

# Appunti di Progettazione del Software

Canale 2 · Prof. Massimo Mecella

## Capitolo 5: Metodi statici e array

**Autore: Tommaso C.**

---

Appunti tratti dalle lezioni in aula e dal materiale del corso

---

Argomenti: Metodi statici · Overloading · Passaggio dei parametri  
Variabili statiche e costanti · Array · Matrici

## Indice

|          |                                                   |           |
|----------|---------------------------------------------------|-----------|
| <b>1</b> | <b>Metodi statici</b>                             | <b>1</b>  |
| <b>2</b> | <b>Definire un metodo statico</b>                 | <b>2</b>  |
| <b>3</b> | <b>Overloading</b>                                | <b>2</b>  |
| <b>4</b> | <b>Variabili locali e passaggio dei parametri</b> | <b>3</b>  |
| <b>5</b> | <b>Variabili statiche e moduli funzionali</b>     | <b>4</b>  |
| <b>6</b> | <b>Array</b>                                      | <b>5</b>  |
| <b>7</b> | <b>Matrici</b>                                    | <b>9</b>  |
| <b>8</b> | <b>Esercizi</b>                                   | <b>11</b> |
| <b>9</b> | <b>Soluzioni</b>                                  | <b>12</b> |

## 1 Metodi statici

### Definizione – Metodo statico

Un **metodo statico** (**static**) si invoca sulla classe che lo definisce, senza oggetto di invocazione:

`NomeClasse.nomeMetodo(parametri attuali)`

È l'equivalente di una funzione o di una procedura del C.

Un metodo invocato su un oggetto, come `s1.concat(s2)`, riceve come parametro implicito l'oggetto a sinistra del punto. Un metodo statico non ha questo parametro: nel suo record di attivazione non c'è un riferimento all'oggetto su cui lavora.

**Quando un metodo è statico.** Invocare un metodo su un oggetto fa pensare a un effetto su quell'oggetto. Il coseno di un numero invece non cambia il numero: restituisce un altro numero, e non c'è un oggetto naturale a cui chiederlo. Le funzioni di questo tipo sono statiche, come i metodi della classe `Math` di `java.lang`:

```
double x = Math.cos(alpha);  
double t = Math.cos(alpha) + Math.sqrt(delta);    // sqrt: radice quadrata
```

### Definizione – Cast

Il **cast** converte esplicitamente un valore in un altro tipo, compatibile:

`(tipo) espressione`

Da `double` a `int` il cast **tronca** la parte decimale: `(int) 5.99` vale 5, `(int) -2.7` vale -2.

Un `double` non si assegna a una variabile `int` senza cast: il compilatore rifiuta la conversione, perché perderebbe la parte decimale.

**Numeri casuali.** `Math.random()` restituisce un `double` casuale nell'intervallo  $[0, 1)$ , con lo 0 compreso e l'1 escluso. Per un intero fra 1 e 6, come il lancio di un dado:

`int x = (int) (Math.random() * 6 + 1);`

`Math.random() * 6 + 1` sta in  $[1, 7)$ , e il cast lo tronca a uno fra  $1, \dots, 6$ . Le parentesi intorno al prodotto servono: `(int) Math.random() * 6 + 1` tronca prima `Math.random()`, che diventa sempre 0, e vale sempre 1 (Esercizio 1).

## 2 Definire un metodo statico

### Definizione – Definizione di un metodo statico

```
public static tipoRisultato nomeMetodo(parametri) { ... }
```

- **tipoRisultato** è un tipo primitivo, una classe oppure **void**;
- **return espressione;** termina il metodo e restituisce il valore dell'espressione.

```
public static int somma(int a, int b) {  
    return a + b;  
}
```

**main** è un metodo statico: la JVM lo invoca senza oggetto (Capitolo 3).

Un metodo che restituisce un valore deve terminare con un **return** in ogni caso possibile; altrimenti il compilatore segnala *missing return statement*. Un metodo **void** non ha bisogno di **return**.

Il valore restituito sta nel record di attivazione del metodo, che si toglie quando il metodo termina: se il chiamante non lo usa o non lo assegna a una variabile, come in `int t = somma(2, 3);`, va perso.

