



# Corso Java



# Caratteristiche principali di Java

- Java è un linguaggio di alto livello e orientato agli oggetti, creato dalla Sun Microsystems nel 1995.

Le motivazioni, che guidarono lo sviluppo di Java, erano quelle di creare un linguaggio semplice e familiare.

Le caratteristiche del linguaggio di programmazione Java sono:

- La tipologia di linguaggio orientato agli oggetti (ereditarietà, polimorfismo, ...)
- la gestione della memoria effettuata automaticamente dal sistema che si preoccupa dell'allocazione e della successiva deallocazione della memoria
- la portabilità, cioè la capacità di un programma di poter essere eseguito su piattaforme diverse senza dover essere modificato e ricompilato

# Caratteristiche di Java

- Semplice e familiare
- Orientato agli oggetti
- Indipendente dalla piattaforma
- interpretato
- Sicuro
- Robusto
- Distribuito e dinamico
- Multi-thread

# Semplice e familiare

- Basato su C
- Sviluppato da zero
- Estremamente semplice: senza puntatori, macro, registri
- Apprendimento rapido
- Semplificazione della programmazione
- Riduzione del numero di errori

# Orientato agli oggetti

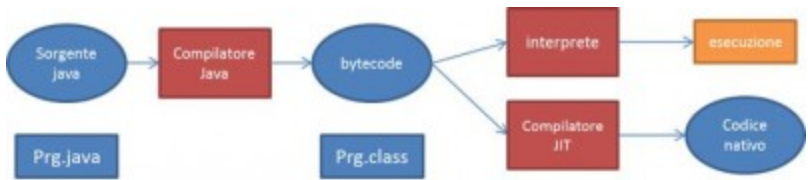
- Orientato agli oggetti dalla base
- In Java tutto è un oggetto
- Incorpora le caratteristiche
  - Incapsulamento
  - Polimorfismo
  - Ereditarietà
- Collegamento dinamico
- Non sono disponibili
  - Ereditarietà multipla
  - Overload degli operatori

# Indipendente dalla piattaforma

- Più efficiente di altri linguaggi interpretati
- Soluzione: la macchina virtuale: JVM
- Linguaggio macchina bytecode

# Interpretato

- Il bytecode deve essere interpretato



- Vantaggi rispetto ad altri linguaggi interpretati
- Codice più compatto
- Efficiente
- Codice confidenziale (non esposto)

## **sicuro**

- Supporta la sicurezza di tipo sandboxing
- Verifica del bytecode
- Altre misure di sicurezza
- Caricatore di classi
- Restrizioni nell'accesso alla rete



# Robusto

- L'esecuzione nella JVM impedisce di bloccare il sistema
- L'assegnazione dei tipi è molto restrittiva
- La gestione della memoria è sempre a carico del sistema
- Il controllo del codice avviene sia a tempo di compilazione sia a tempo di esecuzione (runtime)

## **Distribuito e dinamico**

- Disegnato per un'esecuzione remota e distribuita
- Sistema dinamico
- Classe collegata quando è richiesta
- Può essere caricata via rete
- Dinamicamente estensibile
- Disegnato per adattarsi ad ambienti in evoluzione

# Multi-thread

- Soluzione semplice ed elegante per la multiprogrammazione
- Un programma può lanciare differenti processi
- Non si tratta di nuovi processi, condividono il codice e le variabili col processo principale
- Simultaneamente si possono svolgere vari compiti

# Versioni Java

Version	Release date	End of Free Public Updates	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?

