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**: se essi 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**, 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, ed è il concetto fondamentale su cui si basano i **database relazionali**.
Tra i raggruppamenti di dati dello stesso tipo includiamo anche le **stringhe**, che sono 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 essa non varia per l'intero ciclo di vita della struttura, da quando cioè sono 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".

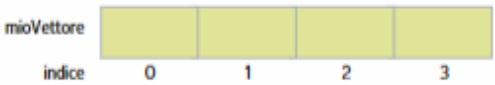
Un **vettore** può essere immaginato come una "cassettiera" composta da un insieme definito di cassetti, in quantità diverse a seconda delle necessità: il programmatore "costruisce" il proprio vettore in base alle dimensioni dello specifico problema da risolvere.

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).

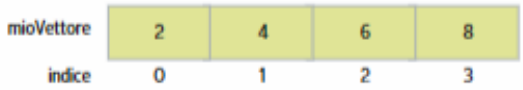
```
mioVettore[1] = 4;
```

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.



> 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 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 o 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: all'interno dell'identificatore dell'array, si memorizza il riferimento (indirizzo) alla posizione di memoria dove viene allocato l'array.

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

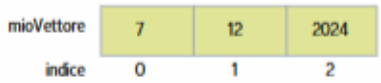
```
int mioVettore = new int[3];                    // vettore di 3 elementi
```

> 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, una data (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:

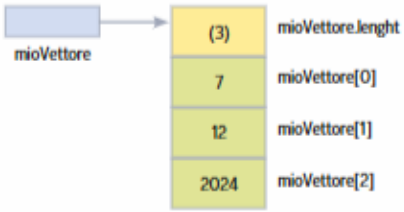
```
mioVettore[0] = 7
mioVettore[1] = 12
mioVettore[2] = 2024
```

Il risultato è mostrato nella seguente figura:



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** ed è 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 una **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

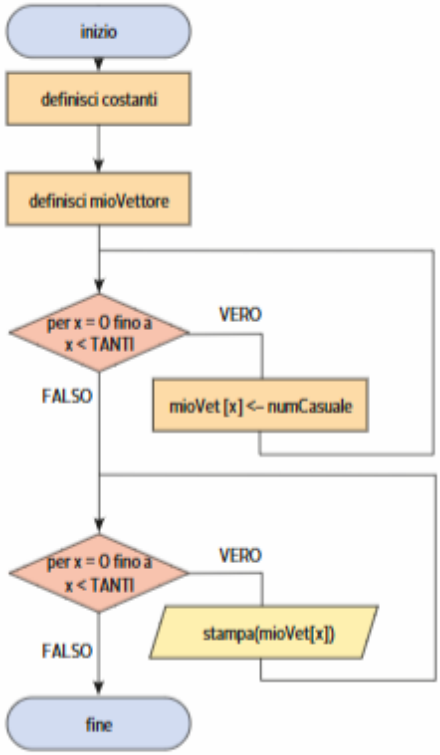
La **pseudocodifica** e il **flow chart** sono riportati di seguito.

L'esecuzione del programma

I affinamento
- **definisci** le variabili
- **riempi** il vettore
- **visualizza** il contenuto

II 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

Random ×

Compila | Annulla | Taglia | Copia | Incolla | Trova... | Chiudi | Codice sorgente

```
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

```
21 15 29 3 13 15 11 6
```

Opzioni

```
22 18 3 10 2 5 20 23
```

BlueJ: BlueJ: Terminale - UA...

Opzioni

```
20 21 11 18 23 16 29 13
```

Can only enter input while your program

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, basterebbe definire due costanti **MIN** e **MAX** e modificare l'istruzione di generazione in:

