

SUPSI

# Arrays

Loris Grossi, Fabio Landoni, Andrea Baldassari

Contenuto realizzato in collaborazione con: T. Leidi, A.E. Rizzoli, S. Pedrazzini

Fondamenti di Informatica

Bachelor in Ingegneria Informatica

# Obiettivo

Essere in grado di utilizzare, in maniera corretta, gli arrays all'interno di programmi Java.

Obiettivi della lezione:

- Conoscere le differenze tra tipi di dato strutturato e non strutturato.
- Conoscere la gestione della memoria per i tipi primitivi e per i tipi riferimento.
- Conoscere l'operatore `new` per l'allocazione della memoria.
- Conoscere e saper usare gli array mono e multi dimensionali.
- Conoscere e saper usare l'istruzione `for` avanzato (`foreach`).
- Saper effettuare una copia di un array (superficiale o profonda).
- Conoscere e saper utilizzare le varie utilities per gli array messe a disposizione dalla classe `Arrays`.

## Variabili

Prima di poter essere utilizzata, una variabile deve essere dichiarata. Ogni dichiarazione di variabile, in Java, è composta da un **tipo di dato** e da un **identificatore** (nome).

Quando viene dichiarata, la variabile può anche essere inizializzata.

La variabile è accessibile, tramite l'identificatore, dal momento in cui viene dichiarata e fino alla fine del blocco in cui essa è contenuta.

```
int abc; // non inizializzata
int x = 20 + 30;

int y = 0, z;
y = x * 4;
z = x * 8;

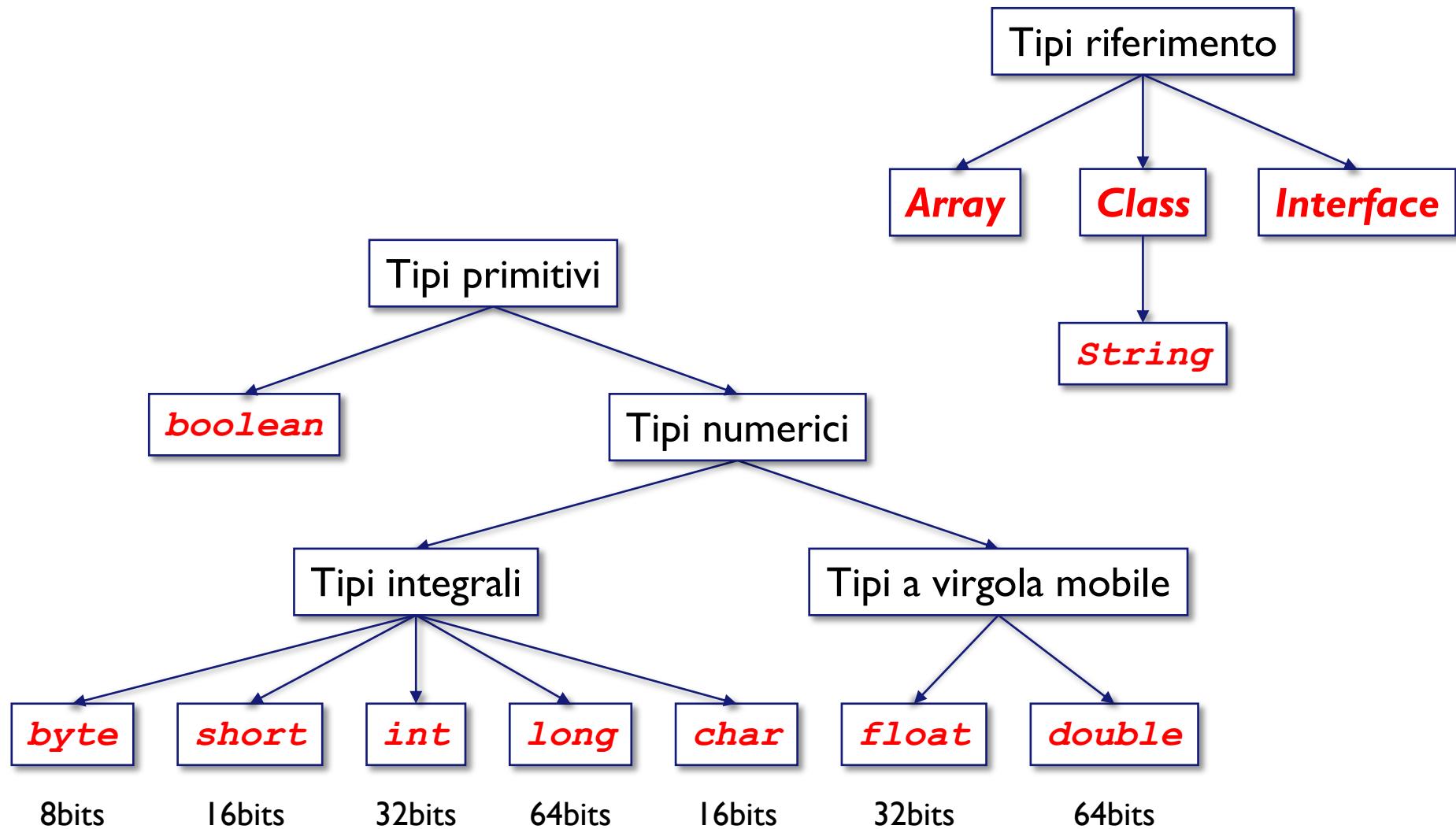
double d = z / (y * 1.75);
String mioNome = "Pippo";
```