# Versioni Java (nuovo approccio Oracle)

Version	Release date	End of Free Public Updates	Extended Support Until
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	At least October 2024 for AdoptOpenJDK At least September 2027 for Amazon Corretto	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	Extended Support until September 2021

# Espressioni aritmetiche

```
public class Triangolo {  
    public static void main ( String [] args ) {  
        System.out.println (5*10/2);  
    }  
}
```

Il programma risolve l'espressione  $5 \cdot 10 / 2$  e stampa il risultato a video

singoli "letterali"

- Letterali interi: 3425, 12, -34, 0, -4, 34, -1234, ....
- Letterali frazionari: 3.4, 5.2, -0.1, 0.0, -12.45, 1235.3423, ....

operatori aritmetici

- moltiplicazione \*
- divisione /
- modulo % (resto della divisione tra interi)
- addizione +
- sottrazione -

Le operazioni sono elencate in **ordine decrescente di priorità** ossia  $3+2*5$  fa 13, non 25

Le parentesi tonde cambiano l'ordine di valutazione degli operatori ossia  $(3+2)*5$  fa 25

Inoltre, tutti gli operatori sono associativi a sinistra ossia  $3+2+5$  corrisponde a  $(3+2)+5$

## operazione di divisione

- L'operazione di divisione / si comporta diversamente a seconda che sia applicato a letterali interi o frazionari
- $25/2 = 12$  (divisione intera)
- $25\%2 = 1$  (resto della divisione intera)
- $25.0/2.0 = 12.5$  (divisione reale)
- $25.0\%2.0 = 1.0$  (resto della divisione intera)
- Una operazione tra un letterale intero e un frazionario viene eseguita come tra due frazionari
- $25/2.0 = 12.5$
- $1.5 + (25/2) = 13.5$  (attenzione all'ordine di esecuzione delle operazioni)
- $2 + (25.0/2.0) = 14.5$



# Operatori

## Operatori aritmetici

- Di assegnazione: = += -= \*= /= &= |= ^=
- Di assegnazione/incremento: ++ -- %=
- Operatori Aritmetici: + - \* / %

Operatore	Significato
+	addizione
-	sottrazione

# Operatori di assegnazione

Operatore	Significato
=	addizione
+=	addizione assegnazione
-=	sottrazione assegnazione
*=	motiplicazione assegnazione
/=	divisione assegnazione
%=	resto assegnazione

# Operatori relazionali

== != > < >= <=

Operatore	Significato
<	minore di
<=	minore di o uguale a
>	maggiore di
>=	maggiore di o uguale a
==	uguale a
!=	non uguale / diverso

# Operatori per Booleani

- Bitwise (interi): & | ^ << >> ~

Operatore	Significato
&&	short circuit AND
	short circuit OR
!	NOT

## Attenzione:

- Gli operatori logici agiscono **solo su booleani**
  - Un intero NON viene considerato un booleano
  - Gli operatori relazionali forniscono valori booleani

# Operatori su reference

**Per i riferimenti/reference, sono definiti:**

- Gli operatori relazionali == e !=
  - test sul riferimento all'oggetto, **NON sull'oggetto**
- Le assegnazioni
- L'operatore "punto"
- NON è prevista l'aritmetica dei puntatori, vengono gestiti dalla JVM

# Operatori matematici

Operazioni matematiche complesse sono permesse dalla **classe Math** (package `java.lang`)

- `Math.sin (x)` calcola  $\sin(x)$
- `Math.sqrt (x)` calcola  $x^{(1/2)}$
- `Math.PI` ritorna `pi`
- `Math.abs (x)` calcola  $|x|$
- `Math.exp (x)` calcola  $e^x$
- `Math.pow (x, y)` calcola  $x^y$

Esempio

- `z = Math.sin (x) - Math.PI / Math.sqrt(y)`

# Caratteri speciali

Literal	Represents
<code>\n</code>	New line
<code>\t</code>	Horizontal tab
<code>\b</code>	Backspace
<code>\r</code>	Carriage return
<code>\f</code>	Form feed
<code>\\</code>	Backslash
<code>\"</code>	Double quote
<code>\ddd</code>	Octal character

# Le variabili e le costanti

- Una variabile è un'area di memoria identificata da un nome
- Il suo scopo è di contenere un valore di un certo tipo
- Serve per memorizzare dati durante l'esecuzione di un programma
- Il nome di una variabile è un **identificatore**
  - può essere costituito da lettere, numeri e underscore
  - non deve coincidere con una parola chiave del linguaggio
  - è meglio scegliere un **identificatore** che sia **significativo** per il programma



