usando il componente Java dell'infrastruttura sottostante.

```
// file Esempio2.kt  
fun main(args: Array<String>):Unit {  
    println(kotlin.math.sin(kotlin.math.PI/3))  
}
```

Kotlin

UN TERZO ESEMPIO (4/4)

```
C:> Esempio2  
0,866025403784439
```

C

```
C:> java Esempio2  
0,866025403784439
```

Java

Il file ottenuto è eseguito sull'infrastruttura Java

OSSERVA: occorre invocare esplicitamente l'interprete Java

```
C:> Esempio2  
0,866025403784439
```

C#

Il file ottenuto è eseguito sull'infrastruttura .NET

```
C:> scala Esempio2  
0,866025403784439
```

Scala

```
C:> kotlin Esempio2Kt  
0,866025403784439
```

Kotlin

Il file ottenuto è eseguito sull'infrastruttura Java (+ librerie Scala o Kotlin)



AMBIENTI ONLINE

Per sperimentare senza dover installare niente sul proprio computer (o usando un tablet o smartphone), si possono usare gli *ambienti online*

- disponibili per praticamente ogni linguaggio
- accessibili con un comune browser
- spesso chiamati «**playground**»
 - Java (Kotlin, Scala, Swift..): <https://code.labstack.com/java>
 - Java: <https://www.studytonight.com/code/playground/java/>
 - C# playground: <https://dotnetfiddle.net/srx9kM>
 - Kotlin playground: <https://play.kotlinlang.org/>
 - Scala: <https://scastie.scala-lang.org/>
<https://scalafiddle.io/>
<https://www.katacoda.com/courses/scala/playground>
 - ...

AMBIENTI ONLINE

.NET Fiddle New Fork Run Share

Options Options

Language: C#

Project Type: Console

Compiler: .NET 4.7.2

C# PlayGround by Alen Vadassery

```
1 using System;
2
3 public class Esem
4 public static vo
5     float f = (fl
6     System.Consol
7 }
8 }
```

0.8660254



Kotlin Playground is an online sandbox to explore Kotlin programming language. Browse code samples directly in the browser

```
public object Contatore {
    private var stato: Int = 0;
    public fun reset(): Unit {stato=0;}
    public fun inc():Unit {stato++;}
    public fun getValue(): Int {return stato;}
}

fun main(args: Array<String>) {
    Contatore.reset(); Contatore.inc();
    System.out.println(Contatore.getValue());
}
```

Scastie Editor Run New Format Clear Annotations Worksheet Save

Build Settings

object Contatore {

You don't need a main method (or extends Scastie) in Worksheet Mode

```
private var stato: Int = 0;
def reset(): Unit = {stato=0;}
def inc():Unit = {stato+=1;}
def getValue(): Int = {return stato;}
}

object MyMain {
    def main(args: Array[String]):Unit = {
        Contatore.reset(); Contatore.inc();
        println(Contatore.getValue());
    }
}
```

codingground SIMPLY EASY CODING Compile and Execute Scala Online (Scala v2.10.6)

Execute Share HelloWorld.scala STDIN Result

```
1 object Contatore {
2     private var stato: Int = 0;
3     def reset(): Unit = {stato=0;}
4     def inc(): Unit = {stato+=1;}
5     def Value(): Int = {return stato;}
6 }

7 def main(args: Array[String]) : Unit = {
8     Contatore.reset(); Contatore.inc();
9     println(Contatore.getValue());
10 }
```

\$scalac *.scala

trinket Java beta Run Share

New Tutorials: NUMPY TKINTER KOTLIN JAVASCRIPT

Java Compiler:

Main.java

```
1 /*
2  * We have created a Main class for you
3  */
4 public class Main {
5     public static void main(String args[]) {
6         /*
7          * Main method...
8          */
9     }
10 }
```

labstack Untitled Notes

Java (11.0)

```
public class Code {
    public static void main(String[] args) {
        System.out.println("Hello from Java!");
    }
}
```

Input

Output

Il problema della documentazione



UN ALTRO ASPETTO: LA DOCUMENTAZIONE

- È noto che un buon programma dovrebbe essere "ben documentato"...
- *...ma l'esperienza insegna che quasi mai ciò viene fatto!*
 - “non c'è tempo”, “ci si penserà poi”...
 - ..e alla fine la documentazione non c'è! ☹
- Java prende atto che *la gente non scrive documentazione* e quindi fornisce uno *strumento per produrla automaticamente a partire da particolari commenti* nel programma: **javadoc**
- Un commento Javadoc inizia con **/**** (anziché **/***)
 - poi, termina normalmente con ***/**
 - può essere *in testa a una classe* o a *singole funzioni*



UN ALTRO ASPETTO: LA DOCUMENTAZIONE

- L'analogo strumento per Scala si chiama **Scaladoc**
 - segue al 99% la stessa sintassi di Javadoc
- L'analogo strumento per Kotlin si chiama **Kdoc**
 - la sintassi è un mix fra quella di Javadoc e il *markdown*
- In C#, il compilatore può estrarre da commenti *'// //'* *un file XML, la cui elaborazione è lasciata però ad altri strumenti*
 - strumenti come SandCastle, GhostDoc, NDoc3 generano vari formati (tipicamente, manuali in stile MSDN) a partire dall'XML



UN ESEMPIO "COMMENTATO"

```
/** File Eempio.java
 * Applicazione Java da linea di comando
 * Stampa la classica frase di benvenuto
@author Enrico Denti
@version 1.0, 5/4/2018
*/
```

Informazioni di documentazione
che verranno estratte

```
public class Eempio0 {
    public static void main(String args[]){
        System.out.println("Hello!");
    }
}
```

ESEMPIO IN C#

```
/// <summary>
/// testo di commento
/// </summary>
public class MyClass{}
...
```

GENERAZIONE AUTOMATICA DI DOCUMENTAZIONE

In Java, per produrre la relativa documentazione:

javadoc -d docs Esempio0.java

Produce nella cartella **docs**
un manuale HTML

Si consulti la
documentazione
di javadoc per i dettagli.