## Esempio variabili

```
public class UsoDiVariabili {  
    public static void main(String[] args) {  
        int valore = 0;  
        System.out.println("Valore iniziale: " + valore);  
  
        valore = valore + 1;  
        System.out.println("Valore incrementato: " + valore);  
  
        String str1 = "Buongiorno";  
        String str2 = "studenti";  
  
        double altroValore = valore + 5.316;  
  
        System.out.println(str1 + " " + str2 + " il valore "  
                           + "finale è: " + altroValore);  
    }  
}
```

Ripasso

# Tipi di dato



## Tipi riferimento

Gli oggetti fanno parte dei tipi riferimento.

Arrays:

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

Classi:

```
Scanner input;  
input = new Scanner(System.in);
```

In Java non esiste l'equivalente della struct (come in C) o del record (come in Ada). Sono stati sostituiti dalla classe.

I tipi riferimento verranno approfonditi nel corso del semestre.

## Tipi di dato

Una maniera comune per catalogare i tipi di dato è quella di dividerli in:

- ***Tipi di dato non strutturati***: ogni identificatore è riferito ad un ***elemento unico e distinto***.

Esempio: int, float, char, long, double, ...

- ***Tipi di dato strutturati***: mediante un unico identificatore è possibile accedere ad ***elementi aggregati***.

Esempio: arrays, classi.

## Gestione della memoria: i tipi primitivi

Le variabili di **tipo primitivo** sono **tipi di dato non strutturato**.

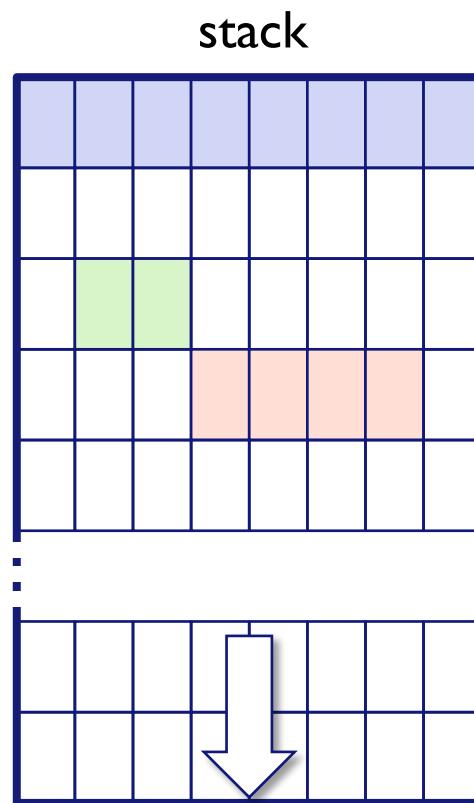
Dichiarando una variabile di tipo primitivo, la **memoria** di quella variabile verrà **dimensionata a seconda del tipo** specificato.

Lo **spazio occupato** dalla variabile viene **allocato** in una zona della **memoria** del programma detta **stack**.

Il contenuto della variabile si trova nell'allocazione stessa della variabile.

# Gestione della memoria: i tipi primitivi

```
double d = 0.0;  
// ...  
short a = 15;  
// ...  
int b = a;
```



d è double (**8 bytes**)

a è short (**2 bytes**)

b è int (**4 bytes**)

## Gestione della memoria: i tipi riferimento

I **tipi riferimento** sono **tipi di dato strutturati**.

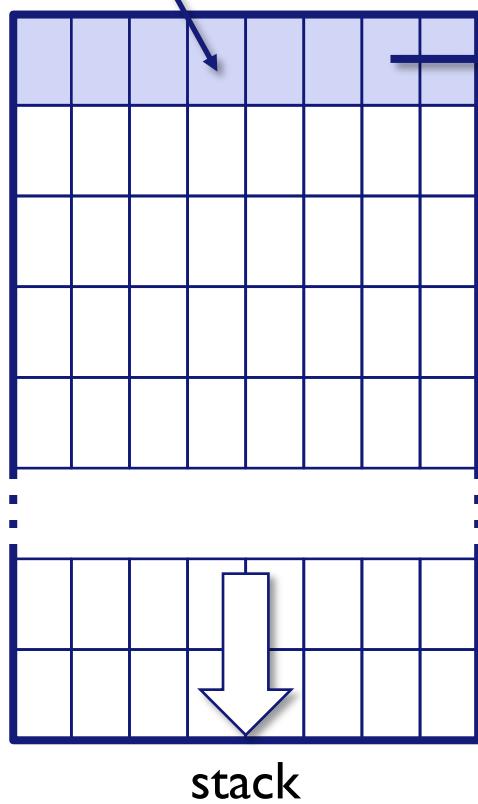
Lo **spazio occupato dai valori** delle variabili di tipo riferimento **non viene allocato nello stack**.