## 3 Overloading

### Definizione – Overloading

L'**overloading** (sovraccarico) dà lo stesso nome a più metodi della stessa classe, che devono differire per numero, tipo o ordine dei parametri, cioè per la segnatura. Il tipo restituito non basta a distinguerli. Il compilatore sceglie il metodo da invocare in base ai parametri attuali.

```
public class ProvaMetodi {  
    public static String generaSaluto() {  
        return "Buongiorno";  
    }  
    public static void stampaSaluto() {  
        System.out.println(generaSaluto());  
    }  
    public static void stampaSaluto(String nome) {  
        System.out.println(generaSaluto() + " " + nome);  
    }  
    public static void stampaSaluto(String nome1, String nome2) {  
        System.out.println(generaSaluto() + " " + nome1 + " e " + nome2);  
    }  
}
```

```
}  
}
```

Le tre versioni di `stampaSaluto` si distinguono per il numero di parametri:

- `stampaSaluto()` stampa **Buongiorno**;
- `stampaSaluto("Mario")` stampa **Buongiorno Mario**;
- `stampaSaluto("Mario", "Anna")` stampa **Buongiorno Mario e Anna**.

Dentro la classe i metodi statici si invocano anche senza il nome della classe.

Le varianti di uno stesso servizio hanno così lo stesso nome, invece di nomi artificiosi come `stampaSaluto1` e `stampaSaluto2`, che in C sono necessari: il codice è più leggibile e la documentazione mostra subito la famiglia di metodi.

```
public static void f() { }  
public static int f() { return 0; } // errore: f() esiste già
```

Le due `f` hanno la stessa segnatura, e il compilatore segnala *method f() is already defined*.

## 4 Variabili locali e passaggio dei parametri

Una variabile dichiarata in un metodo, statico o no, è visibile solo in quel metodo, e la sua cella sta nel record di attivazione del metodo, nello stack. Anche i parametri formali sono variabili locali: quando il metodo è invocato, ricevono una copia dei parametri attuali.

### Definizione – Passaggio per valore

In Java i parametri si passano sempre **per valore**: il parametro formale riceve una copia del valore del parametro attuale.

- Per un tipo primitivo si copia il valore: il metodo non può cambiare la variabile del chiamante.
- Per un oggetto si copia il riferimento: il metodo lavora sullo stesso oggetto del chiamante, e può modificarlo con un side effect.

Il side effect su un parametro si vede con gli array, nella Sezione 6.

**Ricorsione.** Ogni invocazione ha il proprio record di attivazione, quindi un metodo può invocare se stesso, come in C:

```
public static int fattoriale(int n) {  
    if (n == 0)  
        return 1;  
    else
```

```
    return n * fattoriale(n - 1);  
}
```

`fattoriale(5)` vale **120**: lo stack contiene sei record di `fattoriale`, per  $n$  da 5 a 0, che si tolgono uno dopo l'altro restituendo i risultati.

## 5 Variabili statiche e moduli funzionali

### Definizione – Variabile statica

Una **variabile statica** si dichiara nel corpo della classe, fuori dai metodi:

```
public static tipo nomeVariabile;
```

Esiste in una sola copia, indipendente dai metodi e dagli oggetti, per tutta l'esecuzione. Da un'altra classe si usa con `NomeClasse.nomeVariabile`.

### Definizione – Costante

Una **costante** è una variabile statica `final`, che non si può modificare:

```
public static final int ZERO = 0;
```

Riceve il valore nella dichiarazione; un'assegnazione successiva è un errore di compilazione.

`Math.PI` è la costante  $\pi$  della classe `Math`. Il nome della classe, con l'iniziale maiuscola per convenzione, dice subito a chi legge che si tratta di un membro statico.

Per convenzione le variabili statiche si dichiarano all'inizio della classe, prima dei metodi, così chi legge le trova subito; il compilatore però le accetta in qualunque punto del corpo della classe.

### Osservazione – Variabili statiche modificabili

Una variabile `public static` non `final` si può leggere e modificare da qualsiasi metodo di qualsiasi classe: è una **variabile globale**, con i difetti visti nel Capitolo 2. Si usa solo in casi rari.