```
mioVettore[conta] = 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
```

METTITI ALLA PROVA

• **Definizione di un vettore** • **Utilizzo di un vettore**

Scrivi un programma che permette di verificare la **legge dei grandi numeri** mediante il lancio di un dado: leggendo in input il numero di volte che un dado deve essere lanciato, simula questa situazione mediante la generazione di un numero casuale compreso tra 1 e 6; quindi calcola la frequenza con la quale ciascuna delle sei facce viene selezionata all'aumentare dei lanci.
Visualizza le uscite per ciascun valore con le relative frequenze.
Confronta la tua soluzione con quella riportata nel file **Dadi_solux.java**.

Vediamo un altro 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, simulando una modalità operativa tipo **Last In First Out** (**LIFO**), dove 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 la codifica in **Java** sono riportati di seguito.

I affinamento
- **dimensiona** il vettore
- letti TANTI numeri
- **visualizza** TANTI numeri a rovescio

II affinamento – pseudocodifica

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

passo negativo

Rovescio × Rovescio2 ×

Compila | Annulla | Taglia | Copia | Incolla | Trova... | Chiudi | Codice

```
1  import java.util.Scanner;
2  public class Rovescio{
3    final static int TANTI = 6;
4    public static void main(String[] args){
5      Scanner in = new Scanner(System.in);
6      int mioVettore[] = new int[TANTI];
7      System.out.print("Programma che ribalta i numeri\n\n");
8      for(int x = 0; x < mioVettore.length; x++){
9        System.out.print('Inserisci un numero: ');
10       mioVettore[x]= in.nextInt();
11     }
12     // visualizza il contenuto del vettore
13     System.out.print("\ni numeri a rovescio: ");
14     for(int x = TANTI-1; x > -1; x--){
15       System.out.print(mioVettore[x]+" " );
16     }
17   }
18 }
```

Il codice sorgente di questo programma lo trovi nel file **Rovescio.java**.

Una seconda possibile codifica dell'algoritmo utilizza un ciclo a conteggio positivo ed effettua l'elaborazione dell'indice direttamente all'interno dell'istruzione di indicizzazione della cella del vettore.

La **pseudocodifica** e la codifica in **Java** sono riportate di seguito.

I affinamento
- **dimensiona** il vettore
- letti TANTI numeri
- **visualizza** TANTI numeri a rovescio

II affinamento – pseudocodifica

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

passo positivo

Rovescio × Rovescio2 ×

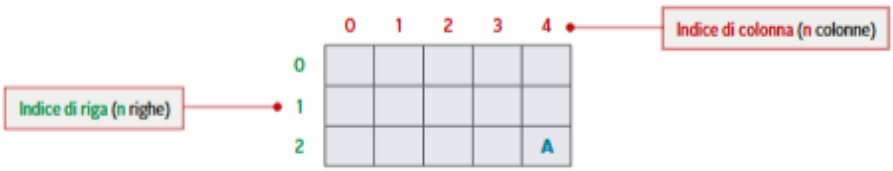
Compila Annulla Taglia Copia Incolla Trova... Chiudi Codice

```
1  import java.util.Scanner;
2  public class Rovescio2{
3    final static int TANTI = 6;
4    public static void main(String[] args){
5      Scanner in = new Scanner(System.in);
6      int mioVettore[] = new int[TANTI];
7      System.out.print("Programma che ribalta i numeri\n\n");
8      for(int x = 0; x < mioVettore.length; x++){
9        System.out.print("Inserisci un numero: ");
10       mioVettore[x]= in.nextInt();
11     }
12     // visualizza il contenuto del vettore
13     System.out.print("\ni numeri a rovescio: ");
14     for(int x = 0; x < mioVettore.length; x++){
15       System.out.print(mioVettore[TANTI-x-1]+" " );
16     }
17   }
18 }
```

Il codice sorgente di questo programma lo trovi nel file **Rovescio2.java**.

Vettori bidimensionali

Gli **array** possono avere più di una dimensione: è infatti possibile definire strutture di dati complesse, gli **array multidimensionali** (o **n-dimensionali**), dove a ogni dimensione viene associato un indice e quindi un elemento viene individuato da un insieme di valori, uno per ogni dimensione.
Le coordinate delle matrici mantengono il nome di **indici**: il primo elemento della coppia prende il nome di **indice di riga**, il secondo di **indice di colonna**.



La definizione della matrice sopra indicata, con 3 righe e 5 colonne, ha la seguente scrittura:

