

9 - Array

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
<http://pages.di.unipi.it/milazzo>
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2017/2018

Quante variabili.... (1)

Supponiamo di voler scrivere un programma che chiede all'utente di inserire 10 numeri, e poi li stampa nell'ordine in cui sono stati inseriti:

Esempio di esecuzione:

```
Inserisci 10 numeri interi:  
7  
6  
12  
667  
-33  
0  
13  
-34  
4  
100  
Risultato:  
7  
6  
12  
667  
-33  
0  
13  
-34  
4  
100
```

Quante variabili.... (2)

Una possibile soluzione:

```
import java.util.Scanner;

public class RipetoDieciNumeri {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        // variabili usate per memorizzare i 10 numeri
        int primo, secondo, terzo, quarto, quinto;
        int sesto, settimo, ottavo, nono, decimo;

        System.out.println("Inserisci 10 numeri interi");

        primo = input.nextInt();
        secondo = input.nextInt();
        ..... // omissis (per fare entrare il programma nella slide)
        nono = input.nextInt();
        decimo = input.nextInt();

        System.out.println("Risultato");
        System.out.println(primo);
        System.out.println(secondo);
        ..... // omissis (per fare entrare il programma nella slide)
        System.out.println(nono);
        System.out.println(decimo);
    }
}
```

Quante variabili.... (3)

Questa soluzione presenta diversi svantaggi:

- dieci variabili diverse
- dieci `nextInt()`
- dieci `println()`

Sarebbe più pratico usare un ciclo, ma come?

E se i numeri da inserire fossero stati 50?

Array

Soluzione: usare un **array**

Un **array** è:

- una **sequenza di variabili**
- di **tipo omogeneo** (tutti dello stesso tipo)
- distinguibili l'uno dall'altro (indirizzabili) in base alla loro posizione all'intero della sequenza (**indici interi**)

Variabili vs Array

10 variabili di tipo int:

primo

7

 secondo

6

 terzo

12

 quarto

667

Un array di tipo int di dimensione 10

numeri

7	6	12	667	-33	0	13	-34	4	100
0	1	2	3	4	5	6	7	8	9

Il numero di elementi dell'array è detto **lunghezza** (o **dimensione**)

Usare un array (1)

Dichiarazione di un array:

- Sintassi: `<tipo>[] <nomeArray>;`
- Esempi:

```
int[] numeri; // dichiara un array di interi
double[] prezzi; // dichiara un array di numeri frazionari
String[] nomi; // dichiara un array di stringhe
```

Creazione di un array:

- Sintassi: `<nomeArray> = new <tipo>[<lunghezza>]`
- Esempi:

```
numeri = new int[10]; // crea un array di 10 interi
prezzi = new double[25]; // crea un array di 25 numeri fraz.
nomi = new String[100]; // crea un array di 100 stringhe
```

Dichiarazione e creazione possono essere abbinati:

```
int[] numeri = new int[10];
```

Usare un array (2)

```
int [] numeri = new int [10];
```

Note sulla creazione di array:

- serve per **allocare** (riservare) l'area di memoria che dovrà contenere i valori
- la **lunghezza** dell'array deve essere specificata in fase di creazione e **non potrà cambiare**
- un array è in realtà un **oggetto**, e per questo si usa **new**

Usare un array (3)

Al momento della creazione, tutti gli elementi sono **inizializzati** a:

- 0 se il tipo è numerico (come `int[]`, `double[]` e `char[]`)
- `false` se il tipo è `boolean[]`
- `null` (non inizializzato) per array di oggetti (come `String[]`)

Esempio:

```
// crea un array di 5 elementi inizializzati a 0
int[] elementi = new int[5];
```

elementi	0	0	0	0	0
	0	1	2	3	4

Gli elementi di un array possono essere inizializzati in fase di dichiarazione

```
// crea un array di 5 elementi inizializzati a 10, 5, 21, -12 e 74
int[] elementi = { 10, 5, 21, -12, 74 };
```

elementi	10	5	21	-12	74
	0	1	2	3	4

Usare un array (4)

I **singoli elementi** di un array possono essere **acceduti** in lettura (in un'espressione) o in scrittura (in un assegnamento) indicandone la posizione (indice) tra **parentesi quadrate**

- **Ricordarsi** che il primo elemento ha indice 0!!!

```
numeri[0] = 7; // assegna 7 al primo elemento dell'array numeri
numeri[1] = 6; // assegna 6 al secondo elemento dell'array numeri
somma = numeri[0]+numeri[1] // somma i primi due elementi di numeri
x = numeri[i] // assegna il valore dell'i-esimo elemento a x
```

ATTENZIONE: se l'array ha dimensione n l'indice tra le parentesi deve essere compreso **tra 0 e $n-1$**

- in caso contrario si incontrerà un errore in fase di esecuzione

Usare un array (4)

La lunghezza di un array può essere scoperta tramite l'attributo `length`

```
int[] numeri = new int[10];  
System.out.println(numeri.length) //stampa 10
```

Attenzione: `length` non è un metodo (non è seguito da parentesi)