### Definizione – Modulo funzionale

Un **modulo funzionale** è una classe che non serve a creare oggetti, ma raccoglie metodi, variabili e costanti statici, come `Math`. Come ogni modulo deve avere alta coesione, basso accoppiamento, interfacciamento esplicito e information hiding.

In un modulo funzionale ogni metodo riceve ciò che gli serve come parametro, e non attraverso variabili globali: così dalla sua intestazione si vede esattamente che cosa usa.

## 6 Array

### Definizione – Array

Un **array** è una sequenza di elementi dello stesso tipo, ciascuno individuato da un indice intero. In Java un array è un **oggetto**, nello heap anche quando gli elementi sono di un tipo primitivo; una variabile di tipo array contiene un riferimento.

|                        |                                                                   |
|------------------------|-------------------------------------------------------------------|
| dichiarazione          | <code>tipo[] nomeArray;</code>                                    |
| creazione              | <code>nomeArray = new tipo[dimensione];</code>                    |
| lunghezza              | <code>nomeArray.length</code>                                     |
| elemento di indice $i$ | <code>nomeArray[i]</code> , con $0 \leq i \leq \text{length} - 1$ |

```
String[] names = new String[100]; // dichiarazione e creazione insieme
names[0] = "Rossi";               // primo elemento
names[99] = "Verdi";              // ultimo elemento
```

L'accesso a un indice fuori da  $0, \dots, \text{length} - 1$ , come `names[100]` o `names[-1]`, è un errore di esecuzione: `ArrayIndexOutOfBoundsException`.

### Nota – `int[] a` e `int a[]`

Java accetta anche la forma del C, `int numbers[]`, con le parentesi sul nome. Compila, ma la forma usata in Java è `int[] numbers`; il tipo è «array di `int`».

**Scorrere un array.** Un ciclo `for` con l'indice da `0` a `length - 1` visita tutti gli elementi. La somma dei quadrati degli interi da `0` a `99`:

```
int[] v = new int[100];
for (int i = 0; i < v.length; i++)
    v[i] = i * i;
int somma = 0;
for (int i = 0; i < v.length; i++)
    somma += v[i]; // somma = somma + v[i]
```

Alla fine `somma` vale `328350`. Con la condizione `i <= v.length` il ciclo arriva a `v[100]`, e il programma si ferma con `ArrayIndexOutOfBoundsException: Index 100 out of bounds for length 100`.

### Proposizione – Valori predefiniti

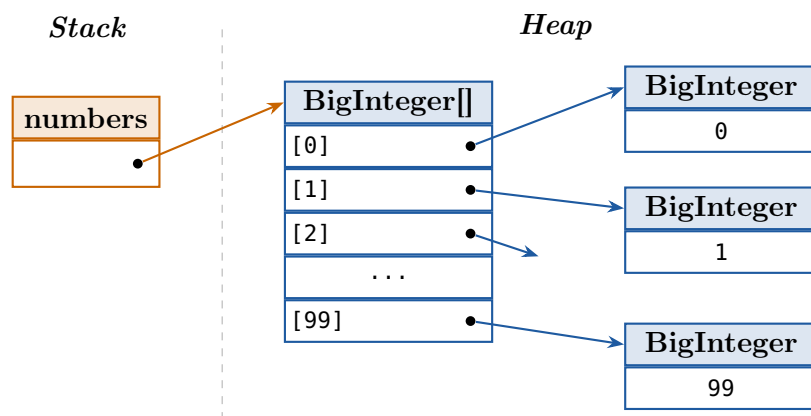
Gli elementi di un array appena creato con **new** hanno un valore predefinito, secondo il tipo:

| Tipo degli elementi      | Valore iniziale                |
|--------------------------|--------------------------------|
| int (e gli altri interi) | 0                              |
| float, double            | 0.0                            |
| char                     | '\0', il carattere di codice 0 |
| boolean                  | false                          |
| una classe               | null                           |