Nello **stack** viene salvato un **riferimento all'indirizzo di memoria** dove effettivamente si trovano i dati.

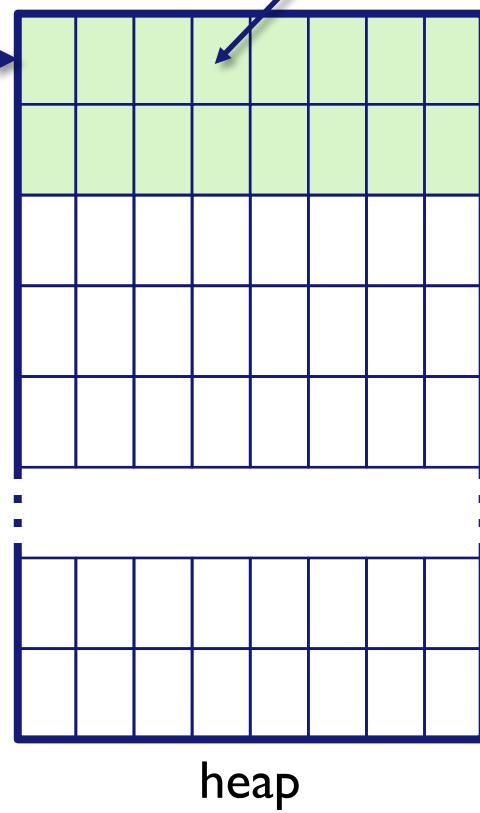
Lo spazio utilizzato per immagazzinare i **dati** viene allocato in una zona della **memoria** del programma detta **heap**.

# Gestione della memoria: i tipi riferimento

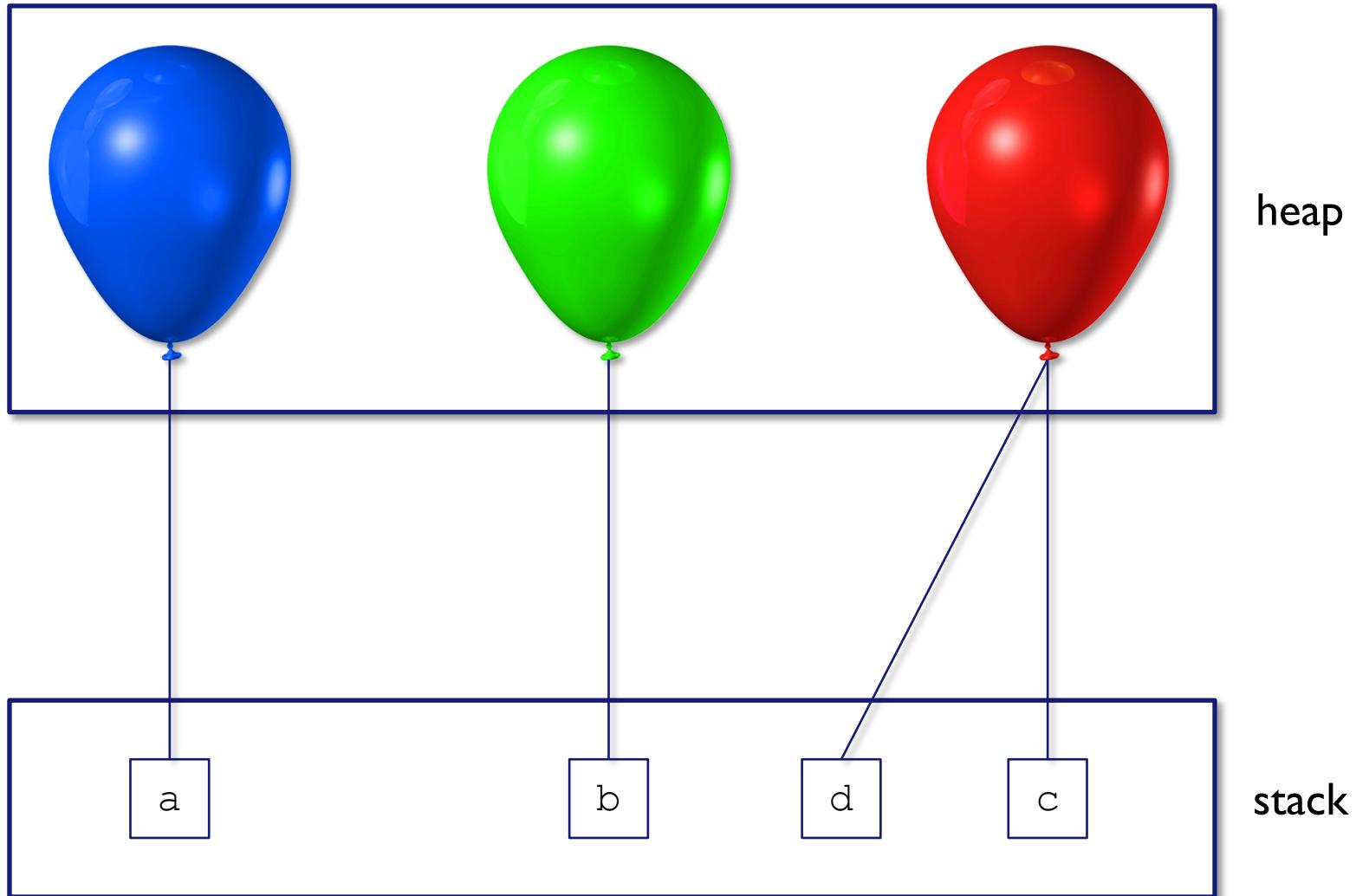
Indirizzo: riferimento (8 bytes)