Questo è necessario in quanto **non sempre si conosce a priori** la dimensione di un array. Ad esempio:

```
....  
if (grande)  
    numeri = new int[1000];  
else  
    numeri = new int[10];  
.....  
// qual e' la lunghezza di numeri?
```

```
void stampaNomi(String[] nomi) {  
    // qual e' la lunghezza di nomi?  
}
```

Esempio

Una soluzione dell'esempio iniziale che usa un array:

```
import java.util.Scanner;

public class RipetoDieciNumeri2 {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        // array usato per memorizzare i 10 numeri
        int numeri[] = new int[10];

        System.out.println("Inserisci 10 numeri interi");
        for (int i=0; i<10; i++) {
            numeri[i] = input.nextInt();
        }

        System.out.println("Risultato");
        for (int i=0; i<10; i++) {
            System.out.println(numeri[i]);
        }
    }
}
```

Il comando for-each (1)

Nell'ambito degli array esiste una versione "semplificata" del comando for

```
for ( String s : nomi ) {  
    System.out.println(s); // stampa tutti gli elementi di nomi  
}
```

Questa versione del for viene chiamato **for-each**

- Nell'esempio: **per ogni** (for each) stringa s nell'array nomi stampa s
- Gli elementi vengono presi **in ordine** di posizione nell'array
- La variabile (locale) usata nel for-each
 - ▶ deve essere di **tipo** compatibile con il tipo dell'array
 - ▶ consente di usare gli elementi dell'array in **sola lettura** (contiene una copia dell'elemento corrente: modifiche alla variabile non hanno effetti sugli elementi dell'array)

L'esempio sopra può essere riscritto con un for tradizionale come segue:

```
for ( int i=0; i<nomi.length; i++) {  
    String s = nomi[i];  
    System.out.println(s);  
}
```

Il comando for-each (2)

Modifichiamo l'esempio iniziale usando un for-each

- Perché non due?

```
import java.util.Scanner;

public class RipetoDieciNumeri3 {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        // array usato per memorizzare i 10 numeri
        int numeri[] = new int[10];

        System.out.println("Inserisci 10 numeri interi");
        for (int i=0; i<10; i++) {
            numeri[i] = input.nextInt();
        }

        System.out.println("Risultato");
        for (int n : numeri) {
            System.out.println(n);
        }
    }
}
```

Il comando for-each (3)

Il for-each può essere usato solo quando l'indice degli elementi non è importante

```
for (int n : numeri) {  
    somma += n;    //somma tutti gli elementi dell'array  
}
```

Quando invece l'indice degli elementi è importante, bisogna usare il for tradizionale

```
for (int i=0; i<numeri.length; i++) {  
    // stampa le posizioni di tutti gli zeri nell'array  
    if (numeri[i]==0) System.out.println(i);  
}
```

Esempi (1)

Inizializzare un array di stringhe con i seguenti valori:

"gatto", "cane", "topo", "criceto", "pesce".

Stampare la stringa più lunga tra quelle presenti nell'array.

Prima soluzione

```
public class StringaPiuLunga {
    public static void main(String[] args) {

        String[] animali = {"gatto","cane","topo","criceto","pesce"};

        String piuLunga = ""; // memorizza la stringa
        for (String s : animali)
            if (s.length()>piuLunga.length()) piuLunga=s;
        }
        System.out.println("La piu' lunga e': " + piuLunga);
    }
}
```


Esempi (2)

Seconda soluzione

```
public class StringaPiuLunga2 {
    public static void main(String[] args) {

        String[] animali = {"gatto", "cane", "topo", "criceto", "pesce"};

        int posPiuLunga = 0; // memorizza la posizione
        for (int i=1; i<animali.length; i++) { // possiamo saltare 0!!
            if (animali[i].length()>animali[posPiuLunga].length())
                posPiuLunga=i;
        }
        System.out.println("La piu' lunga e': " + animali[posPiuLunga]);
    }
}
```

Esempi (3)

Gli **oggetti della classe Random** del pacchetto `java.util` prevedono numerosi metodi per generare numeri casuali. Tra questi, il metodo `nextInt(num)` genera un numero a caso tra 0 (compreso) e `num` (escluso).

Realizzare una classe `ArrayCasuale` che prevede due metodi ausiliari:

- `generaArray(n)` che restituisce un nuovo array di `n` elementi inizializzato con numeri casuali tra 0 e 100
- `sommaTutti(a)` che restituisce la somma di tutti gli elementi dell'array `a`

Il metodo `main` deve utilizzare i metodi ausiliari per generare un array di 10 elementi dei quali stampare la somma.

Esempi (4)

```
import java.util.Random; //importa la classe di libreria

public class ArrayCasuale {

    public static void main(String[] args) {
        int[] numeri = generaArray(10);
        System.out.println(sommaTutti(numeri));
        // alternativa: System.out.println(sommaTutti(generaArray(10)));
    }

    private static int[] generaArray(int n) {
        int[] ris = new int[n]; //crea l'array da restituire
        Random rng = new Random(); //oggetto "random number generator"

        for (int i=0; i<n; i++)
            ris[i] = rng.nextInt(100); //genera gli elementi

        return ris;
    }

    private static int sommaTutti(int[] a) {
        int somma = 0;
        for (int x : a) somma += x;
        return somma;
    }
}
```