In C#, analoga funzione
è svolta dal compilatore
`csc` con opzione `/doc`

Class Esempio0

java.lang.Object
Esempio0

```
public class Esempio0
extends java.lang.Object
```

Applicazione Java da linea di comando. Stampa la classica frase di benvenuto

La frase riportata è quella estratta dai commenti

I tag come @author, @version vengono inseriti *solo a richiesta*

Modifier and Type	Method and Description
static void	main(java.lang.String[] args) Il main.

Le basi del linguaggio: tipi di dato di base



TIPI PRIMITIVI: sì o no?

- Java e C# mantengono la nozione di *tipo primitivo* del C
 - pur estendendoli e ridefinendoli
 - approccio «conservativo» dovuto a ragioni storiche (chi proveniva dal C era abituato ad averli) e di prestazioni (all'epoca)
 - MA l'esperienza ha dimostrato che non è stata una grande idea!
- I linguaggi definiti successivamente (Scala, Kotlin) vanno nella direzione di *sostituirli con opportuni tipi di oggetti*
 - approccio «evolutivo» mirante a garantire *uniformità*
 - non vi sono più le ragioni storiche di vent'anni fa, né problemi di prestazioni (anche perché «sotto banco» il tipo primitivo potrebbe continuare a esistere..)
 - visivamente, si nota un leggero cambio di nome (`int` → `Int`) ma concettualmente e nell'uso quotidiano si semplificano molte cose



BOOLEAN

- Un boolean non è più sinonimo di «intero 0/1»
- È un **tipo autonomo, totalmente disaccoppiato dagli interi**
 - le espressioni relazionali e logiche danno come risultato un `boolean`, non un `int` come in C
 - **intenzionalmente non si convertono boolean in interi e viceversa, neanche con cast** (bisogna scriversi funzioni apposite)
- SINTASSI
 - Java: **`boolean`** (tipo primitivo)
 - C#: **`bool`** (tipo primitivo)
 - Scala / Kotlin: **`Boolean`** (tipo di oggetto)

In tutti questi linguaggi, gli unici due valori ammessi per questo tipo sono **`false`** e **`true`**

BOOLEAN

- Un boolean non è più sinonimo di «intero 0/1»
- È un tipo autonomo, *totalmente disaccoppiato* dagli interi
 - le espressioni relazionali e logiche danno come risultato un `boolean`, non un `int` come in C
 - intenzionalmente non si convertono boolean in interi e viceversa, *neanche con cast* (bisogna scriversi funzioni apposite)
 - **solo C#** fornisce, nel componente **Convert**, una serie di funzioni di utilità che coprono anche questa casistica

```
public static void Main()
{
    bool a = true, b = false;
    int x = Convert.ToInt16(a); Console.WriteLine(x);
    int y = Convert.ToInt16(b); Console.WriteLine(y);
    int n = 51, z = 0;
    bool t = Convert.ToBoolean(n); Console.WriteLine(t);
    bool f = Convert.ToBoolean(z); Console.WriteLine(f);
}
```

```
1
0
True
False
```



NUMERI INTERI

- Interi con segno

- Java: **byte** (1 byte) -128 ... +127 Scala/Kotlin: **Byte**
C#: **sbyte**
- Java/C#: **short** (2 byte) -32768 ... +32767 Scala/Kotlin: **Short**
- Java/C#: **int** (4 byte) $-2 \cdot 10^9 \dots +2 \cdot 10^9$ Scala/Kotlin: **Int**
- Java/C#: **long** (8 byte) $-9 \cdot 10^{18} \dots +9 \cdot 10^{18}$ Scala/Kotlin: **Long**

NB: le costanti `long` terminano con la lettera `L`

- Interi senza segno (solo C#)

- **byte** (1 byte) 0... 255
- **ushort** (2 byte) 0... 65535
- **uint** (4 byte) 0... $4 \cdot 10^9$
- **ulong** (8 byte) 0... $1.8 \cdot 10^{19}$

NB: le costanti `ulong` terminano con la lettera `L`

NUMERI REALI

- Standard IEEE-754

- Java/C#: **float** (4 byte) $-10^{45} \dots +10^{38}$ Scala/Kotlin: **Float**
- Java/C#: **double** (8 byte) $-10^{324} \dots +10^{308}$ Scala/Kotlin: **Double**
- **float**: circa 6-7 cifre decimali significative (precisione: $6 \cdot 10^{-8}$)
NB: le costanti **float** terminano con la lettera F
- **double**: circa 14-15 cifre decimali significative (precisione: $1 \cdot 10^{-16}$)

- Fuori standard

- solo C#: **decimal** (16 byte) $-10^{28} \dots +10^{28}$
- **decimal**: circa 28-29 cifre decimali significative (precisione: $1 \cdot 10^{-28}$)
NB: le costanti **decimal** terminano con la lettera M
PRO: molto preciso, perché internamente usa la base 10;
ciò è utile nei calcoli finanziari
CONTRO: molto più lento (20 volte) e range più ridotto



NUMERI REALI: COMPATIBILITÀ

- In Java, C#, Scala sono ammessi solo *gli assegnamenti che non causano perdita di informazione*.

Quindi, ad esempio:

La frase `double x = 3.54F;` è *lecita* (da `float` a `double` non si perde precisione)

La frase `float f = 3.54;` è *illecita* (da `double` a `float` si perderebbe precisione)

In C#, anche un valore `decimal` non può essere assegnato a una variabile `float` o `double`, poiché si perderebbe in precisione.

La frase `double d = 3.54M;` è quindi *illecita* (come sopra).



ESEMPIO IN JAVA

Esempio precedente (ok):

```
public class Esempio2 {  
    public static void main(String[] args) {  
        double x = Math.sin(Math.PI/3);  
        System.out.println(x);  
    }  
}
```

Ora, invece di un `double`, usiamo un `float`

```
public class Esempio2 {  
    public static void main(String[] args) {  
        float f = Math.sin(Math.PI/3);  
        System.out.println(f);  
    }  
}
```

ERRORE DI COMPILAZIONE

*Possible loss of precision
Found double, required float*

Java e C# non accettano che un valore `double` sia "impunemente" assegnato a una variabile `float`, perché causerebbe una perdita d'informazione

ESEMPIO IN C#

Esempio precedente (ok):

```
public class Esempio2 {  
    public static void Main(string[] args) {  
        double x = System.Math.Sin(System.Math.PI/3);  
        System.Console.WriteLine(x);  
    }  
}
```

..ma invece di un

```
public class Esempio3 {  
    public static void Main(string[] args) {  
        float f = System.Math.Sin(System.Math.PI/3);  
        System.Console.WriteLine(f);  
    }  
}
```

IL COMPILATORE C# è anche più pignolo...
per non dire "ironico" 😊

Cannot implicitly convert type 'double' to 'float'.

