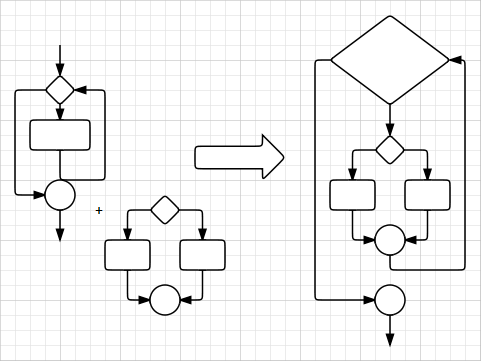
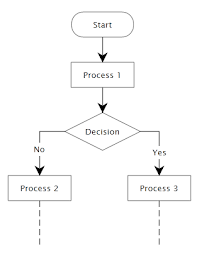
# Il controllo del flusso

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

* IF – ELSE
* WHILE e DO-WHILE
* FOR e FOREACH
* SWITCH - CASE



## Selezione, scelta condizionale



if statements

* E' un’istruzione condizionale, permette cioè di eseguire un blocco di istruzioni solo se si verifica una determinata condizione.

if (condition) {

//statements;

}

## else if [opzionale]

* Indichiamo anche cosa fare se non si supera la condizione

else if (condition2) {

//statements;

}

## else [opzionale]

else {

//statements;

}

## Switch Statements

Serve per gestire in maniera più ordinata varie condizioni, è un modo più elegante in alcune situazioni.

switch (espressione) {

case valore1:

//statements;

break;

...

case valoren:

//statements;

break;

default:

//statements;

}

## Cicli definiti - for

Se il numero di iterazioni è prevedibile.

for (partenza, fine, incremento)

blocco di istruzioni da ripetere

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 - while

si ripete il ciclo fintanto che una condizione è verificata (è vera)

* la condizione booleana true/false
* determina la continuazione del programma
* ed esegue l'elenco delle operazioni del blocco
* Da usare se il numero di iterazioni non è noto all'inizio del ciclo.

while (condition) {

//statements;

}

//Esempio: quando il numero di iterazioni dipende da valori in input

while(true) {

x = Integer.parseInt(JOptionPane.showInputDialog("Immetti numero positivo"));

if (x > 0) break;

}

## do-while

* Quando voglio eseguire almeno una volta l'istruzione, anche se la condizione è impostata su false
* Si verifica la condizione dopo il primo ciclo

do {

//statements;

} while (condition);

## Cicli con interruzione: break

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

* Se stiamo eseguendo un ciclo, possiamo utilizzare la parola break per interromperlo in qualsiasi momento.
* Si interrompe quindi il ciclo e si riprende l'esecuzione delle istruzioni al di fuori di esso.

### 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 salto: continue

Invece terminare completamente il ciclo ed uscire fuori, continue interrompe solo l'iterazione corrente e passa alla successiva.

{

int a;

for ( a = 1; a <= 10 ; a++ ) // Run a = 1, 2, ..., 10

{

if ( a == 4 )

{

continue; // Skip printing 4...

}

System.out.println(a); // Print a

}

}

## Cicli con condizione

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 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

## 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();

}

## codice esempi d'uso

* [raccolta esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/03_Condizioni_Cicli.md)
* [altri esempi](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow)
* [01\_if-elseif-else](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/01_if-elseif-else)
* [02\_switch](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/02_switch)
* [03\_while](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/03_while)
* [04\_for](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/04_for)
* [05\_foreach](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/05_foreach)
* [06\_labels](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/06_labels)
* [giochi](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/giochi)
* [programmi](https://github.com/maboglia/CorsoJava/tree/master/esempi/02_control-flow/programmi)

# Array

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

## Il riferimento ad array

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

## Cosa sono gli array?

Nel linguaggio di programmazione Java, gli array sono oggetti (§4.3.1), vengono creati dinamicamente e possono essere assegnati a variabili di tipo Object (§4.3.2). Tutti i metodi della classe Object possono essere invocati su un array. [Java docs](https://docs.oracle.com/javase/specs/jls/se7/html/jls-10.html)

In Java gli array sono Oggetti

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

### Dimensione dell'array

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

## 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];

## riempire l'array utilizzando un inizializzatore

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

int[] numeriPrimi = {2,3,5,7,11,13};

//...

