

VETTORI IN JAVA

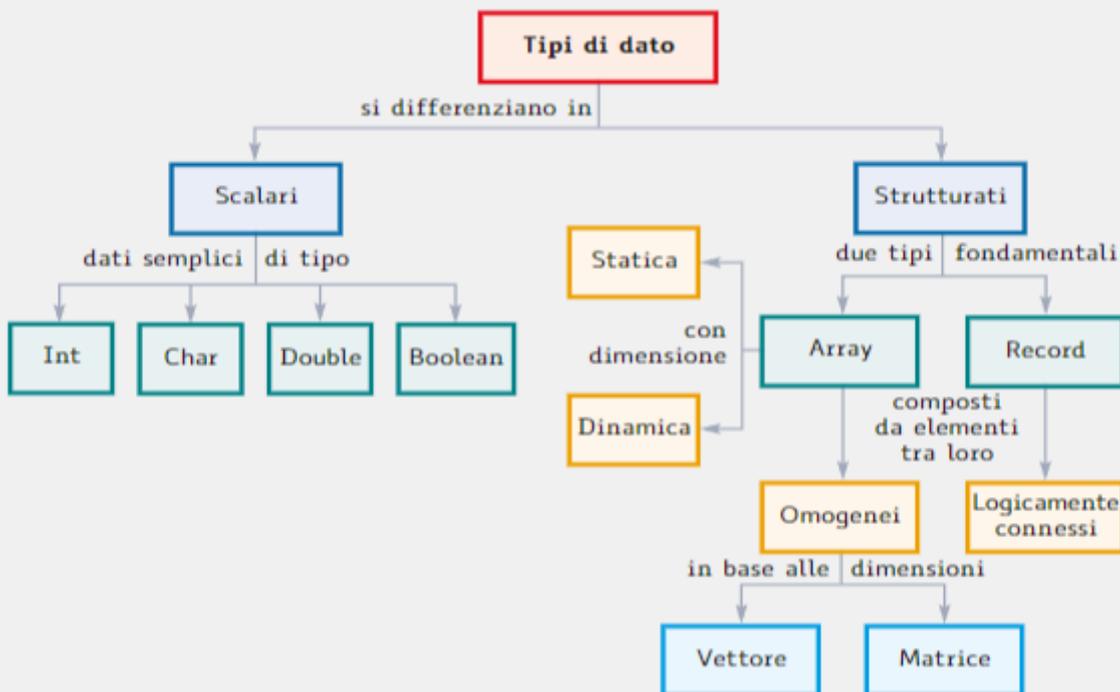
1

Dati strutturati semplici: gli array

IN QUESTA LEZIONE IMPAREREMO...

- a conoscere i dati strutturati
- a utilizzare i vettori

MAPPA CONCETTUALE



Introduzione ai dati strutturati

Le variabili che abbiamo definito e utilizzato sinora rientrano tra i **tipi di dati semplici**, così chiamati in quanto ogni elemento corrisponde a una cella di memoria.



I dati semplici che abbiamo utilizzato sono `int`, `char`, `double`, `boolean` e, se sono disponibili nel linguaggio di programmazione senza la necessità di richiamare librerie, sono anche detti **tipi semplici primitivi** (o scalari).

Inoltre, queste variabili sono memorizzate nella **RAM**, cioè in una memoria di tipo elettronico che, quando cessa di essere alimentata, perde il suo contenuto: sono quindi **dati volatili** e non permanenti.

Tutti i moderni linguaggi di programmazione permettono di effettuare dei “raggruppamenti” di questi **dati semplici** per realizzare le cosiddette **strutture di dati** (o **dati strutturati**).

I dati strutturati sono strutture di dati ottenute mediante la composizione di altri dati di tipo semplice.

I **dati strutturati predefiniti** utilizzabili nei linguaggi di programmazione sono chiamati:

- **vettori/matrici** o **array**;
- **record** o **struct**.

Vedremo, nel prosieguo della trattazione, che un **array** è una collezione finita di **N** variabili dello stesso tipo, mentre un **record** è un raggruppamento logico di dati tra loro non omogenei.

Tra i raggruppamenti di dati dello stesso tipo includiamo anche le **stringhe**, che altro non sono se non un raggruppamento di variabili di tipo **char**.

È anche possibile realizzare una struttura dati partendo da un'altra struttura dati invece che da dati semplici: in questo caso, si ottiene una **struttura di dati complessa**.