An explicit conversion exists (are you missing a cast?)

Purtroppo, in italiano quel sense of humour va perso.. 😞

Impossibile convertire in modo implicito il tipo 'double' in 'float'. È presente una conversione esplicita. Probabile cast mancante.



CONVERSIONI ESPLICITE

- Se si vuole *consapevolmente* usare un `float` per memorizzare un valore `double`, *accettando la perdita di precisione che ne deriverà*, occorre *asserirlo esplicitamente*
 - in **Java e C#**, con un **CAST**
 - in **Scala e Kotlin**, con opportune *funzioni di conversione*
- **PRINCIPIO-CHIAVE**: per convertire da un tipo a un altro, *soprattutto in caso si rischi una perdita di informazione*, occorre che **il progettista renda esplicito il suo *Design Intent***
 - ossia, dica chiaramente, *scrivendo qualcosa*, che ciò non è il frutto di una svista



CONVERSIONI ESPLICITE

- Esempio in Java

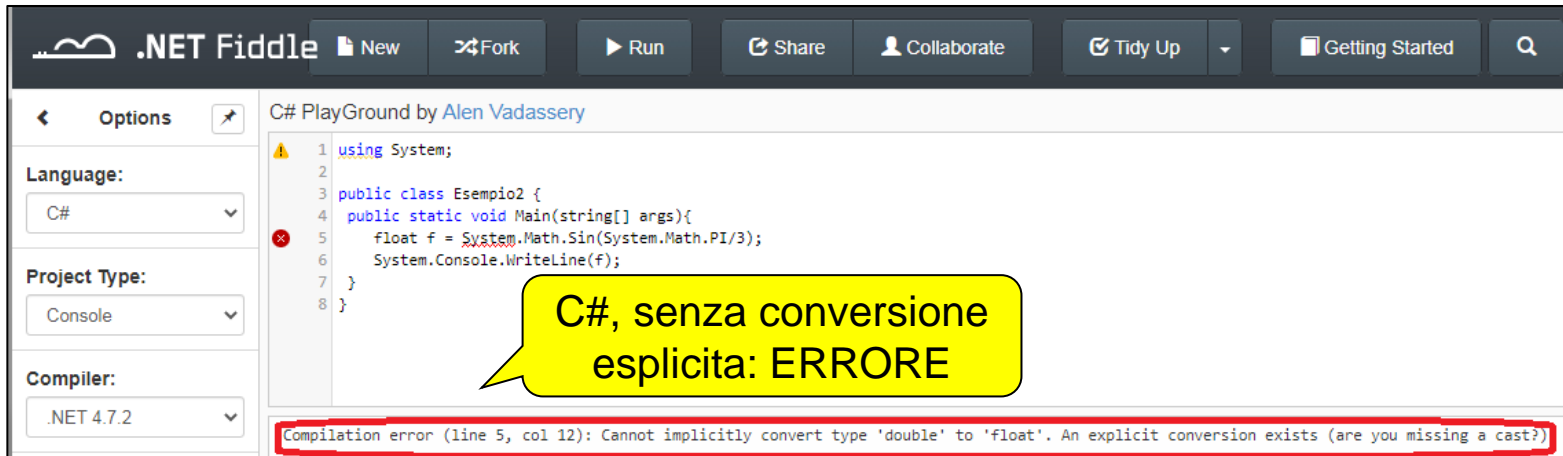
```
public class Esempio2 {  
    public static void main(String[] args) {  
        float f = (float) Math.sin(Math.PI/3);  
        System.out.println(f);  
    }  
}
```

Ora, compilazione ok

- Esempi in C#, Scala, Kotlin

- vedere screenshot dagli ambienti online nelle slide seguenti
- NB: in Scala, alcune funzioni vanno invocate *senza le parentesi ()*, in ossequio al cosiddetto «principio di accesso uniforme» [ne parleremo a suo tempo..]

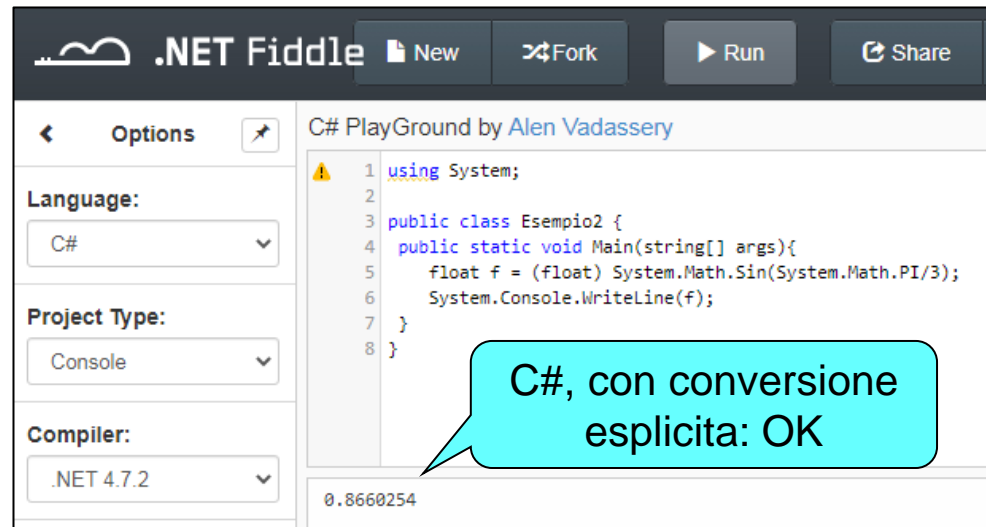
ESPERIMENTI ONLINE IN C#



The screenshot shows the .NET Fiddle interface. On the left, the 'Options' panel is set to Language: C#, Project Type: Console, and Compiler: .NET 4.7.2. The main editor displays the following C# code:

```
1 using System;
2
3 public class Esempio2 {
4     public static void Main(string[] args){
5         float f = System.Math.Sin(System.Math.PI/3);
6         System.Console.WriteLine(f);
7     }
8 }
```

A yellow speech bubble points to line 5, stating: "C#, senza conversione esplicita: ERRORE". At the bottom, a red box highlights the compilation error message: "Compilation error (line 5, col 12): Cannot implicitly convert type 'double' to 'float'. An explicit conversion exists (are you missing a cast?)"