## esempio

```
public class Triangolo {  
    public static void main ( String [] args ) {  
  
        int base , altezza ;  
        int area ;  
  
        base = 5;  
        altezza = 10;  
        area = base * altezza / 2;  
  
        System.out.println ( area );  
    }  
}
```

Usando le variabili il programma risulta essere **più chiaro**:

- Si capisce meglio quali siano la base e l'altezza del triangolo
- Si capisce meglio che cosa calcola il programma

# Dichiarazione

- In Java ogni variabile deve essere **dichiarata prima del suo uso**
- Nella dichiarazione di una variabile se ne specifica il **nome** e il **tipo**
- Nell'esempio, abbiamo dichiarato tre variabili con nomi base, altezza e area, tutte di tipo int (numeri interi)
  - int base , altezza ;
  - int area ;

**ATTENZIONE!** Ogni variabile deve essere dichiarata **UNA SOLA VOLTA**  
(la prima volta che compare nel programma)

```
base =5;  
altezza =10;  
area = base * altezza /2;
```

# Assegnazione

- Si può memorizzare un valore in una variabile tramite l'operazione di assegnazione
- Il valore da assegnare a una variabile può essere un letterale o il risultato della valutazione di un'espressione
- Esempi:

```
base =5;  
altezza =10;  
area = base * altezza /2;
```

- I valori di base e altezza vengono letti e usati nell'espressione
- Il risultato dell'espressione viene scritto nella variabile area

## Dichiarazione + Assegnazione

Prima di poter essere usata in un'espressione una variabile deve:

- essere stata dichiarata
- essere stata assegnata almeno una volta (inizializzata)
- **NB: si possono combinare dichiarazione e assegnazione.**

Ad esempio:

```
int base = 5;  
int altezza = 10;  
int area = base * altezza / 2;
```

# Costanti

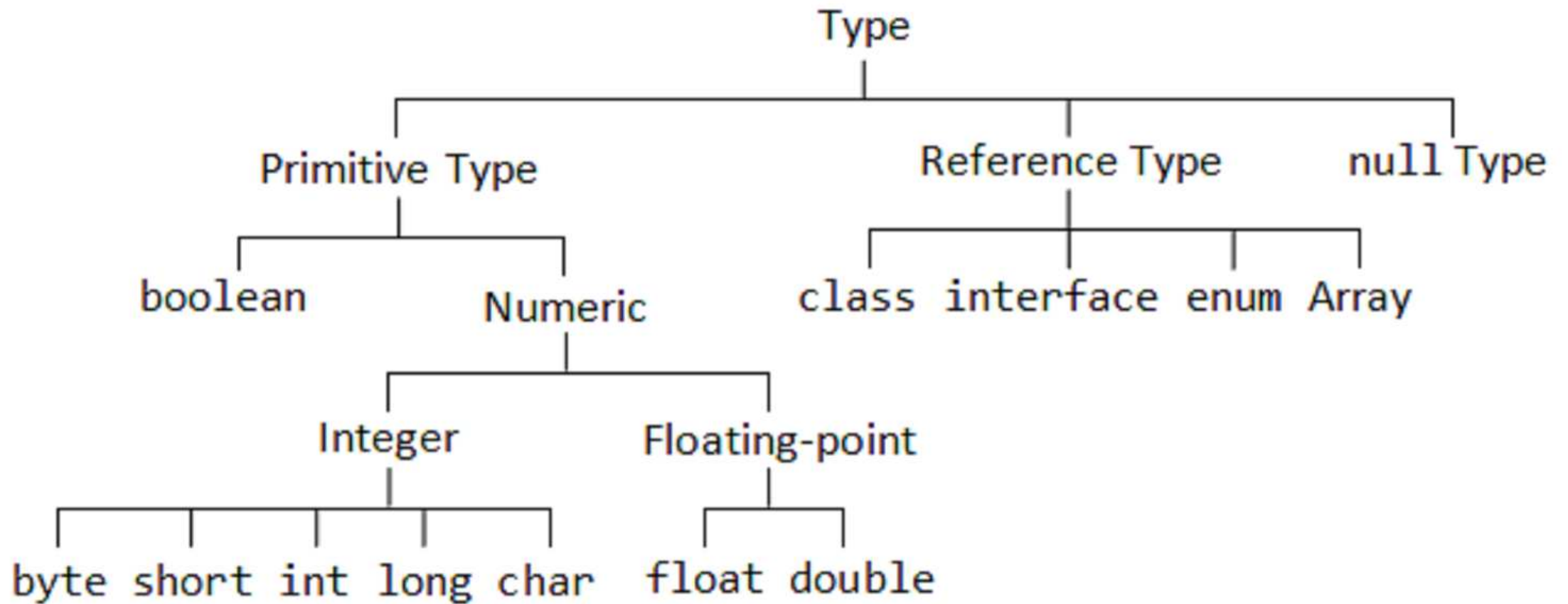
Nella dichiarazione delle variabili che **NON DEVONO** mai cambiare valore si può utilizzare il modificatore **final**

```
final double IVA = 0.22;
```