Per esempio, potremmo realizzare un vettore composto da stringhe, oppure un vettore composto da vettori.



Una delle **proprietà** delle **strutture dati** è determinata dalla loro **dimensione**: se questa non varia per l'intero ciclo di vita della struttura, da quando cioè vengono definite e create le variabili fino al termine del loro utilizzo, la **struttura dati** è definita **statica**, se invece la dimensione cambia durante l'esecuzione del programma la struttura dati è definita **dinamica**.

Il vettore o array monodimensionale

L'**array** è uno strumento, o meglio un oggetto, che permette di **aggregare dati omogenei** per poterli facilmente elaborare, cioè memorizzare, ritrovare e manipolare.

Idealmente, l'**array** è costituito da tante **variabili** semplici “affiancate” tra loro, con una caratteristica comune: devono essere **tutte dello stesso tipo**, cioè tutte variabili **numeriche**, oppure **booleane**, oppure **carattere**.

Non è possibile aggregare mediante l'array variabili di tipo diverso.

L'**array** prende il nome di **vettore** quando gli elementi sono disposti secondo una sola dimensione, cioè “in una sola riga” oppure in “una sola colonna”.

Un **vettore** può essere immaginato come una “cassettiera” composta da un insieme definito di cassette, in quantità diverse a seconda delle necessità: il programmatore “costruisce” il proprio vettore in base alle dimensioni dello specifico problema da risolvere.



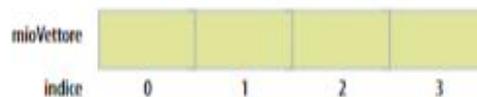
Ogni “cassetto” corrisponde a una **variabile**, e viene individuato in base alla sua **posizione** all'interno del **vettore**: il primo “cassetto” ha posizione 0, il secondo posizione 1 ecc.

Il numero intero che indica la posizione dell'elemento nel vettore si chiama **indice**; esso è il numero d'ordine che **individua univocamente l'elemento**; ogni **posizione all'interno del vettore** prende anche il nome di cella.

Come per le variabili semplici, anche per le variabili strutturate è necessario un **identificatore** che ne permetta l'utilizzo: a ogni variabile viene quindi associato un nome, la cui scelta rispetta le convenzioni adottate per i nomi delle variabili semplici.

ESEMPIO

Un array di dimensione quattro con nome **mioVettore** può essere "visualizzato" come nella figura seguente.



Dichiarazione di variabili di tipo vettore

Per poter utilizzare una variabile di tipo array bisogna definire i seguenti tre elementi:

- la **natura** delle variabili che dovrà contenere (cioè, il loro **tipo**);
- il **numero** delle variabili, quindi delle celle del vettore (la **dimensione**);
- il **nome** che sarà associato all'array (l'**identificatore**).

In **linguaggio di progetto** la dichiarazione di un vettore presenta la seguente struttura.

```
<tipo><nomeVettore>[<I dimensione>]
```

Quindi, il vettore usato nell'esempio precedente, verrà così definito:

```
int mioVettore [4] //dichiara un vettore di 4 elementi
```

L'operatore che indica l'**array** è costituito dalla coppia di parentesi **[]**.

Al momento della dichiarazione di **mioVettore**, vengono riservate (allocate) **4 locazioni di memoria consecutive**, ciascuna contenente una variabile di tipo intero: è quindi necessario indicare il numero delle celle con un **valore costante**, in quanto al momento della compilazione tale valore deve essere noto.

Manipolazione di vettori

Definito il vettore con le sue celle, si procede con le operazioni di **manipolazione dei dati**:

- l'**inserimento** di un elemento in una cella;
- la **lettura** del contenuto di una cella.

L'inserimento di un valore in una cella avviene mediante un'istruzione di assegnazione con cui si indica in quale posizione del vettore si vuole memorizzare il valore.

Per esempio, con l'istruzione riportata di seguito, il valore 4 viene inserito nella seconda posizione del vettore (cioè nella locazione di memoria corrispondente al secondo elemento del vettore).

```
mioVettore [1] = 4;
```

La scrittura `mioVettore[1]` denota l'elemento del vettore `mioVettore` di **indice 1** e l'assegnazione precedente può essere letta da destra verso sinistra come "il valore 4 viene assegnato alla cella di posizione 1 dell'array `mioVettore`" oppure, più sinteticamente, "assegno 4 alla posizione 1 di `mioVettore`".