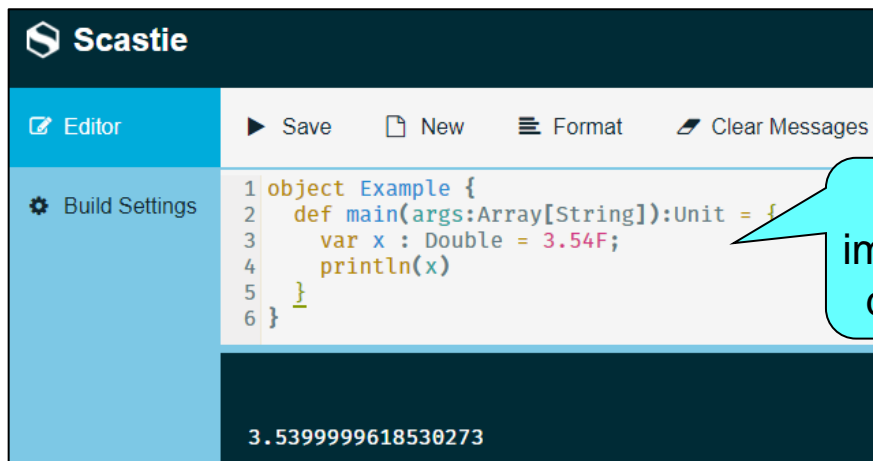


The screenshot shows the .NET Fiddle interface with the same settings as the first image. The C# code in the editor is modified to include an explicit cast:

```
1 using System;
2
3 public class Esempio2 {
4     public static void Main(string[] args){
5         float f = (float) System.Math.Sin(System.Math.PI/3);
6         System.Console.WriteLine(f);
7     }
8 }
```

A cyan speech bubble points to line 5, stating: "C#, con conversione esplicita: OK". The output at the bottom of the editor shows the value: 0.8660254.

ESPERIMENTI ONLINE IN Scala



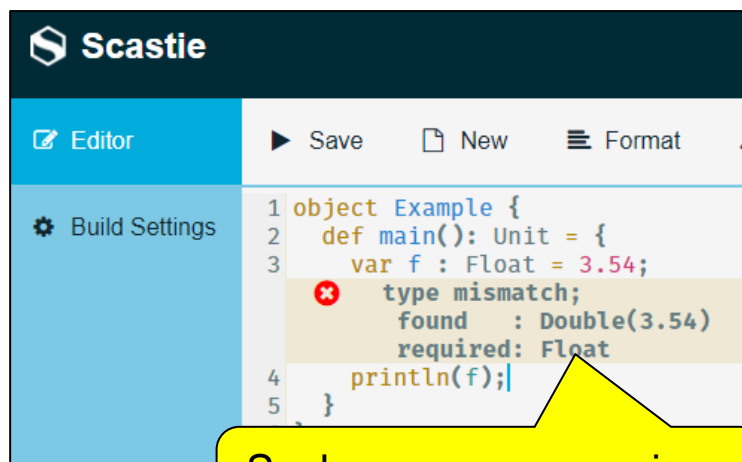
Scastie

Editor Save New Format Clear Messages

```
1 object Example {
2   def main(args:Array[String]):Unit = {
3     var x : Double = 3.54F;
4     println(x)
5   }
6 }
```

3.5399999618530273

Scala, conversione implicita senza perdita di informazione: OK



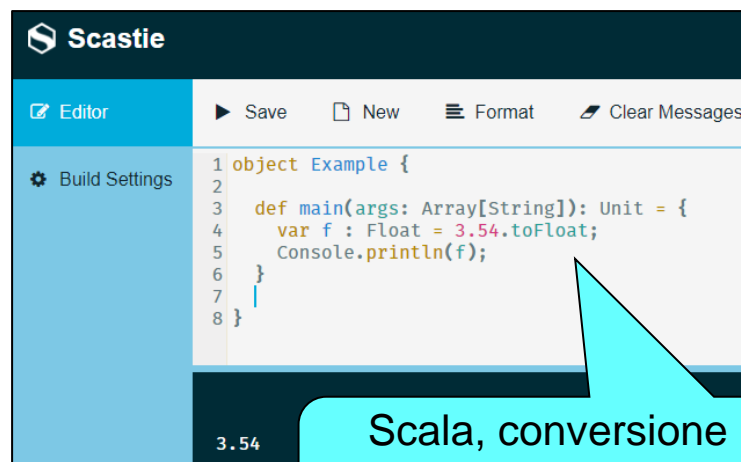
Scastie

Editor Save New Format

```
1 object Example {
2   def main(): Unit = {
3     var f : Float = 3.54;
4     println(f);
5   }
6 }
```

type mismatch;
found : Double(3.54)
required: Float

Scala, senza conversione esplicita: ERRORE



Scastie

Editor Save New Format Clear Messages

```
1 object Example {
2   def main(args: Array[String]): Unit = {
3     var f : Float = 3.54.toFloat;
4     Console.println(f);
5   }
6 }
7
8 }
```

3.54

Scala, conversione esplicita col metodo **toFloat**: OK



IN KOTLIN, INVECE...