- Il modificatore **final** trasforma la variabile in una costante
- Il compilatore si occuperà di controllare che il valore delle costanti non venga **mai modificato** (ri-assegnato) dopo essere stato inizializzato.
- Aggiungere il modificatore **final** non cambia funzionamento programma, ma serve a prevenire errori di programmazione
- Si chiede al compilatore di controllare che una variabile non venga ri-assegnata per sbaglio
- Sapendo che una variabile non cambierà mai valore, il compilatore può anche eseguire delle **ottimizzazioni** sull'uso di tale variabile.

## Input dall'utente

- Per ricevere valori in input dall'utente si può usare la classe `Scanner`, contenuta nel package **`java.util`**
- La classe `Scanner` deve essere richiamata usando la direttiva `import` prima dell'inizio del corpo della classe



## Java definisce alcuni tipi primitivi

- Per efficienza Java definisce tipi primitivi
- La dichiarazione di una istanza alloca spazio in memoria

# Tabelle riassuntive: tipi di dato

## Primitive Data Types

type	bits
byte	8 bit
short	16 bit
int	32 bit
long	64 bit
float	32 bit
double	64 bit

**I caratteri sono considerati interi**



## I tipi numerici, i char

- Esempi
- `123` (int)
- `256789L` (L o l = long)
- `0567` (ottale) `0xff34` (hex)
- `123.75` `0.12375e+3` (float o double)
- `'a'` `'%'` `'\n'` (char)
- `'\123'` (\ introduce codice ASCII)

## Tipo boolean

- `true`
- `false`

# Esempi

```
int i = 15;  
long longValue = 1000000000000001;  
byte b = (byte)254;  
  
float f = 26.012f;  
double d = 123.567;  
boolean isDone = true;  
boolean isGood = false;  
char ch = 'a';  
char ch2 = ';';
```

```
// dichiarazione e inizializzazione contemporanea
short mioShort = 851;
System.out.println(mioShort);

long mioLong = 34093L;
System.out.println(mioLong);

double mioDouble = 3.14159732;
System.out.println(mioDouble);

float mioFloat = 324.4f;
System.out.println(mioFloat);

char mioChar = 'y';
System.out.println(mioChar);

boolean mioBoolean = true;
System.out.println(mioBoolean);

byte mioByte = 127;
System.out.println(mioByte);
}
}
```

Data Type	Bits	Minimum	Maximum
byte	8	-128	127
short	16	-32,768	32,767
int	32	-2,147,483,648	2,147,483,647
long	64	-9.22337E+18	9.22337E+18

# Il controllo del flusso

Java mette a disposizione del programmatore diverse strutture sintattiche per consentire il **controllo del flusso**