**Array di oggetti.** In un array di oggetti, **new** crea solo le celle, tutte **null**: gli oggetti si creano a parte. **BigInteger**, del pacchetto **java.math**, rappresenta interi di grandezza illimitata:

```
BigInteger[] numbers = new BigInteger[100]; // 100 celle null
for (int i = 0; i < numbers.length; i++)
    numbers[i] = BigInteger.valueOf(i); // 100 oggetti BigInteger
```

**BigInteger.valueOf(i)** è un metodo statico che restituisce un **BigInteger** con il valore dell'int **i**.



*Dopo il ciclo: nello stack c'è solo il riferimento **numbers**; l'array e i cento **BigInteger** sono nello heap. Prima del ciclo tutte le celle valevano **null**.*

Se l'unico riferimento all'array è una variabile locale, alla fine del metodo l'array e i suoi oggetti diventano irraggiungibili, e il garbage collector li recupera, con tutto il lavoro fatto per riempirli. Un oggetto che serve ancora va tenuto raggiungibile.

**Inizializzazione con un elenco.** Quando i valori sono noti mentre si scrive il programma, l'array si crea elencandoli fra graffe:

```
int[] primes = {2, 3, 5, 7, 11, 13};
primes = new int[] {17, 19, 23, 29, 31};    // nuovo array, assegnato a primes
```

Anche senza `new` esplicito l'array è un oggetto nello heap, lungo quanto l'elenco. La seconda forma, con `new int[]`, si usa in un'assegnazione o come parametro attuale; il primo array, se nessun'altra variabile lo denota, resta al garbage collector.

**Array vuoti e null.** Un array di lunghezza 0, `new int[0]` o `new int[] {}`, esiste ma non ha elementi: `a.length` vale 0. Serve, per esempio, a un metodo che restituisce un array e non ha trovato risultati. Una variabile array che vale `null` invece non denota nessun array: `a.length` dà `NullPointerException`.

#### Definizione – Ciclo for migliorato

`for (tipo elemento : nomeArray) blocco`

esegue il blocco una volta per ogni elemento dell'array, nell'ordine, con `elemento` uguale a `nomeArray[0]`, poi a `nomeArray[1]`, e così via.

```
int sum = 0;
for (int n : numbers)    // "per ogni n in numbers"
    sum += n;
```

Il ciclo migliorato serve solo a leggere: `elemento` è una variabile locale che riceve una copia di ogni elemento, e assegnarle un valore non cambia l'array. Per scrivere negli elementi serve il ciclo con l'indice.

#### Proposizione – Copiare un array

- `int[] b = a;` copia il riferimento: `a` e `b` denotano lo stesso array, e una modifica fatta attraverso uno si vede dall'altro.
- `int[] b = Arrays.copyOf(a, a.length);` crea un array nuovo con gli stessi elementi: le modifiche a uno non toccano l'altro.

```
int[] primes = {2, 3, 5, 7, 11, 13};
int[] numbers = primes;
numbers[5] = 42;           // ora anche primes[5] vale 42
int[] copiedPrimes = Arrays.copyOf(primes, primes.length);
copiedPrimes[0] = 99;      // primes[0] resta 2
```

Per un array di oggetti, `Arrays.copyOf` copia i riferimenti: i due array sono distinti, ma i loro elementi denotano gli stessi oggetti.

La classe `Arrays` di `java.util` è un modulo funzionale con altri metodi utili:

- `Arrays.sort(a)` ordina l'array;
- `Arrays.toString(a)` ne restituisce una rappresentazione come stringa.

Per l'array `{"Rossi", "Bianchi", "Verdi"}`, dopo l'ordinamento si ottiene `[Bianchi, Rossi, Verdi]`.

**Array come parametri.** Un array passato a un metodo è passato per riferimento all'oggetto (Sezione 4): il metodo può modificarne gli elementi, e il chiamante vede la modifica.

```
public static void f(int[] x) {
    x[0] = 1;
}
```

Dopo `int[] a = new int[3]; f(a);` l'elemento `a[0]` vale 1.