Contenuto della variabile

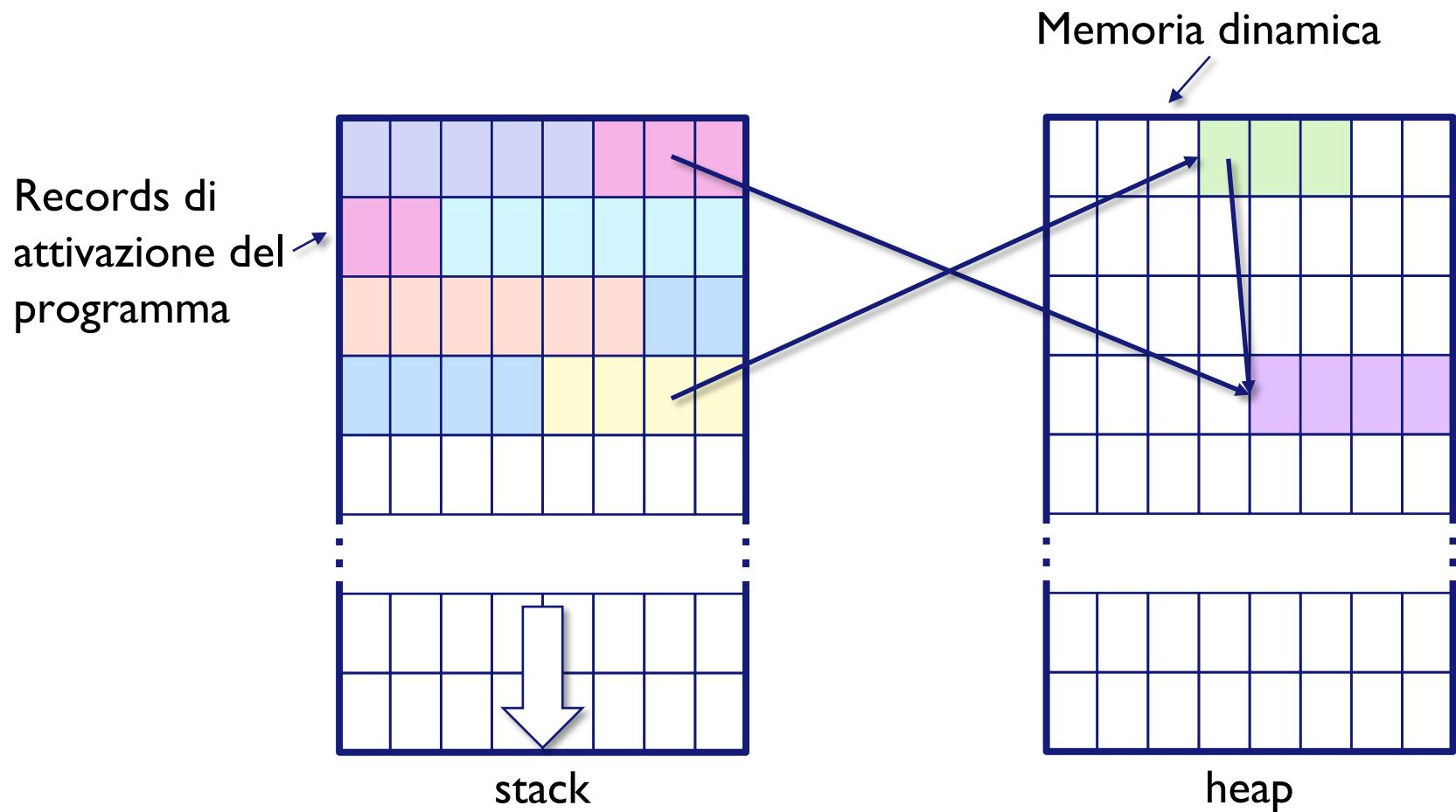


## I tipi riferimento: un'analogia



# Gestione dinamica della memoria

Gestione dinamica della memoria: operatore *new* e *garbage collector*.



## Allocazione e rilascio della memoria

La memoria dinamica di un programma (heap) va **allocata esplicitamente** all'interno del programma.

In Java, per l'allocazione si usa l'operatore **`new`**. Quest'operatore riserva lo spazio memoria necessario nello heap e restituisce l'indirizzo di memoria dell'area riservata. Quest'indirizzo non deve andare perso. È quindi necessario salvarlo in una variabile:

```
Scanner input;  
input = new Scanner(System.in);
```

## Allocazione e rilascio della memoria

In Java, non è necessario **rilasciare** esplicitamente la memoria dinamica (come ad esempio in C).

Se ne occupa automaticamente il **garbage collector** quando una determinata area non ha più nessun riferimento.