## Selezione, scelta condizionale

### if statements

```
if (condition) {  
    //statements;  
}
```

## else if

```
[optional]  
else if (condition2) {  
    //statements;  
}
```

# else

```
[optional]  
else {  
  
//statements;  
  
}
```

## switch Statements

```
switch (espressione) {  
    case valore1:  
        //statements;  
        break;  
    ...  
    case valoren:  
        //statements;  
        break;  
    default:  
        //statements;  
}
```

# Cicli definiti

Se il numero di iterazioni è prevedibile dal contenuto delle variabili all'inizio del ciclo.

```
for (init; condition; adjustment) {  
    //statements;  
}
```

Esempio: prima di entrare nel ciclo so già che verrà ripetuto 10 volte

```
int n=10;  
for (int i=0; i<n; ++i) {  
    ...  
}
```



# Cicli indefiniti

Se il numero di iterazioni non è noto all'inizio del ciclo.

```
while (condition) {  
    //statements;  
}
```

```
do {
```

```
//statements;
```

```
} while (condition);
```

Esempio: il numero di iterazioni dipende dai valori immessi dall'utente.

```
while(true) {  
    x = Integer.parseInt(JOptionPane.showInputDialog("Immetti numero positivo"));  
    if (x > 0) break;  
}
```

## Cicli annidati

Se un ciclo appare nel corpo di un altro ciclo.

Esempio: stampa quadrato di asterischi di lato n

```
for (int i=0; i<n; i++) {  
    for (int j=0; j<n; j++) System.out.print("*");  
    System.out.println();  
}
```

## Cicli con filtro

Vengono passati in rassegna un insieme di valori e per ognuno di essi viene fatto un test per verificare se il valore ha o meno una certa proprietà in base alla quale decideremo se prenderlo in considerazione o meno.

Esempio: stampa tutti i numeri pari fino a 100

```
for (int i=1; i<100; ++i) { // passa in rassegna tutti i numeri fra 1 e 100
    if (i % 2 == 0) // filtra quelli pari
        System.out.println(i);
}
```

## Cicli con filtro e interruzione

Se il ciclo viene interrotto dopo aver filtrato un valore con una data proprietà.

Esempio: verifica se un array contiene o meno numeri negativi

```
boolean trovato = false;
for (int i=0; i<v.length; ++i) // passa in rassegna tutti gli indici dell'array v
    if (v[i]<0) { // filtra le celle che contengono valori negativi
        trovato = true;
        break; // interrompe ciclo
    }
// qui trovato vale true se e solo se vi sono numeri negativi in v
```

# Cicli con accumulatore

Vengono passati in rassegna un insieme di valori e ne viene tenuta una traccia cumulativa usando una opportuna variabile.

Esempio: somma i primi 100 numeri interi.

```
int somma = 0; // variabile accumulatore di tipo int
for (int i=1; i<100; ++i) { // passa in rassegna tutti i numeri fra 1 e 100
    somma = somma + i; // accumula i valori nella variabile accumulatore
}
```

Esempio: data una stringa s, ottieni la stringa rovesciata

```
String rovesciata = ""; // variabile accumulatore di tipo String
for (int i=0; i<s.length(); ++i) { // passa in rassegna tutti gli indici dei caratteri di s
    rovesciata = s.substring(i, i+1) + rovesciata; // accumula i caratteri in testa all'accumulatore
}
```

## Cicli misti

Esempio di ciclo definito con filtro e accumulatore: calcola la somma dei soli valori positivi di un array

```
int somma = 0;
for (int i=0; i<v.length; ++i) // passa in rassegna tutti gli indici dell'array v
    if (v[i]>0) // filtra le celle che contengono valori positivi
        somma = somma + v[i]; // accumula valore nella variabile accumulatore
```



# Array

- Sequenze ordinate di
  - Tipi primitivi (int, float, etc.)
  - Riferimenti ad oggetti ( vedere classi! )
- Elementi dello stesso tipo
  - Indirizzati da indici
  - Raggiungibili con l'operatore di indicizzazione: le **parentesi quadre** []
  - Raggruppati sotto lo stesso nome