- In Kotlin, prevale l'idea che le conversioni debbano **essere esplicite anche quando non c'è perdita di informazione**, per far emergere *sempre e comunque* il Design Intent
- Quindi, anche gli assegnamenti che in Java, C# e Scala sono leciti, qui diventano illeciti:

ESEMPIO KOTLIN

var x : Double = 3.54F; *diventa anch'essa una frase illecita*

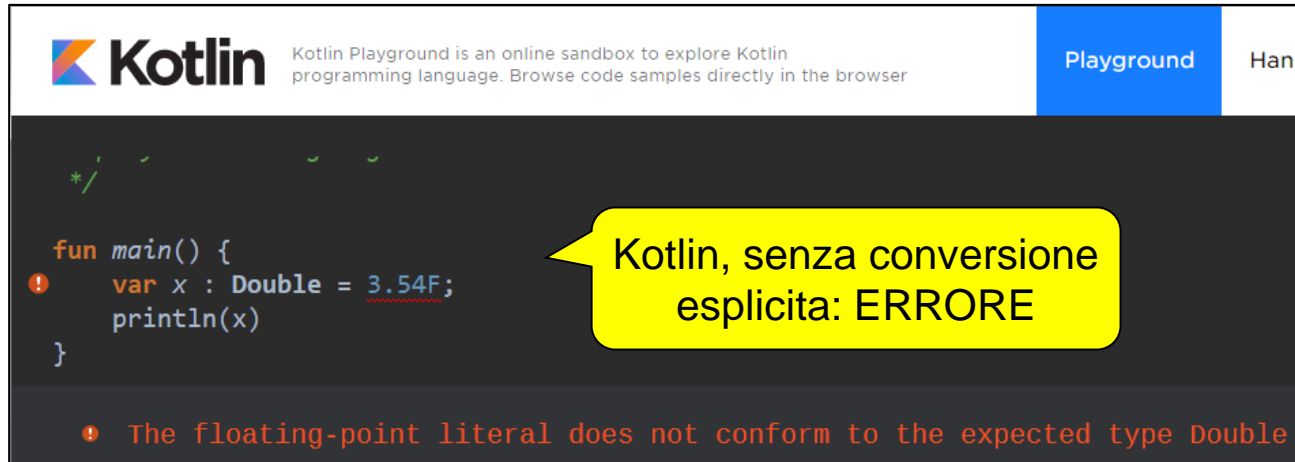
Niente conversioni implicite: deve emergere il DESIGN INTENT!

Si dovrà scrivere quindi una conversione *esplicita* (NO CAST), tramite le apposite funzioni della serie toXXX (qui, toDouble):

var x : Double = 3.54F.toDouble();



ESPERIMENTI ONLINE in Kotlin



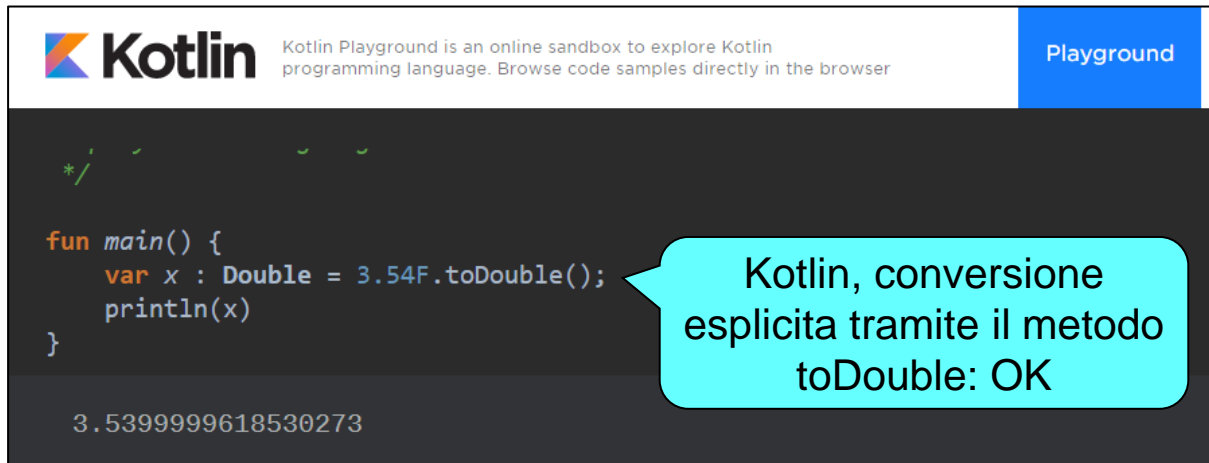
Kotlin Playground is an online sandbox to explore Kotlin programming language. Browse code samples directly in the browser

Playground Hand

```
fun main() {  
    var x : Double = 3.54F;  
    println(x)  
}
```

Kotlin, senza conversione esplicita: ERRORE

The floating-point literal does not conform to the expected type Double



Kotlin Playground is an online sandbox to explore Kotlin programming language. Browse code samples directly in the browser

Playground

```
fun main() {  
    var x : Double = 3.54F.toDouble();  
    println(x)  
}
```

Kotlin, conversione esplicita tramite il metodo toDouble: OK

3.5399999618530273



CARATTERI (1/2)

- A differenza del C, «carattere» non è più sinonimo di «byte»
 - 127 caratteri non bastano più da un pezzo!
 - il mondo non ospita solo le culture occidentali...
- Nuovo approccio: un «carattere» di 2 byte (UTF-16)
 - primi 127 caratteri uguali ad ASCII, primi 255 ad ANSI / ASCII
- Standard UNICODE
 - *Basic Multilingual:* 2 byte = 65536 «code point»
 - *Supplementary characters:* altri 16*65536 «code point»
 - *"Support for supplementary characters is a common business requirement in East Asian markets. Government applications require them to correctly represent rare Chinese characters. Publishing applications need them to represent the full set of historical and variant characters. The Hong Kong government defined characters needed for Cantonese"*

CARATTERI (2/2)

- Standard UNICODE

- full range da 000000_H a 10FFFF_H (1.114.112 caratteri)
- suddivisi in 17 «piani» da 65.536 caratteri ciascuno (5+16=21 bit)
- *Basic Multilingual* piano 0: range da 00000_H a 0FFFF_H
- *Supplementary Multilingual*: piano 1: range da 10000_H a 1FFFF_H
- Ulteriori caratteri: piani 2+: range da 20000_H a 10FFFF_H