## Gli arrays

Gli **arrays** vengono utilizzati per aggregare dati **di tipo omogeneo in vettori di dati o matrici di dati**. Le matrici possono essere bidimensionali o multidimensionali.

Vettore di dati:

Indice da 0 a 3

3	72	4	1
---	----	---	---

Matrice bidimensionale di dati:

Indice da 0 a 3

Indice da 0 a 1

3	12	8	11
31	23	-4	71

# Dichiarazione degli arrays

Per **dichiarare un array** bisogna:

- dichiarare quale è il tipo base,
- dichiarare quale è la dimensione.

Ad esempio:

```
int[] mioArr1 = new int[10];
```

Dichiara che la variabile `mioArr1` è un array di 10 valori di tipo `int`.

```
int[] mioArr2 = { 3, 72, 4, 1 };
```

Dichiara che la variabile `mioArr2` è un array di 4 `int` ed inizializza l'array con i valori di partenza.

# Utilizzo degli arrays

Per **accedere alle componenti di un array** si utilizza la notazione seguente:

```
variabileArray[indice]
```

Ad esempio:

```
int valore = mioArr1[3];
```

Attenzione:

- L'indice parte da 0. Quindi `mioArr1[3]` è il quarto elemento dell'array.
- `indice` deve essere un valore di tipo intero.

## Esempio

```
import java.util.Scanner;

public class TrovaMinimo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Inserire due valori: ");
        int n1 = input.nextInt();
        int n2 = input.nextInt();
        input.close();

        int min;
        if (n1 < n2)
            min = n1;
        else
            min = n2;
        System.out.println("Min: " + min);
    }
}
```

## Esempio

```
Scanner input = new Scanner(System.in);
System.out.print("Inserire sei valori: ");
int n1 = input.nextInt();
int n2 = input.nextInt();
int n3 = input.nextInt();

...
int n6 = input.nextInt();
input.close();

int min;
if (n1 < n2 && n1 < n3 && ... )
    min = n1;
else if (n2 < n1 && n2 < n3 && ... )
    min = n2;
...
else
    min = n6;
System.out.println("Min: " + min);
```

## Esempio

```
Scanner input = new Scanner(System.in);
System.out.print("Inserire sei valori: ");
int n1 = input.nextInt();
int n2 = input.nextInt();
int n3 = input.nextInt();

...
int n6 = input.nextInt();
input.close();

int min = n1;
if (n2 < min)
    min = n2;
if (n3 < min)
    min = n3;

...
if (n6 < min)
    min = n6;
System.out.println("Min: " + min);
```

## Esempio

```
import java.util.Scanner;

public class TrovaMinimo {
    public static void main(String[] args) {
        int[] n = new int[6];
        Scanner input = new Scanner(System.in);
        System.out.print("Inserire sei valori: ");
        for (int i = 0; i < n.length; i++)
            n[i] = input.nextInt();
        input.close();

        int min = n[0];
        for (int i = 1; i < n.length; i++)
            if (n[i] < min)
                min = n[i];
        System.out.println("Min: " + min);
    }
}
```

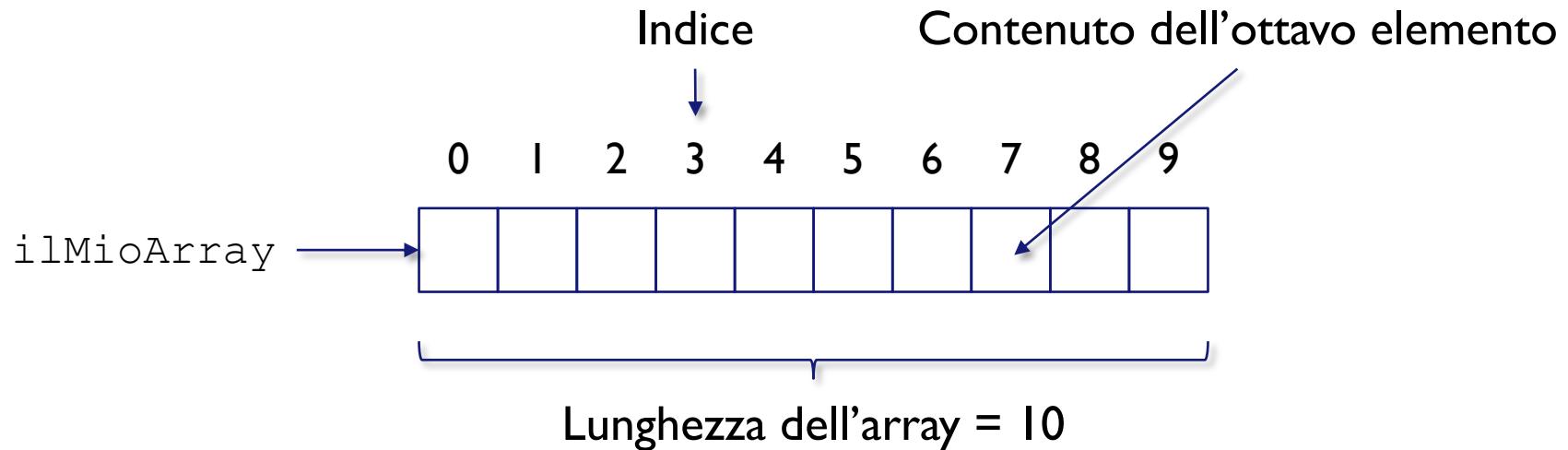
# Arrays

L'**array** è un **tipo di dato strutturato** i cui elementi sono tutti dello stesso tipo.