## In Java gli array sono Oggetti

- Sono allocati nell'area di memoria riservata agli oggetti creati dinamicamente (heap)

### Dimensione

- Può essere stabilita a run-time (quando l'oggetto viene creato)
- È fissa (non può essere modificata)
- E' nota e ricavabile per ogni array

# Array Mono-dimensionali (vettori)

Dichiarazione di un riferimento a un array

- `int[] voti;`
- `int voti[];`

La dichiarazione di un array non assegna alcuno spazio

```
voti == null
```

# Creazione di un Array

L'operatore new crea un array:

- Con costante numerica

```
int[] voti;  
...  
voti = new int[10];
```

- Con costante simbolica

```
final int ARRAY_SIZE = 10;  
int[] voti;  
...  
voti = new int[ARRAY_SIZE];
```

- Con valore definito a run-time

```
int[] voti;  
... definizione di x (run-time) ...  
voti = new int[x];
```

*\*Utilizzando un inizializzatore-*

(che permette anche di riempire l'array)

- L'operatore new inizializza le variabili
  - 0 - per variabili di tipo numerico (inclusi i char)
  - false - per le variabili di tipo boolean

```
int[] primi = {2, 3, 5, 7, 11, 13};  
...  
int [] pari = {0, 2, 4, 6, 8, 10,};  
// La virgola finale e' facoltativa  
// (elenchi lunghi)
```

- Dichiarazione e creazione possono avvenire contestualmente
- L'attributo length indica la lunghezza dell'array, cioè il numero di elementi
- Gli elementi vanno da 0 a length-1

```
for (int i=0; i<voti.length; i++)  
voti[i] = i;
```

## **In Java viene fatto il bounds checking**

- Maggior sicurezza
- Maggior lentezza di accesso

## **Il riferimento ad array**

- Non è un puntatore al primo elemento
- È un puntatore all'oggetto array
- Incrementandolo non si ottiene il secondo elemento



## Array di oggetti

Per gli array di oggetti (e.g., Integer) `Integer [] voti = new Integer [5];` ogni elemento e' un riferimento

**L'inizializzazione va completata con quella dei singoli elementi**

```
voti[0] = new Integer (1);  
voti[1] = new Integer (2);  
...  
voti[4] = new Integer (5);
```

# Array Multi-dimensional (Matrici)

## Array contenenti riferimenti ad altri array

Sintatticamente sono estensioni degli array a una dimensione

## Sono possibili righe di lunghezza diverse

(matrice = array di array)

```
int[][] triangle = new int[3][]
```

## Le righe non sono memorizzate in posizioni adiacenti

- Possono essere spostate facilmente

```
// Scambio di due righe
double[][] saldo = new double[5][6];
...
double[] temp = saldo[i];
saldo[i] = saldo[j];
saldo[j] = temp;
```

- L'array è una struttura dati efficiente ogni volta che il numero di elementi è noto
- Il ridimensionamento di un array in Java risulta poco efficiente
- Utilizzare altre strutture dati se il numero di elementi contenuto non è noto

## Il pacchetto `java.util` contiene metodi statici di utilità per gli array

- Copia di un valore in tutti gli (o alcuni) elementi di un array
  - `Arrays.fill (<array>, <value>);`
  - `Arrays.fill (<array>, <from>, <to>, <value>);`
- Copia di array
  - `System.arraycopy (<arraySrc>, <offsetSrc>, <arrayDst>, <offsetDst>, <#elements>);`
- Confronta due array
  - `Arrays.equals (<array1>, <array2>);`
- Ordina un array (di oggetti che implementino l'interfaccia `Comparable`)
  - `Arrays.sort (<array>);`
- Ricerca binaria (o dicotomica)
  - `Arrays.binarySearch (<array>);`