```
int miaMatrice[3][5]                    // dichiara una matrice di 3 righe e 5 colonne
```

Vediamo il suo utilizzo in un programma.

UN ESEMPIO COMPLETO: TEMPERATURE ESTIVE

PROBLEMA SVOLTO PASSO PASSO

Il problema
Una stazione metereologica esegue ogni giorno un insieme di letture per le ore più calde (11:00-16:00). Si vuole sapere, relativamente alla settimana di Ferragosto, in quale giorno e ora si è registrata la temperatura massima e qual è stata la temperatura media per ogni fascia oraria.

L'analisi e la strategia risolutiva
Per prima cosa determiniamo la dimensione del problema: dobbiamo effettuare un insieme di letture per una settimana (7 giorni) a intervalli orari (5 letture). Definiamo quindi due costanti GIORNI = 7 e ORE = 5 come estremi per la matrice miaMat[GIORNI][ORE], nella quale inseriremo le temperature, espresse per comodità in gradi kelvin, quindi con numeri interi.

Successivamente dobbiamo individuare:

- la temperatura più alta, cioè l'elemento di valore maggiore, presente nella tabella;
- la temperatura media per ciascuna fascia oraria, cioè la media per ogni colonna.

Per memorizzare l'elemento maggiore abbiamo bisogno di tre variabili:

- la prima deve contenere il valore;
- le altre due devono contenere le "coordinate" giorno-ora in cui si è registrata questa temperatura.

Per comodità definiremo tre **variabili** **tempMAX**, **oraMAX**, **giornoMAX** e una funzione **calcolaMassimo()** che le aggiorna in base ai dati analizzati.
Per memorizzare le temperature medie, dobbiamo definire un vettore di 5 celle che conterrà il valore medio delle temperature di ogni fascia e sarà calcolato da un'apposita funzione **calcolaMedia()**.

La pseudocodifica e l'algoritmo risolutivo

La **pseudocodifica** e il codice in **linguaggio Java** sono riportati di seguito.

I affinamento	II affinamento
• **definisci** le variabili	GIORNI ← 7 ORE ← 5 int miaMat[GIORNI][ORE] int media[GIORNI]
• **effettua** le letture giornaliere	funzione leggiMatriceManuale() per tutti gli elementi leggi la temperatura
• **visualizza** il contenuto	funzione mostraMatrice()
• **calcola** e **visualizza** la temperatura massima	funzione calcolaMassimo() per tutti gli elementi confrontali con il numero desiderato se è maggiore memorizzalo visualizza il massimo
• **calcola** le medie per fasce	funzione calcolaMedia() per tutti gli elementi somma tutte le temperature calcola e memorizza la media
• **visualizza** i risultati	funzione mostraVettore()

Nel successivo affinamento dettagliamo solo due funzioni di calcolo:

III affinamento – pseudocodifica

```
void calcolaMassimo()
 int miaMat[GIORNI][ORE]
 int tempMAX, oraMAX, giornoMAX
 per y da 0 a ORE-1 fai
   per x da 0 a GIORNI-1 fai
     se(miaMat[x][y] > tempMAX)
       tempMAX ← miaMat[x][y]
       oraMAX ← y
       giornoMAX ← x
     fineSe
   finePer
 finePer
 stampa(tempMAX);
return void
```

```
void calcolaMedia()
  int medie[ORE], media
  per y da 0 a ORE-1 fai
    media ← 0
    per x da 0 a GIORNI-1 fai
      media ← media + miaMat[x][y]
    finePer
    medie[y] ← media/GIORNI
  finePer
return void
```

Riportiamo di seguito i codici di tutte le funzioni.

1. Funzione che riempie la matrice inserendo i dati da tastiera