Completiamo l'esempio inserendo altri valori nelle altre celle.

```
mioVettore[0]=2;  
mioVettore[2]=6;  
mioVettore[3]=8;
```

Graficamente, la situazione risultante è la seguente.

mioVettore	2	4	6	8
indice	0	1	2	3



Avremmo ottenuto la medesima situazione finale utilizzando un ciclo a conteggio:

```
int mioVettore[4]  
per x ← 0 a 3 fai  
  mioVettore[x] ← 2+(x*2)  
finePer
```

Il **ciclo a conteggio** è particolarmente **indicato per operare con i vettori**, dato che siamo proprio in presenza di situazioni nelle quali è definito il numero delle operazioni da eseguire, in quanto conosciamo a priori la dimensione del vettore.

I vettori in Java

In Java un **array** è un **oggetto** che realizza una raccolta di dati che siano tutti dello stesso tipo; viene definito con la seguente sintassi:

```
<tipo> <nomeVettore>=new <tipo> [<costante>];
```

dove <tipo> può essere sia un **tipo semplice**, quindi intero, reale, carattere sia, come vedremo, un **tipo composto** oppure un **oggetto**.



La primitiva **new** utilizzata nell'istruzione ci indica che l'elemento creato è un oggetto: come vedremo in seguito, all'interno dell'identificatore dell'array viene memorizzato il **riferimento** (indirizzo) alla **posizione di memoria** dove viene allocato l'array.

Definiamo come primo esempio un vettore di 3 elementi di tipo intero:

```
int mioVettore=new int {3};
```



Gli elementi del vettore hanno l'indice che inizia dal valore 0, quindi il primo elemento ha posizione 0, il secondo posizione 1 e il terzo posizione 2.

Memorizziamo, per esempio, la data odierna (7 dicembre 2018) nel vettore mettendo il giorno in posizione 0, il mese in posizione 1 e l'anno in posizione 2: per scrivere in una data cella basta indicarne la sua posizione:

```
mioVettore[0]=7  
mioVettore[1]=12  
mioVettore[2]=2018
```

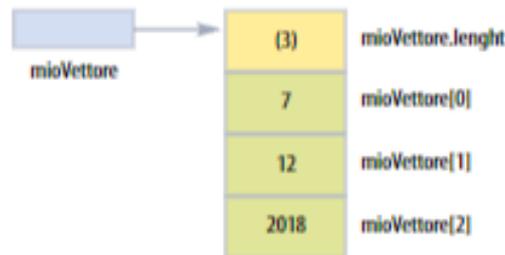
Il risultato è mostrato nella seguente figura:

mioVettore	7	12	2018
indice	0	1	2



La creazione fa un'inizializzazione implicita con valore 0 per **int** e **double**, **false** per i **boolean**.

Agli array viene associato un particolare **attributo**, che "contiene" la dimensione del vettore: ha nome **length**, e viene memorizzato assieme al vettore, come "fosse un suo elemento" (è una *variabile istanza* dell'oggetto).



Riferendoci all'esempio precedente, nella figura possiamo vedere schematicamente come un vettore viene memorizzato, e possiamo ottenere la dimensione di un **array** semplicemente con l'istruzione:

```
int dimensione = mioVettore.length
```

Una volta creato, un array ha comunque **dimensione fissa**, quindi non può essere "allargato" a piacere: **Java** mette a disposizione anche "array con dimensione variabile" mediante la classe **Vector**, che vengono presentati nell'ambito delle **strutture dinamiche**.

Utilizzare i vettori

Realizziamo un primo programma che utilizza i vettori come struttura dati per memorizzare un insieme di valori.

✓ PROBLEMA SVOLTO PASSO PASSO

✓ Il problema

Scriviamo un segmento di codice che definisce un vettore, lo riempie con numeri casuali e ne visualizza il contenuto sullo schermo.

✓ L'analisi e la strategia risolutiva

Come prima operazione, definiamo un vettore di interi con dimensione parametrica, indicando come costante **TANTI** il dato nel quale inseriremo il numero degli elementi del vettore.

Utilizzando una costante manifesta risulterà immediato riutilizzare il programma nel caso in cui ci fossero modifiche di dimensione dei dati: basterà cambiare solo il valore di tale costante.

Analogamente, definiamo una costante per stabilire il valore massimo dei numeri generati casualmente (**MAX**) e utilizziamo un primo ciclo per riempire il vettore e un secondo ciclo per visualizzarne il contenuto sullo schermo.