Permette di immagazzinare e gestire, in maniera efficiente, grandi quantità di informazione **omogenea**.

Per il tipo degli elementi si può usare **qualsiasi tipo di dato**, sia tipi primitivi che tipi riferimento, compresi altri arrays o classi (che discuteremo in seguito).

## L'array monodimensionale



La **lunghezza** dell'array corrisponde al **numero di elementi** in esso contenuti.  
L'istruzione `ilMioArray.length` permette di ottenere tale lunghezza.

In Java, la numerazione degli elementi **parte sempre da 0**:

- L'indice del primo elemento è 0.
- L'indice dell'ultimo elemento è  $N-1$  dove  $N$  è la lunghezza dell'array.

## L'array monodimensionale: dichiarazione

In Java, una variabile **array monodimensionale** viene dichiarata con il nome del tipo dei suoi elementi, seguito da una coppia di parentesi quadre ed il relativo identificatore.

Esempi:

```
String[] nomi = null;  
int[] valori = null;  
double[] numeri = null;
```

```
String nomi[] = null;  
int valori[] = null;  
double numeri[] = null;
```



È possibile dichiarare l'array anche così,  
ma è meglio utilizzare la variante con le  
parentesi dopo il tipo di dato.

Una variabile così dichiarata assume il **valore iniziale null**.

## L'array monodimensionale: istanziazione

Ad ogni variabile di tipo array può venir associato un riferimento ad un oggetto array che risiede nello heap. È quindi **necessario creare l'oggetto array** da referenziare utilizzando l'**operatore new**.

L'operatore `new` riserva lo spazio di memoria necessario nello heap e restituisce l'indirizzo di memoria dell'area riservata.

La **lunghezza** dell'array (dimensione) viene **specificata al momento della creazione**. Da quel momento in poi **non** è più possibile ridimensionare l'array.

Esempi:

```
nomi = new String[10];
valori = new int[175];
numeri = new double[88];
```

## Dichiarazione d'array: esempi

```
byte[] byteArray = new byte[10]; // Un array di byte
short[] shortArray = new short[256]; // Un array di short
long[] longArray = new long[70]; // Un array di long

float[] floatArray = new float[1000]; // Un array di float
double[] doubleArray = new double[4]; // Un array di double

boolean[] booleanArray = new boolean[55]; // Un array di boolean

char[] charArray = new char[30]; // Un array di char

String[] stringArray = new String[99]; // Un array di String
```

## L'array monodimensionale: utilizzo

Per **accedere agli elementi di un array** si utilizza la notazione seguente:

```
variabileArray[indice]
```

Esempio:

```
String[] ilMioArray = new String[3];
ilMioArray[0] = "Ciao";
ilMioArray[1] = "come";
ilMioArray[2] = "stai?";
```

Bisogna ricordare che:

- La numerazione dell'**indice parte da 0**.
- Il valore di **indice** deve essere di **tipo intero**.

## Creazione e accesso all'array

Ricordarsi sempre che:

- La **dichiarazione** di una variabile di tipo array **non crea l'array**. Per crearlo bisogna utilizzare l'**operatore new**. La dimensione (lunghezza) va specificata al momento della creazione.
- Se si cerca di accedere ad un array che **non è stato creato**, il programma in esecuzione si interrompe segnalando l'errore **NullPointerException**.
- Se si cerca di accedere ad un elemento  **$k < 0$  o  $k \geq \text{lunghezza}$** , il programma in esecuzione si interrompe segnalando l'errore **ArrayIndexOutOfBoundsException**.

## Creazione e accesso all'array: esempi

```
int x;  
int[] valori = null;  
x = valori[5]; // NullPointerException  
  
valori = new int[200];  
x = valori[5];  
x = valori[0];  
x = valori[valori.length - 1];  
  
x = valori[-5]; // ArrayIndexOutOfBoundsException  
x = valori[300]; // ArrayIndexOutOfBoundsException  
x = valori[valori.length]; // ArrayIndexOutOfBoundsException
```

## Dimensione dell'array

La dimensione dell'array viene **specificata al momento della creazione** dell'array. Da quel momento in poi, **non** è più possibile ridimensionare l'array.

L'istruzione `ilMioArray.length` permette di ottenere la dimensione (lunghezza) dell'array.

Se lo spazio all'interno di un array non dovesse bastare, è necessario crearne uno nuovo più grande e travasare tutti i valori contenuti in quello di partenza.