Un carattere («code point») si indica con la notazione **U+nnnn**

NB: l'intervallo da D800 a DFFF del piano 0 non è assegnato (serve per UTF-16)

Piano	Intervallo	Descrizione
0	000000-00FFFF	Basic Multilingual
1	010000-01FFFF	Supplementary Multilingual
2	020000-02FFFF	Supplementary Ideographic
3	030000-03FFFF	Tertiary Ideographic
...	040000-0DFFFF	non usati
14	0E0000-0EFFFF	Supplementary Special-purpose
15	0F0000-0FFFFF	Supplementary Private Use Area A
16	100000-10FFFF	Supplementary Private Use Area B



- [illegible]



European Scripts	African Scripts	South Asian Scripts	East Asian Scripts
Armenian	Bamum	Bengali	Bopomofo
Armenian Ligatures	Bamum Supplement	Brahmi	Bopomofo Extended
Coptic	Egyptian Hieroglyphs (1MB)	Devanagari	CJK Unified Ideographs (Han) (28MB)
Coptic in Greek block	Ethiopic	Devanagari Extended	CJK Extension-A (6.3MB)
Cypriot Syllabary	Ethiopic Supplement	Gujarati	CJK Extension B (30MB)
Cyrillic	Ethiopic Extended	Gurmukhi	CJK Extension C (2.8MB)
Cyrillic Supplement	Ethiopic Extended-A	Kaithi	CJK Extension D
Cyrillic Extended-A	N'Ko	Kannada	(see also Unihan Database)
Cyrillic Extended-B	Osmanya	Kharoshthi	CJK Compatibility Ideographs (.5MB)
Georgian	Tifinagh	Lepcha	CJK Compatibility Ideographs Supplement
Georgian Supplement	Vai	Limbu	CJK Radicals / KangXi Radicals
Glagolitic	Middle Eastern Scripts	Malayalam	CJK Radicals Supplement
Gothic	Arabic	Meetei Mayek	CJK Strokes
Greek	Arabic Supplement	Ol Chiki	Ideographic Description Characters
Greek Extended	Arabic Presentation Forms-A	Oriya	Hangul Jamo
Latin	Arabic Presentation Forms-B	Saurashtra	Hangul Jamo Extended-A
Latin-1 Supplement	Aramaic, Imperial	Sinhala	Hangul Jamo Extended-B
Latin Extended-A	Avestan	Syloti Nagri	Hangul Compatibility Jamo
Latin Extended-B	Carian	Tamil	Halfwidth Jamo
Latin Extended-C	Cuneiform (1MB)	Telugu	Hangul Syllables (.7MB)
Latin Extended-D	Cuneiform Numbers and Punctuation	Thaana	Hiragana
Latin Extended Additional	Old Persian	Vedic Extensions	Katakana
Latin Ligatures	Ugaritic	Southeast Asian Scripts	Katakana Phonetic Extensions
Fullwidth Latin Letters	Hebrew	Batak	Kana Supplement
Linear B	Hebrew Presentation Forms	Balinese	Halfwidth Katakana
Linear B Syllabary	Lycian	Buginese	Kanbun
Linear B Ideograms	Lydian	Cham	Lisu
Ogham	Mandaic	Javanese	Yi
Old Italic	Old South Arabian	Kayah Li	Yi Syllables (.5MB)
Phaistos Disc	Pahlavi, Inscriptional	Khmer	Yi Radicals
Runic	Parthian, Inscriptional	Khmer Symbols	American Scripts
Shavian	Phoenician	Lao	Cherokee
Phonetic Symbols	Samaritan	Myanmar	Deseret
IPA Extensions	Syriac	Myanmar Extended-A	Unified Canadian Aboriginal Syllabics
Phonetic Extensions	Central Asian Scripts	New Tai Lue	UCAS Extended
Phonetic Extensions Supplement	Mongolian	Rejang	Other
Modifier Tone Letters	Old Turkic	Sundanese	Alphabetic Presentation Forms
Spacing Modifier Letters	Phags-Pa	Tai Le	Halfwidth and Fullwidth Forms
Superscripts and Subscripts	Tibetan	Tai Tham	ASCII Characters
Combining Diacritics		Tai Viet	
Combining Diacritical Marks		Thai	
Combining Diacritical Marks Supplement		Philippine Scripts	
Combining Half Marks		Buhid	
		Hanunoo	
		Taaglog	

[illegible]

UNICODE OVERVIEW

Questi supplementary character li conoscete..?
Code point da 1F600_H in poi

unicode.org/emoji/charts/full-emoji-list.html

Full Emoji List, v13.1

[Index & Help](#) | [Images & Rights](#) | [Spec](#) | [Proposing Additions](#)

This chart provides a list of the Unicode emoji characters and sequences, with images from different vendors, CLDR name, date, source, and keywords. The ordering of the emoji is by code point in the Code column. [Recently-added emoji](#) are marked by a © in the name and outlined images; their images may show as a group with "..." before and after.

































































































Emoji with skin-tones are not listed here: see [Full Skin Tone List](#).

For counts of emoji, see [Emoji Counts](#).

While these charts use a particular version of the [Unicode Emoji data files](#), the images and format may be updated at any time. For any production usage, consult those data files. For the column header, for further information, see [Index & Help](#).