✓ La pseudocodifica e l'algoritmo risolutivo

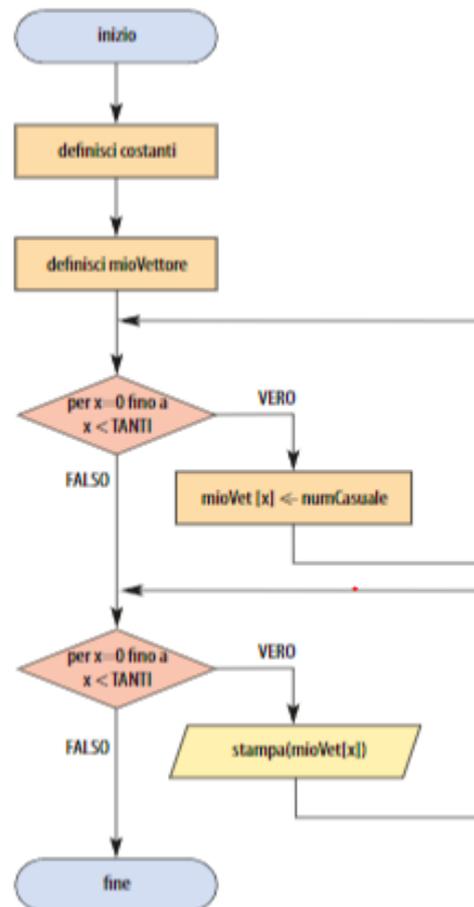
La **pseudocodifica** e il **flow chart** sono riportati di seguito.

l'affinamento

- definisci le variabili
- riempi il vettore
- visualizza il contenuto

Il affinamento - pseudocodifica

```
inizio
int mioVet [TANTI]
per x da 0 a TANTI-1 fai
    mioVet [x] ← numeroCasuale (MAX)
finePer
per x da 0 a TANTI-1 fai
    scrivi(mioVet [x])
finePer
fine
```



✓ La codifica in linguaggio di programmazione Java

Linguaggio Java

```
public class Random{
    final static int TANTI = 8; // costanti condivise
    final static int MAX = 30;
    public static void main(String[] args){
        // creo il vettore di TANTI elementi
        int mioVettore[] = new int[TANTI];
        // riempio il vettore con numeri casuali tra 0 e MAX
        for (int x = 0; x < mioVettore.length; x++){
            mioVettore[x] = (int)(MAX * Math.random());
        }
        // visualizzo il contenuto del vettore
        for (int x = 0; x < mioVettore.length; x++){
            System.out.print(mioVettore[x]+" ");
        }
    }
}
```

✓ L'esecuzione del programma

```
C:\dev\java2>java Random
8 27 10 20 17 10 17 2
```

Il codice sorgente di questo programma lo trovi nel file [Random.java](#).

Se volessimo avere i dati in un **range** con estremo inferiore diverso da 0 sarebbe bastato definire due costanti MIN e MAX e modificare l'istruzione di generazione in:

```
mioVettore[cont] = MIN + (rand() % (MAX-MIN)) // numero tra MIN e MAX in C++
```

```
mioVettore[x] = MIN + (int)((MAX-MIN) * Math.random()); // numero tra MIN e MAX in Java
```

Vediamo un secondo semplice esempio nel quale effettuiamo anche operazioni di I/O.

✓ PROBLEMA SVOLTO PASSO PASSO

✓ Il problema

Scrivi un programma che legge un gruppo di numeri e li visualizza a rovescio.



✓ L'analisi

Ci viene richiesto di leggere una sequenza di interi in input e di visualizzarli a rovescio, cioè simulando una modalità operativa tipo **Last In First Out (LIFO)**, dove cioè l'ultimo elemento inserito è il primo che viene tolto.

Simuliamo quindi una situazione di struttura "a pila", della quale però dobbiamo necessariamente conoscere il numero massimo di elementi per poter dimensionare il vettore che li memorizza.