## Dimensione dell'array: esempio

```
public class IstanziazioneArray {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
        System.out.print("Dimensione array: ");  
        int dimensione = input.nextInt();  
  
        float[] valori = new float[dimensione];  
        System.out.print("Valori: ");  
        for (int i = 0; i < valori.length; i++)  
            valori[i] = input.nextFloat();  
        input.close();  
  
        ...  
  
        // Raddoppia la dimensione dell'array  
        float[] tmp = new float[valori.length * 2];  
        for (int i = 0; i < dimensione; i++)  
            tmp[i] = valori[i];  
        valori = tmp;  
        System.out.println("Lunghezza: " + valori.length);  
    }  
}
```

# Dichiarazione, istanziazione e inizializzazione

Dichiarazione, istanziazione e successiva inizializzazione:

```
int[] numeri = null;  
numeri = new int[6];  
numeri[0] = 50; numeri[1] = 20; numeri[2] = 45;  
numeri[3] = 82; numeri[4] = 25; numeri[5] = 63;
```

... oppure anche:

```
int[] numeri = new int[6];  
numeri[0] = 50; numeri[1] = 20; numeri[2] = 45;  
numeri[3] = 82; numeri[4] = 25; numeri[5] = 63;
```

... oppure:

```
int[] numeri = { 50, 20, 45, 82, 25, 63 };
```

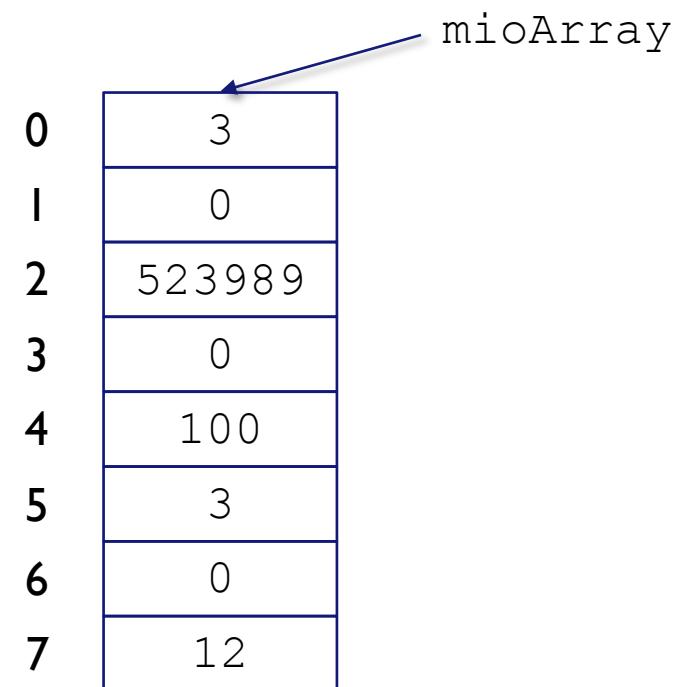
La **lunghezza** dell'array viene fissata quando si istanzia l'array!

## Arrays: inizializzazione e accesso

Le celle di un array che non vengono esplicitamente inizializzate contengono **il valore di default** del tipo di dato dell'array (0 se numeri o caratteri, false se boolean o null se oggetti).

**È comunque consigliato sempre inizializzare l'array esplicitamente.**

```
int[] mioArray = null;  
mioArray = new int[8];  
mioArray[4] = 100;  
mioArray[5] = 3;  
mioArray[7] = 12;  
mioArray[2] = 523989;  
int x = mioArray[5];  
mioArray[0] = x;
```



## Inizializzatore d'array

**L'inizializzatore di array** (array initializer) permette di creare e contemporaneamente inizializzare i valori contenuti in un array. La **lunghezza** dell'array equivale al **numero di elementi** presenti nell'inizializzatore.

```
int[] valori = { 1, 4, 9 };
```

Equivale a:

```
int[] valori = new int[3];
valori[0] = 1;
valori[1] = 4;
valori[2] = 9;
```

È possibile inizializzare un array con l'inizializzatore **solo una volta** al momento della dichiarazione. Il compilatore non permette una doppia inizializzazione.

## Inizializzatore d'array: esempio

```
public class InizializzatoreArray {  
    public static void main(String[] args) {  
        String[] frutti = { "Pera", "Mela", "Kiwi", "Uva" };  
  
        String str = "";  
        for (int i = 0; i < frutti.length; i++) {  
            String frutto = frutti[i];  
            str += frutto;  
  
            if (i < frutti.length - 1)  
                str += ", ";  
        }  
  
        System.out.println(str);  
    }  
}
```

## Array e cicli

L'operazione più tipica che possiamo compiere su di un array è quella di **scorrere i suoi elementi**, uno ad uno, dal primo all'ultimo.