Smileys & Emotion

face-smiling

	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600													grinning face
2	U+1F601													grinning face with big eyes
3	U+1F602													grinning face with smiling eyes
4	U+1F603													beaming face with smiling eyes
5	U+1F604													grinning squinting face
6	U+1F605													grinning face with sweat
7	U+1F606													rolling on the floor laughing
8	U+1F607													face with tears of joy

CARATTERI: SINTASSI

- Java / C#: **char** (tipo primitivo in Java)
- Scala / Kotlin: **Char** (tipo di oggetto)
- Conversioni carattere \leftrightarrow intero:
 - in Java e Scala, conversione *automatica*
 - in C#, conversione classica tramite cast (nel verso `int` \rightarrow `char`)
 - in Kotlin, conversione *esplicita* tramite appositi metodi `toXX()`

```
public static void main(String[] args)
{
    char ch = 'A';
    int x = ch;
    ch = 72;
    System.out.println(ch);
}
```

Java Run

Java

```
C# Playground by Alen Vadassery
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         char ch = 'A';
8         int x = ch;
9         ch = (char) 72;
10        Console.WriteLine(ch);
11    }
12 }
```

C#

```
fun main(args: Array<String>) : Unit {
    var ch : Char = 'A';
    var x : Int = ch.toInt();
    ch = 72.toChar();
    println(ch);
}
```

Kotlin

```
def main(args: Array[String]) : Unit = {
    var ch : Char = 'A';
    var x : Int = ch;
    ch = 72;
    println(ch);
}
```

Scala



DA UNICODE A UTF

- Le migliaia di caratteri possibili in Unicode pongono il problema di come specificare caratteri *non presenti sulle tastiere*
- Si usa una *codifica semi-numerica*: ' `\u2122` '
- Unicode però si limita ad assegnare codici ai caratteri:
non dice come debbano essere mappati su sequenze di byte
- **UTF = Unicode Transformation Format**
a mapping from every Unicode code point to a unique byte sequence
 - UTF-8 a lunghezza variabile, usa da 1 a 4 byte per carattere
→ molto usato per testo, email, ...
 - UTF-16 a lunghezza variabile, usa 2 o 4 byte; i caratteri extra sono rappresentati da *coppie di codici* usando il range riservato fra D800-DFFF → programmazione
 - UTF-32 usa sempre 4 byte → semplice, ma vorace

PERCHÉ UTF?

- Unicode elenca migliaia di caratteri e li numera, ma ci sono tanti modi di «scrivere concretamente» quei numeri
- Storicamente, ogni piattaforma faceva un po' da sé
 - ASCII standard per tutti, ma solo per i 127 caratteri inglesi..
 - .. da lì in poi, tanti standard diversi incompatibili fra loro
 - e non parliamo del ritorno a capo: CR o CR+LF? (Mac/Unix vs Win)
- **UTF** è una sorta di «*lingua franca*» per far interoperare macchine e piattaforme *anche molto diverse fra loro*
 - UTF-8 in particolare è usatissimo per le email, o negli editor *per assicurarsi che il formato sia leggibile anche su altri computer*
 - se un testo non è UTF e lo condividi con qualcun altro (che magari ha un Mac mentre tu hai Windows), molti caratteri risulteranno «sbagliati» o illeggibili – a partire dalle *lettere accentate!*

UTF-8

• UTF-8

- usa 1 solo byte per i primi 128 caratteri → **compatibile ASCII**
- usa 2 byte per i successivi 1920 caratteri (quasi tutti i più usati)
- usa 3 byte per i rimanenti caratteri del Basic Multilingual
- usa 4 byte solo per gli altri piani Unicode (tra cui molte Emoji..)

UTF-8 online calculator

www.browserling.com/tools/utf8-encode

www.browserling.com/tools/utf8-decode

U+1F60x		😊	😄	😁	😆	😅	😂	😇	😈	😉	😊	😋	😌	😍	😎	😏
U+1F61x	😊		😄	😁	😆		😅		😂		😇		😈		😉	
U+1F62x	😊	😋	😌	😍	😎			😏	😐	😑						
U+1F63x	😒	😓	😔	😕		😗	😘	😙	😚	😛	😜	😝	😞	😟	😠	😡
U+1F64x	😢					😤	😥	😦	😧	😨	😩	😪	😫	😬	😭	😮

char	Code point	Valore in binario	UTF-8
'\$'	U+0024	010 0100 (7 bit significativi)	00100100
'£'	U+00A3	000 1010 0011 (11 bit significativi)	11000010 1010 0011
'€'	U+20AC	0010 0000 1010 1100 (16 bit sign.)	11100010 10000010 10101100
😊	U+01F608	0 0001 1111 0110 0000 1000 (21 bit)	11110000 10011111 10011000 10001000

UTF-8

- UTF-8

- usa 1 solo byte per i primi 128 caratteri → **compatibile ASCII**
- usa 2 byte per i successivi 1920 caratteri (quasi tutti i più usati)
- usa 3 byte per i rimanenti caratteri del Basic Multilingual
- usa 4 byte solo per gli altri piani Unicode (tra cui molte Emoji..)

UTF-8 online calculator

www.browserling.com/tools/utf8-encode

www.browserling.com/tools/utf8-decode

U+1F60x		😊	😄	😁	😆	😅	😂	😇	😈	😉	😊	😋	😌	😍	😎	😏
U+1F61x	😊		😄	😁	😆		😅		😂		😇		😈		😉	
U+1F62x	😊	😋	😌	😍	😎			😏	😐	😑						
U+1F63x	😒	😓	😔	😕		😗	😘	😙	😚	😛	😜	😝	😞	😟	😠	😡
U+1F64x	😢					😤	😥	😦	😧	😨	😩	😪	😫	😬	😭	😮

char	Code point	UTF-8
'\$'	U+0024	00
'£'	U+00A3	10 0011
'€'	U+20AC	010 10101100
😊	U+01F608	0 0001 1111 0110 0000 1000 (21 bit) 11110000 10011111 10011000 10001000

Occhio alle Emoji nei messaggi.. !
Sono supplementary character, occupano l'equivalente di 4 caratteri standard ciascuno!