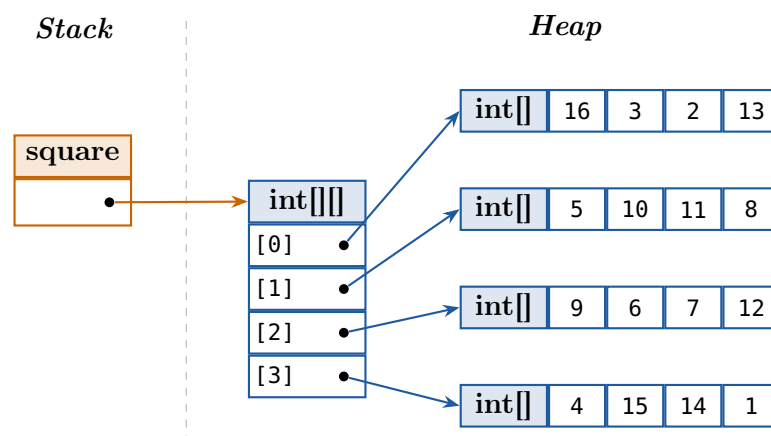
## 7 Matrici

### Definizione – Matrice

Una **matrice** è un array bidimensionale, cioè un array di array:

|                                  |                                                                           |
|----------------------------------|---------------------------------------------------------------------------|
| dichiarazione                    | <code>tipo[][] nomeMatrice;</code>                                        |
| creazione                        | <code>nomeMatrice = new tipo[n][m];</code><br>con $n$ righe e $m$ colonne |
| elemento                         | <code>nomeMatrice[i][j]</code> , riga $i$ , colonna $j$                   |
| numero di righe                  | <code>nomeMatrice.length</code>                                           |
| numero di colonne della riga $i$ | <code>nomeMatrice[i].length</code>                                        |

```
int[][] square = {
    {16, 3, 2, 13},
    { 5, 10, 11, 8},
    { 9, 6, 7, 12},
    { 4, 15, 14, 1}
};
int element = square[1][2]; // 11
```



La matrice `square`: un array di quattro righe, ciascuna un array di quattro `int`.

**Scansione.** Due cicli annidati, uno sulle righe e uno sulle colonne. La somma di due matrici della stessa dimensione:

```
public static double[][] sommaMatrici(double[][] A, double[][] B) {
    double[][] C = new double[A.length][A[0].length];
    for (int i = 0; i < A.length; i++)
        for (int j = 0; j < A[0].length; j++)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}
```

**Righe di lunghezza diversa.** Poiché ogni riga è un array a sé, le righe possono avere lunghezze diverse: si crea l'array delle righe con `new int[n][]` e poi ciascuna riga (Esercizio 3). In questo caso la scansione usa `A[i].length` per la riga `i`, oppure due cicli migliorati:

```
for (int[] row : triangle) {
    for (int element : row)
        System.out.print(element + " ");
    System.out.println();
}
```

`Arrays.deepToString(m)` restituisce la rappresentazione di una matrice a tutti i livelli, come `[[16, 3, 2, 13], [5, 10, 11, 8], ...]`; `Arrays.toString(m)` mostrerebbe solo i riferimenti delle righe.

### Riepilogo

- **Metodo statico:** `NomeClasse.metodo(parametri)`, senza oggetto di invocazione; `return` in ogni percorso se non è `void`.
- **Cast:** `(int) x` tronca un `double`; `(int) (Math.random() * k) + 1` è un intero fra 1 e  $k$ .
- **Overloading:** stesso nome, parametri diversi; il solo tipo restituito non basta.
- **Parametri:** sempre per valore; per un oggetto si copia il riferimento, e il metodo può modificare l'oggetto.
- **Statico:** variabili e costanti `static final` della classe; un modulo funzionale raccoglie solo membri statici.
- **Array:** oggetto di lunghezza fissa, `new tipo[n]`, indici da 0 a `length - 1`, elementi ai valori predefiniti.
- **Copia:** `b = a` condivide l'array, `Arrays.copyOf` ne crea uno nuovo.
- **Matrice:** array di array; `m.length` righe, `m[i].length` colonne della riga `i`.