// La virgola finale e' facoltativa

int [] numeriPari = {0, 2, 4, 6, 8, 10,};

* 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
* Accesso più lento

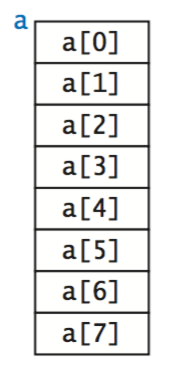
## Array Mono-dimensionali (vettori)

Dichiarazione di un riferimento a un array

* int[] voti;
* int voti[];

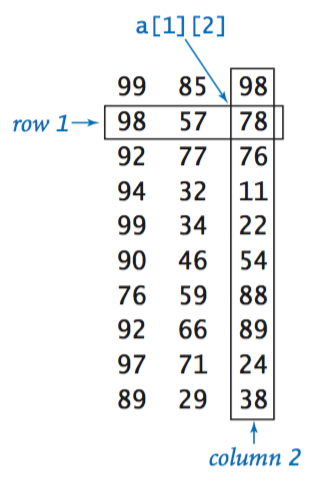
La dichiarazione di un array non assegna alcuno spazio

voti == null



## Array Multi-dimensionali

* Array contenenti riferimenti ad altri array
* Sintatticamente sono estensioni degli array a una dimensione
* matrice: array di array, regolare, irregolare, rettangolare...



* 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

## 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 Sono possibili righe di lunghezza diverse\*\*

int[][] m = {

{1, 2, 3, 4},

{1, 2, 3},

{1, 2},

{1}

};

## 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);

## Algoritmi per lavorare con gli array: la classe java.util.Arrays

* 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>);

### altre risorse

* [raccolta esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/05_Array.md)
* [altri esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/03_array)
* [esempi ed esercizi su array](https://gist.github.com/maboglia/112d52611f8fa02c1772e0b48405fc72)

# Stringhe e Caratteri

La gestione delle stringhe è fondamentale in Java, e il linguaggio offre un'ampia varietà di metodi e funzionalità per manipolare le stringhe. Ecco alcuni concetti e operazioni comuni nella gestione delle stringhe in Java:

### Creazione di Stringhe

// Dichiarazione e inizializzazione di una stringa

String str1 = "Hello, World!";

// Creazione di una stringa utilizzando il costruttore

String str2 = new String("Java String");

// Concatenazione di stringhe

String concatString = str1 + " " + str2;

### Metodi Principali

// Lunghezza di una stringa

int length = str1.length();

// Ottenere un carattere in posizione specifica

char charAtIndex = str1.charAt(7);

// Confronto di stringhe (sensibile alle maiuscole/minuscole)

boolean areEqual = str1.equals(str2);

// Confronto di stringhe (ignorando maiuscole/minuscole)

boolean areEqualIgnoreCase = str1.equalsIgnoreCase("hello, world!");

// Ricerca di una sottostringa

boolean containsSubstring = str1.contains("Hello");

// Estrazione di sottostringhe

String substring = str1.substring(0, 5);

// Conversione di stringhe in minuscolo/maiuscolo

String lowercase = str1.toLowerCase();

String uppercase = str1.toUpperCase();

// Rimozione degli spazi bianchi iniziali e finali

String trimmedString = " Trim Me ".trim();

### Concatenazione di Stringhe

// Utilizzo dell'operatore +

String fullName = firstName + " " + lastName;

// Utilizzo del metodo concat()

String greeting = "Hello".concat(" World");

// Utilizzo di StringBuilder per concatenazioni efficienti

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append("Java");

stringBuilder.append(" StringBuilder");

String result = stringBuilder.toString();