```
static void riempiManuale(){
  Scanner in = new Scanner(System.in);
  int valore;                  // numero da cercare
  int x, y;
  for(y = 0; y < ORE; y++)
    for(x = 0; x < GIORNI ; x++){
      System.out.print("Inserisci la temperatura: ");
      valore =  in.nextInt();
      miaMat[x][y] = valore;
    }
}
```

2. Funzione che calcola la media delle temperature per ciascuna fascia

```
static void calcolaMedia(){
  int x, y, media;
  for (y = 0; y < ORE; y++){      // per tutte le fasce
    media = 0;
    for (x = 0; x < GIORNI; x++){ // sommo i giorni
      media = media + miaMat[x][y];
    }
    medie[y] = media / GIORNI;    // divido per i giorni
  }
}
```

3. Funzione che visualizza la matrice

```
static void mostraMatrice(){
  int x, y;
  for(y = 0; y < ORE; y++){
    System.out.println();
    System.out.print("giorno "+(y+1)+": ");
    for(x = 0; x < GIORNI ; x++)
      System.out.print(miaMat[x][y]+" ");
  }
  System.out.println();
}
```

4. Funzione che visualizza il vettore

```
public static void mostraVettore(){
 System.out.println( "temperatura media x fasce orarie");
 System.out.print("[ ");
 for(int x = 0; x < medie.length; x++)
   System.out.print(medie[x] + " ");
 System.out.println(" ]");
}
```

Per migliorare la comunicazione con l'utente è opportuno scrivere una funzione **decodFascia(int)**, che decodifica il valore dell'intervallo orario (1–5) nelle corrispondenti fasce ("11–12", "12–13", "13–14","14–15","15–16") e verrà richiamata dalla funzione che calcola il valore massimo.

5. Funzione che calcola il valore massimo di temperatura registrata

```
static void calcolaMassimo(){
  int x, y;
  tempMAX = 0;
  oraMAX = 0;
  giornoMAX = 0;
  for(y = 0; y < ORE; y++){
    for(x = 0; x < GIORNI; x++)
      if(miaMat[x][y] > tempMAX){
        tempMAX = miaMat[x][y];
        oraMAX = y;
        giornoMAX = x;
      }
  }
    System.out.print( "\ntemperat. massima: " + tempMAX);
  System.out.print ("\nfascia oraria     : ");
  decodFascia(oraMAX); //da y a fascia
}
```

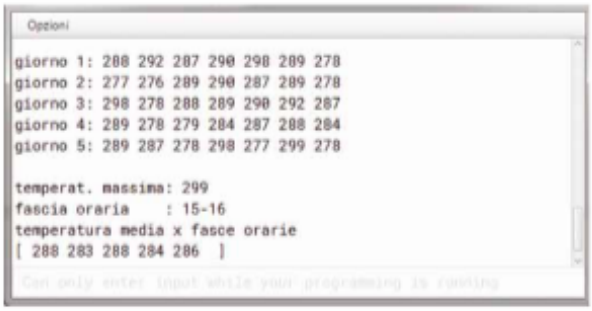
6 Programma principale che richiama le singole funzioni