## 8 Esercizi

### Esercizio 1 – La carta casuale

Le carte di un seme vanno dall'asso (1) al re (13). Scrivere un'istruzione che estrae a caso il valore di una carta, un intero fra 1 e 13 con la stessa probabilità. Che cosa vale invece `(int) Math.random() * 13 + 1`?

*Soluzione a pagina 12 →*

### Esercizio 2 – Massimo comun divisore

Scrivere un metodo statico ricorsivo `mcd(int x, int y)` che calcola il massimo comun divisore di due interi positivi  $x \geq y$  con il metodo di Euclide. Detto  $r$  il resto della divisione intera di  $x$  per  $y$ :

- se  $r = 0$ , cioè se  $x$  è multiplo di  $y$ ,  $\text{mcd}(x, y) = y$ ;
- altrimenti  $\text{mcd}(x, y) = \text{mcd}(y, r)$ .

Quanto vale `mcd(48, 18)`?

*Soluzione a pagina 12 →*

### Esercizio 3 – Il triangolo di Tartaglia

Nel triangolo di Tartaglia la riga  $i$  (da 0) ha  $i + 1$  elementi: il primo e l'ultimo valgono 1, e ogni altro elemento è la somma dei due che gli stanno sopra nella riga precedente, nella stessa colonna e in quella prima. Memorizzare le prime  $n = 5$  righe in una matrice `int[][]` con righe di lunghezza diversa, e stamparle una per riga.

*Soluzione a pagina 12 →*

## 9 Soluzioni

### Soluzione dell'Esercizio 1 – La carta casuale

```
int carta = (int) (Math.random() * 13) + 1;
```

`Math.random()` sta in  $[0, 1)$ , quindi `Math.random() * 13` sta in  $[0, 13)$ ; il cast tronca a un intero fra 0 e 12, ciascuno su un intervallo di ampiezza 1, e il `+1` porta a  $1, \dots, 13$ . Anche `(int) (Math.random() * 13 + 1)` va bene.

Senza le parentesi il cast si applica a `Math.random()`, che è minore di 1 e diventa 0: l'espressione vale sempre  $0 \cdot 13 + 1 = 1$ .

*← Torna all'Esercizio 1, pagina 11*

### Soluzione dell'Esercizio 2 – Massimo comun divisore

Il resto si calcola con `%`; i due casi della regola diventano i due rami di un `if`.

```
public static int mcd(int x, int y) {
    int r = x % y;
    if (r == 0)
        return y;
    else
        return mcd(y, r);
}
```

`mcd(48, 18)`:  $48 = 2 \cdot 18 + 12$ , quindi si invoca `mcd(18, 12)`;  $18 = 1 \cdot 12 + 6$ , quindi `mcd(12, 6)`;  $12 = 2 \cdot 6$ , resto 0, e il risultato è 6. La ricorsione termina perché il resto è sempre minore del divisore, e il secondo parametro diminuisce a ogni invocazione.

*← Torna all'Esercizio 2, pagina 11*

### Soluzione dell'Esercizio 3 – Il triangolo di Tartaglia

Si crea l'array delle  $n$  righe senza indicare le colonne, poi ogni riga con la sua lunghezza  $i + 1$ . L'elemento di colonna  $j$ , per  $0 < j < i$ , è `triangolo[i - 1][j - 1] + triangolo[i - 1][j]`.

```
int n = 5;
int[][] triangolo = new int[n][];
for (int i = 0; i < n; i++) {
    triangolo[i] = new int[i + 1];
    triangolo[i][0] = 1;
    triangolo[i][i] = 1;
    for (int j = 1; j < i; j++)
```

```
        triangolo[i][j] = triangolo[i - 1][j - 1] + triangolo[i - 1][j];
    }
    for (int[] riga : triangolo) {
        for (int elemento : riga)
            System.out.print(elemento + " ");
        System.out.println();
    }
```

Il programma stampa le righe 1, 1 1, 1 2 1, 1 3 3 1 e 1 4 6 4 1. Per la riga 0 le due assegnazioni scrivono lo stesso elemento, e il ciclo interno non viene eseguito per le righe 0 e 1.

*[← Torna all'Esercizio 3, pagina 11](#)*