### Formattazione di Stringhe

// Utilizzo di String.format()

String formattedString = String.format("Il prezzo è %.2f", 12.3456);

// Utilizzo di StringBuilder per la costruzione di stringhe complesse

StringBuilder complexString = new StringBuilder();

complexString.append("Utente: ").append(username).append(", Punteggio: ").append(score);

### Conversione tra Stringhe e Tipi Primitivi

// Conversione da String a int

int intValue = Integer.parseInt("123");

// Conversione da int a String

String stringValue = String.valueOf(456);

// Altre conversioni con i metodi parseXXX()

double doubleValue = Double.parseDouble("3.14");

boolean booleanValue = Boolean.parseBoolean("true");

### Manipolazione di Caratteri

// Ottenere l'indice di un carattere

int index = str1.indexOf('W');

// Sostituire caratteri

String replacedString = str1.replace('l', 'z');

// Divisione di una stringa in base a un delimitatore

String[] parts = str1.split(",");

// Trasformazione di una stringa in un array di caratteri

char[] charArray = str1.toCharArray();

### Verifica di Null e Stringhe Vuote

// Verifica se una stringa è nulla o vuota

if (str1 == null || str1.isEmpty()) {

// Fai qualcosa

}

// Verifica se una stringa è nulla o contiene solo spazi bianchi

if (str1 == null || str1.trim().isEmpty()) {

// Fai qualcosa

}

Questi sono solo alcuni degli aspetti principali della gestione delle stringhe in Java. La classe String offre molte altre funzionalità che possono essere utili a seconda delle esigenze specifiche dell'applicazione. Inoltre, l'uso di StringBuilder è consigliato per operazioni di concatenazione efficienti, specialmente quando si lavora con molte operazioni di modifica di stringhe.

## 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 l'inizializzazione

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

### 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](https://github.com/maboglia/CorsoJava/tree/master/esempi/04_Stringhe.md)

### 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);

NB: La classe String crea nuovi oggetti ogni volta che concateni con +, meglio usare la classe StringBuilder...

### Carattere i-esimo

* char charAt (int)
* myString.charAt(i)

### Lunghezza stringa

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

### 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 >=< 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)

### compareTo

int compareTo(String other)

Esegue una comparazione lessicale. Ritorna un intero:

* < 0 se la stringa corrente è minore della stringa other
* = 0 se le due stringhe sono identiche
* > 0 se la stringa corrente è maggiore di other

#### indexOf

int indexOf(int ch)

Restituisce l’indice del carattere specificato

#### lastIndexOf

int lastIndexOf(int ch)

E’ come indexOf() ma viene restituito l’indice dell’ultima occorrenza trovata

#### length

int length()

Restituisce il numero di caratteri di cui è costituita la stringa corrente

#### replace

String replace(char oldChar, char newChar)

Restituisce una nuova stringa, dove tutte le occorrenze di oldChar sono rimpiazzate con newChar

#### substring

String substring(int startIndex)

Restituisce una sottostringa della stringa corrente, composta dai caratteri che partono dall’indice startIndex alla fine

#### substring

String substring(int startIndex, int number)

Restituisce una sottostringa della stringa corrente, composta dal numero number di caratteri che partono dall’indice startIndex

#### toLowerCase

String toLowerCase()

Restituisce una nuova stringa equivalente a quella corrente ma con tutti i caratteri minuscoli

#### toUpperCase

String toUpperCase()

Restituisce una nuova stringa equivalente a quella corrente ma con tutti i caratteri maiuscoli

## codice esempi d'uso

* [raccolta esempi](https://github.com/maboglia/CorsoJava/blob/master/esempi/04_Stringhe.md)
* [altri esempi](https://github.com/maboglia/CorsoJava/tree/master/esempi/04_API_Java/P01_java.lang/String)
* [stringbuilder](https://github.com/maboglia/CorsoJava/tree/master/esempi/04_API_Java/P01_java.lang/StringBuilder)