```
public static void main(String[] args){
  int numero;
  miaMat = new int[GIORNI][ORE];  // matrice bidimension];
  medie = new int[ORE]; // crea il vettore
  riempiManuale();          // carica i dati nella matrice
  mostraMatrice();          // visualizza la matrice
  calcolaMassimo();         // carica temperatura massima
  calcolaMedia();           // calcola le medie per fasce
  mostraVettore();          // visualizza le medie
 }
}
```

L'esecuzione del programma

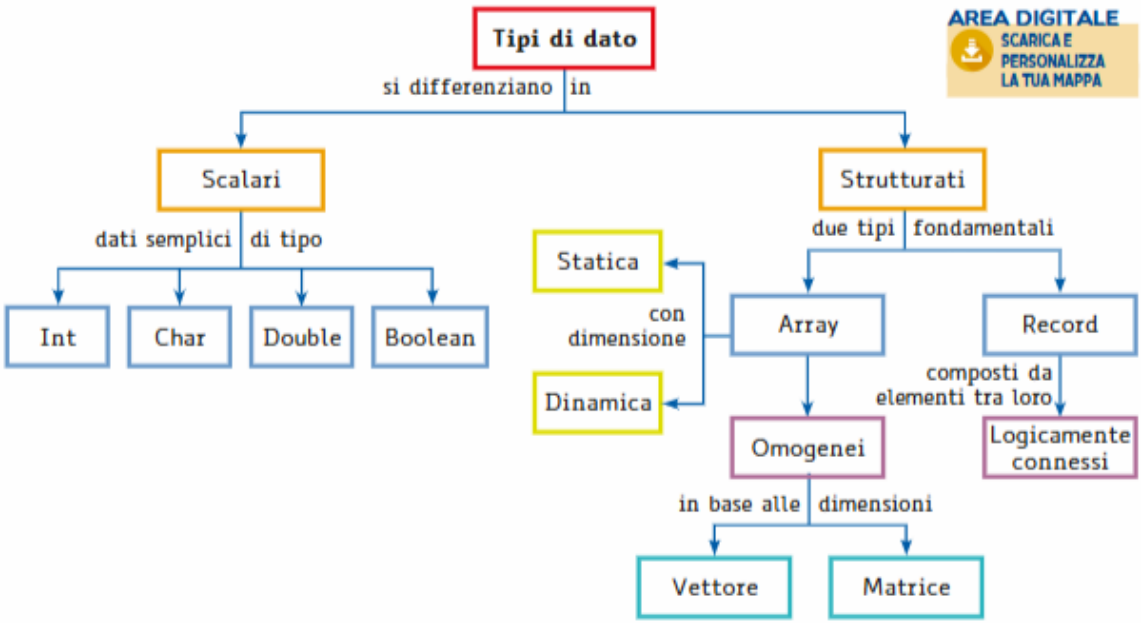
Riportiamo solo l'output dei risultati, senza la fase di inserimento dei dati; un primo miglioramento si ottiene inserendo i nomi dei giorni, come nel secondo output, che si potrebbe ulteriormente migliorare indicando sopra ogni colonna la fascia oraria di rilevazione e la conversione dei dati da gradi kelvin a gradi centigradi.

Il codice sorgente di questo programma lo trovi nel file **Temperature2.java**.



MAPPA CONCETTUALE

didattica inclusiva



Che cosa abbiamo imparato?

- L'array è uno strumento o, per meglio dire, un oggetto che permette di aggregare dati omogenei per poterli facilmente elaborare, cioè memorizzare, ritrovare e manipolare.
- Ogni posizione all'interno del vettore prende anche il nome di cella e viene indicizzata tramite la sua posizione, che in Java parte dal valore 0.
- Gli elementi che caratterizzano una matrice sono tre: tipo, dimensione, e identificatore
- L'utilizzo delle matrici all'interno dei metodi/funzioni può essere fatto solo se le funzioni possono avere accesso a tutti i componenti della matrice, quindi operare "per indirizzo" e non per valore, dato che risulta impensabile duplicare tutta la matrice. La modalità di utilizzo più semplice è sfruttare le regole di visibilità, cioè utilizzare direttamente la matrice definita all'interno del main().

Ora prova tu a rispondere

- Che cosa si intende per dati omogenei?
- Dov'è memorizzato un array?
- Come viene definito un array in Java?
- Quando è necessario dimensionare un array?
- Con quale tipo di iterazione generalmente si elaborano gli array? Perché?
- Che cosa si intende con indice di un array?
- Che indice ha il primo elemento dell'array?
- Che cos'è una cella di un vettore?
- Che cosa si intende per array a due dimensioni?
- Come si individua un elemento in una matrice?
- Quanti e quali sono gli elementi che caratterizzano una matrice?
- Quali sono le coordinate dell'elemento posto in alto a sinistra nella matrice?
- Quale tipo di iterazione viene utilizzato per elaborare le matrici?