# Esempi di Array

## Array Monodimensionali

```
int[] list = new int[10];  
  
list.length;  
  
int[] list = {1, 2, 3, 4};
```

## Array Multidimensionali

```
int[][] list = new int[10][10];  
list.length;  
list[0].length;  
int[][] list = {{1, 2}, {3, 4}};
```

## Array irregolari

```
int[][] m = {  
    {1, 2, 3, 4},  
    {1, 2, 3},  
    {1, 2},  
    {1}  
};
```

esempi ed esercizi su array



# Stringhe e Caratteri

Caratteristiche principali

## Classi disponibili

- String
  - Modella stringhe (sequenze – array di caratteri)
  - **Non modificabile** (dichiarata final)
- StringBuilder
  - Modificabile
- StringBuffer (non si usa più)
  - Modificabile
- Character
- CharacterSet

## Definizione

```
String myString;
```

```
myString = new String ("stringa esempio");
```

- Oppure

```
String myString = new String ("stringa esempio");
```

- Solo per il tipo String vale

```
String myString = "stringa esempio";
```

- Il carattere " (doppi apici) può essere incluso come "
- Il nome della stringa è il riferimento alla stringa stessa
- Confrontare due stringhe NON significa confrontare i riferimenti

**NB: I metodi che gestiscono il tipo String NON modificano la stringa, ma ne creano una nuova**

# Concatenare stringhe

- Operatore concat

- `myString1.concat(myString2)`
- `String s2 = "Ciao".concat(" a tutti").concat("!");`
- `String s2 = "Ciao".concat(" a tutti".concat("!"));`

- Utile per definire stringhe che occupano più di una riga

- Operatore +

`"questa stringa" + "e formata da tre" + "stringhe"`

- La concatenazione funziona anche con altri tipi, che vengono automaticamente convertiti in stringhe

```
System.out.println ("pi Greco = " + 3.14);
```

```
System.out.println ("x = " + x);
```

## Lunghezza stringa

- `int length()`
  - `myString.length()`
  - `"Ciao".length()` restituisce 4
  - `"".length()` restituisce 0
- Se la lunghezza è N, i caratteri sono indicizzati da 0 a N-1

## Carattere i-esimo

- `char charAt (int)`
- `myString.charAt(i)`

## Confronta stringa con altra stringa

- `boolean equals(String s)`
  - \* `myString.equals("stringa")` ritorna true o false
- `boolean equalsIgnoreCase(String s)`
- `myString.equalsIgnoreCase("StRiNgA")`

## Confronta con altra stringa facendone la differenza

- `int compareTo(String str)`
- `myString.compareTo("stringa")` ritorna un valore  $\geq < 0$

## Trasforma int in String

- `String valueOf(int)`
- Disponibile per tutti tipi primitivi

## Restituisce indice prima occorrenza di c

- `int indexOf(char c)`
- `int indexOf(char c, int fromCtrN)`

## Altri metodi

- `String toUpperCase(String str)`
- `String toLowerCase(String str)`
- `String substring(int startIndex, int endIndex)`
- `String substring(int startIndex)`

## Esempio

```
String s1, s2;  
s1 = new String("Prima stringa");  
s2 = new String("Prima stringa");  
System.out.println(s1);  
/// Prima stringa  
System.out.println("Lunghezza di s1 = " +  
s1.length());  
// 26  
if (s1.equals(s2)) ...  
// true  
if (s1 == s2) ...  
// false  
String s3 = s3.substring (2, 6);  
// s3 == "ima s"
```

[altri esempi sulle stringhe](#)

# Casting e promotion

- `( nometipo ) variabile`
- `( nometipo ) espressione`
- Trasforma il valore della variabile (espressione) in quello corrispondente in un tipo diverso
- Il cast si applica anche a `char`, visto come tipo intero positivo
- La promotion è automatica quando necessaria
  - Es. `double d = 3 + 4;`
- Il casting deve essere esplicito: il programmatore si assume la responsabilità di eventuali perdite di informazione
  - Per esempio
  - `int i = ( int ) 3.0 * ( int ) 4.5;` i assume il valore 12
  - `int i = ( int ) ( 3.0 * 4.5 );` i assume il valore 13