UTF-16

- UTF-16

- usa 2 byte per i primi 65536 caratteri (il Basic Multilingual)
- usa 4 byte per gli altri piani Unicode
- **più complesso ma efficiente → usato in Java, .NET, macOS (Cocoa)**
- per distinguere le sequenze di 2 byte da quelle di 4 byte, queste ultime sono rappresentate tramite una *coppia di valori* nel range D800-DFFF, che Unicode (Basic Multilingual) mantiene riservati

char	Code point	Valore in binario	UTF-16
'\$'	U+0024	010 0100 (7 bit significativi)	00000000 00100100
'£'	U+00A3	000 1010 0011 (11 bit significativi)	00000000 1010 0011
'€'	U+20AC	0010 0000 1010 1100 (16 bit sign.)	00100000 10101100
🐱	U+01F608	0 0001 1111 0110 0000 1000 (21 bit)	11011000 00111101 11011110 00001000

\uD83D \uDE08

UTF-16


- UTF-16

- i caratteri fino a FFFF sono codificati senza modifiche
- quelli da 10000 in su, si esprimono come *coppia surrogata*

UTF-16 online calculator

<https://convertcodes.com/utf16-encode-decode-convert-string/>


- MA l'ordine con cui sono memorizzati in memoria può variare
→ 4 sotto-codifiche lecite: UTF-16 (2 versioni), UTF-16LE, UTF-16BE

char	Code point	UTF-16 binario		UTF-16 hex	
	U+01F608	11011000	00111101 11011110 00001000	D8 3D DE 08	
UTF-16 hex		UTF-16 (LE)	UTF-16 (BE)	UTF-16BE	UTF-16LE
D8 3D DE 08		FF,FE,3D,D8,08,DE	FE,FF,D8,3D,DE,08	D8,3D,DE,08	3D,D8,08,DE

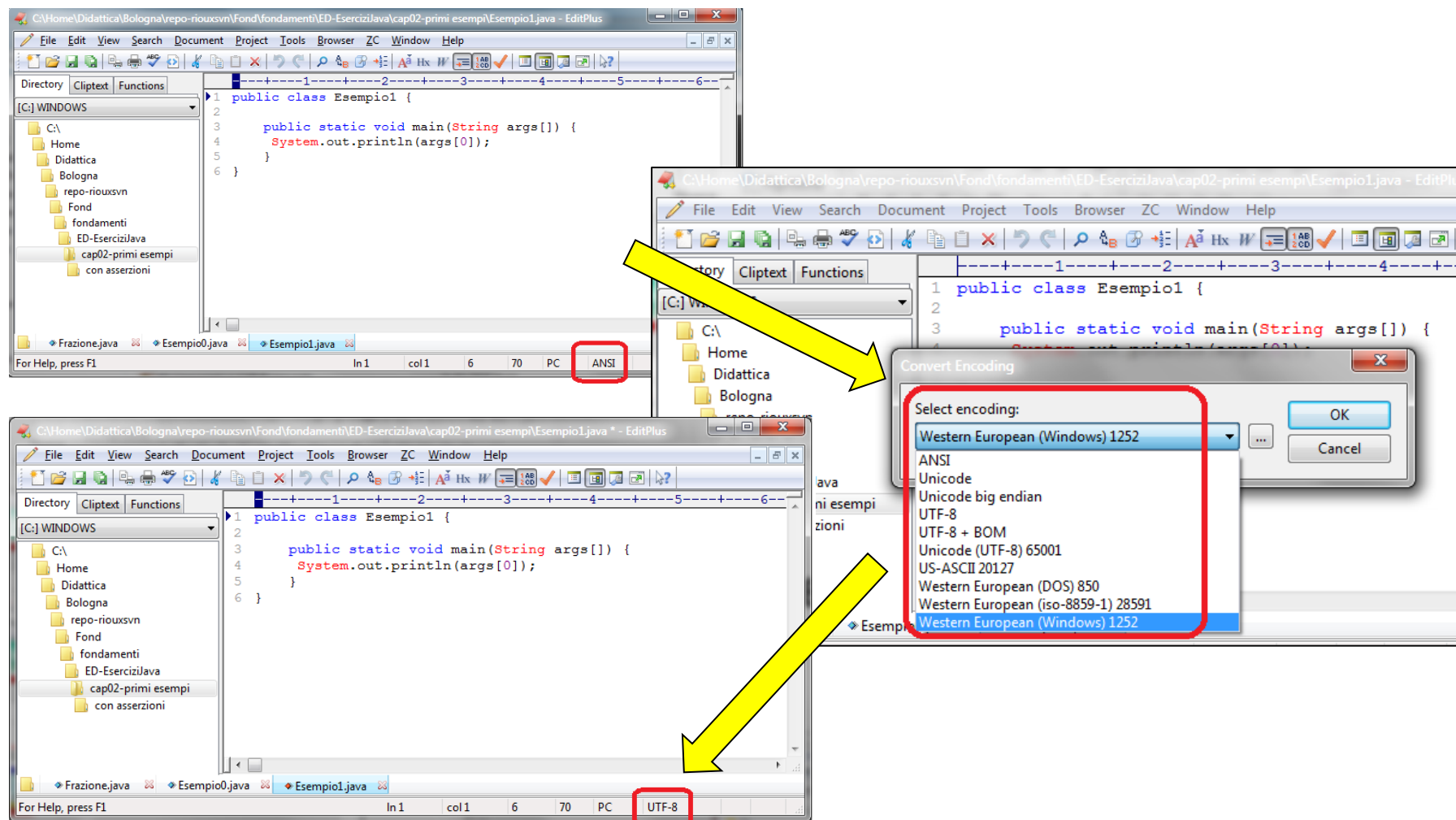
\uFEFF è un marcatore usato per distinguere in modo automatico Little Endian da Big Endian

UTF-32

- UTF-32
 - usa sempre e comunque 4 byte per tutti i caratteri Unicode
 - molto semplice, MA usa una *quantità sproporzionata di memoria!*
 - nel 99,99% dei casi, i caratteri usati sono Basic Multilingual, che richiederebbero solo 2 byte; per non parlare dei primi 128 caratteri ASCII, che ne richiederebbero uno solo!
 - Morale: vantaggio più apparente che reale (gli editor di testo devono comunque gestire i caratteri combinati, gli ideogrammi..)

char	Code point	Valore in binario	UTF-32
'\$'	U+0024	010 0100 (7 bit significativi)	0000000000000000 000000000100100
'£'	U+00A3	000 1010 0011 (11 bit significativi)	0000000000000000 0000000010100011
'€'	U+20AC	0010 0000 1010 1100 (16 bit sign.)	0000000000000000 0010000010101100
	U+01F608	0 0001 1111 0110 0000 1000 (21 bit)	00000000000000001 1111011000001000

UTF NEGLI EDITOR



UTF NEGLI EDITOR