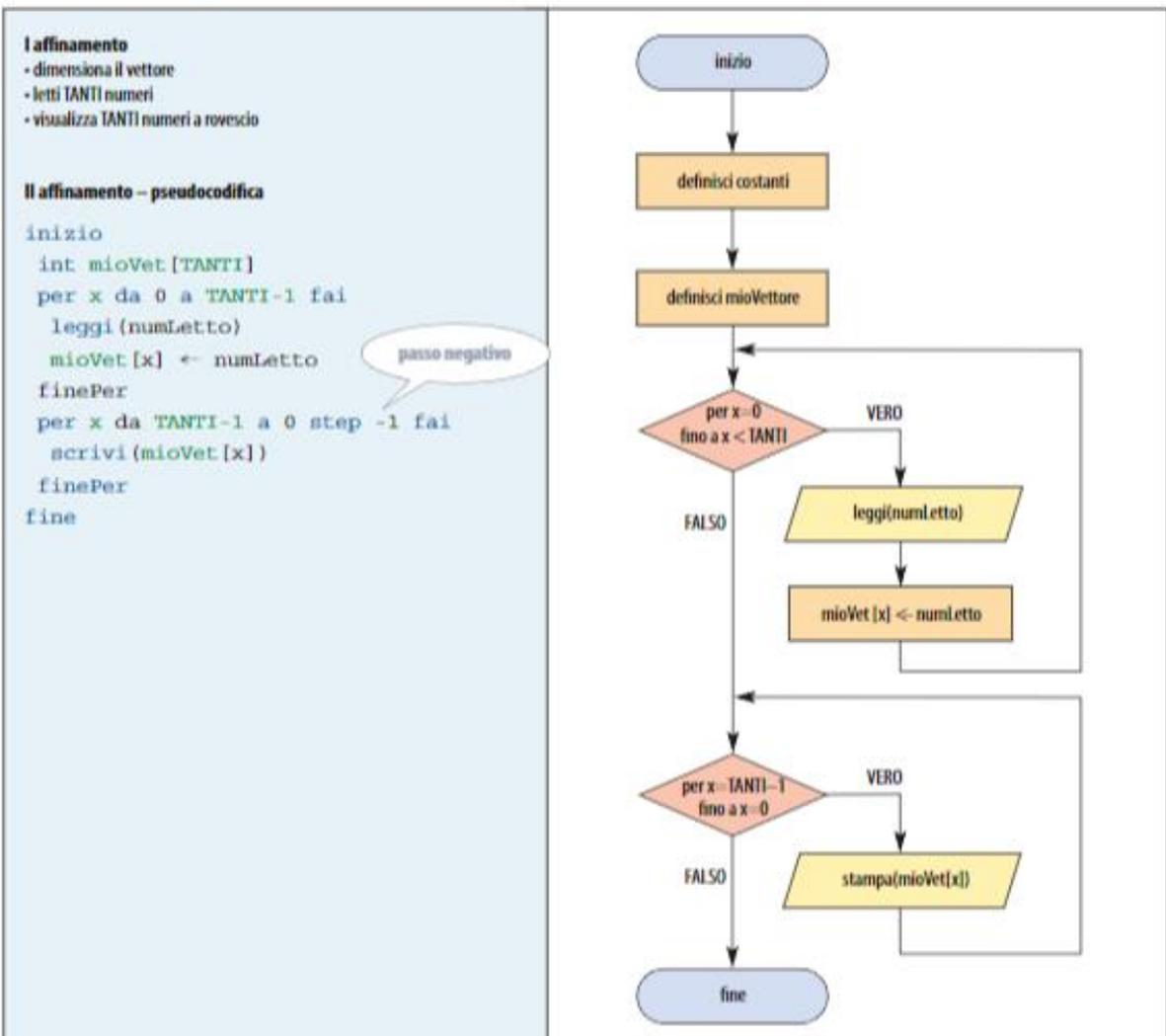
✓ La definizione della strategia

Come prima operazione definiamo la costante **TANTI** che, come in tutti i nostri esempi, contiene il numero che rende parametrica la nostra soluzione.

Il programma avrà una fase di input all'interno di un ciclo a conteggio e, al termine dell'immissione dei dati, un secondo ciclo li visualizzerà sullo schermo, a partire dall'ultimo inserito, cioè decrementando il contatore partendo da **TANTI** per terminare a 0.

✓ La pseudocodifica e l'algoritmo risolutivo

La **pseudocodifica** e il **flow chart** sono riportati di seguito.



✓ La codifica in linguaggio di programmazione

Linguaggio Java

```
import java.util.Scanner;
public class Rovescio{
    final static int TANTI = 6;
    public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        int mioVettore[] = new int[TANTI];
        for(int x = 0; x < mioVettore.length; x++){
            System.out.print("Inserisci un numero: ");
            mioVettore[x]= in.nextInt();
        }
        // visualizza il contenuto del vettore
        System.out.print("\ni numeri a rovescio: ");
        for(int x = TANTI-1; x > -1; x--){
            System.out.print(mioVettore[x]+" ");
        }
    }
}
```