# casting dei tipi reference (oggetti)

- è permesso solo in caso di **ereditarietà**
- la conversione da sotto-classe a super-classe è **automatica**
- la conversione da super-classe a sotto-classe richiede **cast esplicito**
- la conversione tra riferimenti non in relazione tra loro **non è permessa**

## esempio promotion

```
char a = 'a';  
// promotion int è più grande e i valori sono compatibili  
int b = a;  
  
System.out.println(a); // a  
System.out.println(b); // 97
```

## esempi type casting

```
byte b = (byte) 261;  
System.out.println(b); // 5  
  
System.out.println( Integer.toBinaryString(b) ); // 101  
System.out.println( Integer.toBinaryString(261) ); // 100000101
```

```
int a = (int) 1936.27;  
  
System.out.println(a); // 1936
```

## con il tipo boolean non si può fare il typecasting

```
int a = (int) true; // vietato - ... cannot be converted to ...  
boolean falso = (boolean) 0; // vietato - ... cannot be converted to ...
```

# metodo

- Termine caratteristico dei linguaggi OOP
- Un **insieme di istruzioni con un nome**
- Uno strumento per risolvere gradualmente i problemi scomponendoli in **sottoproblemi**
- Uno strumento per **strutturare** il codice
- Uno strumento per **ri-utilizzare** il lavoro già svolto
- Uno strumento per rendere il **programma più chiaro** e leggibile

# quando e perché usare i metodi

1. Quando il programma da realizzare è articolato diventa conveniente identificare **sottoproblemi** che possono essere risolti individualmente
2. scrivere **sottoprogrammi** che risolvono i sottoproblemi richiamare i **sottoprogrammi** dal programma principale (main)
3. Questo approccio prende il nome di **programmazione procedurale** (o astrazione funzionale)
4. In Java i **sottoprogrammi** si realizzano tramite **metodi ausiliari**
5. Sinonimi usati in altri linguaggi di programmazione: **funzioni**, **procedure** e (sub)**routines**

- **metodi statici**: dichiarati `static`
- richiamabili attraverso nome della classe
- **p.es**: `Math.sqrt()`

```
public class ProvaMetodi
{
    public static void main(String[] args) {
        stampaUno();
        stampaUno();
        stampaDue();
    }

    public static void stampaUno() {
        System.out.println("Hello World");
    }

    public static void stampaDue() {
        stampaUno();
        stampaUno();
    }
}
```

## Metodi non static

- I metodi **non static** rappresentano operazioni effettuabili su singoli oggetti
- La documentazione indica per ogni metodo il tipo ritornato e la lista degli argomenti formali che rappresentano i dati che il metodo deve ricevere in ingresso da chi lo invoca
- Per ogni argomento formale sono specificati:
  - un tipo (primitivo o reference)
  - un nome (identificatore che segue le regole di naming)

# Invocazione di metodi non static

- L'invocazione di un metodo non static su un oggetto istanza della classe in cui il metodo è definito si effettua con la sintassi:
- Ogni volta che si invoca un metodo si deve specificare una lista di argomenti attuali
- Gli argomenti attuali e formali sono in corrispondenza posizionale
- Gli argomenti attuali possono essere delle variabili o delle espressioni
- Gli argomenti attuali devono rispettare il tipo attribuito agli argomenti formali
- La documentazione di ogni classe (istanziabile o no) contiene l'elenco dei metodi disponibili
- La classe **Math** non è istanziabile
- La classe **String** è "istanziabile ibrida"
- La classe **StringBuilder** è "istanziabile pura"



## Metodi predicativi

Un metodo che restituisce un tipo primitivo `boolean` si definisce **predicativo** e può essere utilizzato direttamente in una condizione.

In inglese sono spesso introdotti da `is` oppure `has`: `isMale()`, `hasNext()`.

Esempi sui metodi ausiliari