Questo lo si può fare con un **ciclo** del tipo:

```
... // Esegui l'inizializzazione necessaria
for (int i = 0; i < arr.length; i++) {
    ... // Processa arr[i]
}
```

Ad esempio, per sommare tutti i numeri contenuti in un array di double:

```
double[] arr = new double[10];
... // Carica il contenuto di arr
double somma = 0.;
for (int i = 0; i < arr.length; i++) {
    somma += arr[i];
}
```

## Array e cicli: esempi

Codice per contare il numero di volte che due elementi consecutivi sono uguali:

```
int[] arr = new int[10];
... // Carica il contenuto di arr
int cnt = 0;
for (int i = 0; i < arr.length - 1; i++) {
    if (arr[i] == arr[i + 1])
        cnt++;
}
```

Codice per trovare il valore massimo presente in un array:

```
int[] arr = new int[10];
... // Carica il contenuto di arr
int max = arr[0];
for (int i = 1; i < arr.length; i++) {
    if (arr[i] > max)
        max = arr[i];
}
```

## Istruzione for avanzata (foreach)

L'istruzione **for avanzata** permette di **iterare** sugli **arrays**. Ad ogni iterazione viene restituito un elemento dell'array. L'array viene percorso completamente dalla prima all'ultima cella.

```
public class SommaNumeri {  
    public static void main(String[] args) {  
        int[] numeri = { 1, 3, 5, 2, 12, 8, 21, 56, 3, 23 };  
        int somma = 0;  
  
        for (int numero : numeri) {  
            System.out.println("Elemento: " + numero);  
            somma += numero;  
        }  
        System.out.println("Somma: " + somma);  
    }  
}
```

## Ciclo foreach: esempio

```
double[] valori = { 1.11, 7.77, 2.32, 99.1, 1.75, -8.22 };  
double somma = 0.;  
  
for (int i = 0; i < valori.length; i++) {  
    double valore = valori[i];  
    somma += valore;  
}  
  
System.out.println("Media: " + somma / valori.length);
```

Oppure:

```
double[] valori = { 1.11, 7.77, 2.32, 99.1, 1.75, -8.22 };  
double somma = 0.;  
  
for (double valore : valori)  
    somma += valore; ← Non c'è a disposizione l'indice!  
  
System.out.println("Media: " + somma / valori.length);
```

## Ciclo foreach: una nota

Il ciclo foreach permette di **accedere ai valori** contenuti nelle celle dell'array, ma non di modificare direttamente le celle (intese come locazioni di memoria dell'array).

Ad esempio:

```
int[] numeri = { 1, 3, 5, 2, 12, 8, 21, 56, 3, 23 };
for (int numero : numeri) {
    ...
    // NON modifica il contenuto dell'array!
    numero = 17;
    ...
}
```

Assegna il valore 17 alla variabile di controllo numero, ma **non modifica il contenuto dell'array**.

## Copia di un array: shallow copy

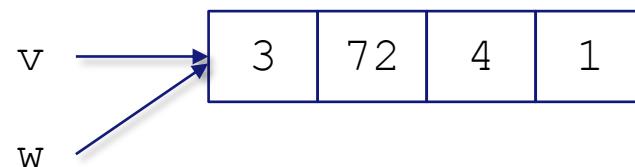
Supponendo di avere dichiarato il seguente array:

```
int [] v = { 3, 72, 4, 1 };
```

Si vuole **fare una copia** di questo array:

```
int [] w = v;
```

L'effetto è semplicemente questo:



È stata fatta quella che viene definita una **copia superficiale** (shallow copy).

## Copia di un array: deep copy

Per fare una **copia vera e propria** di un array (e come vedremo di un qualsiasi tipo riferimento) bisogna procedere come segue:

- **Allocare la memoria** necessaria.
- **Copiare, valore per valore**, i dati nella nuova zona di memoria allocata precedentemente.
- **Importante**: se i valori nell'array sono a loro volta oggetti, bisogna procedere ad allocare la memoria e a copiare i valori ricorsivamente.

## Copia di un array: deep copy

Per esempio:

```
int[] v = { 3, 72, 4, 1 };
// Crea un nuovo array con la stessa taglia dell'originale
int[] w = new int[v.length];
for (int i = 0; i < v.length; i++) {
    // Copia ogni elemento da v a w
    w[i] = v[i];
}
```

Per comodità si può usare anche la routine predefinita `System.arraycopy()`:

```
int[] v = { 3, 72, 4, 1 };
// Crea un nuovo array con la stessa taglia dell'originale
int[] w = new int[v.length];
// Copia l'intero array
System.arraycopy(v, 0, w, 0, v.length);
```

## Confronti di arrays

Per scoprire se il contenuto di due arrays è uguale **NON si può utilizzare l'operatore ==**. Come per la copia, vanno **confrontati i valori** contenuti nell'array.

In alternativa, si può utilizzare la **routine predefinita `Arrays.equals()`**. Bisogna però ricordarsi di importare la classe Arrays (`java.util.Arrays`).

**Importante:** la funzione **`equals()`** della classe Arrays ritorna true solo se i due arrays contengono gli **stessi elementi nello stesso ordine**.

## Confronti di arrays: esempio

```
public class ConfrontoArrays {  
    public static void main(String[] args) {  
        int[] a1 = { 1, 2, 3, 4, 5, 6 };  
        int[] a2 = { 1, 2, 3, 4 };  
  
        boolean uguale = true;  
        if (a1.length != a2.length)  
            uguale = false;  
        else {  
            for (int i = 0; i < a1.length; i++) {  
                if (a1[i] != a2[i]) {  
                    uguale = false;  
                    break;  
                }  
            }  
        }  
  
        System.out.println("a1 uguale a a2? " + (uguale ? "Sì" : "No"));  
    }  
}
```



## Confronti di arrays: esempio

```
import java.util.Arrays;

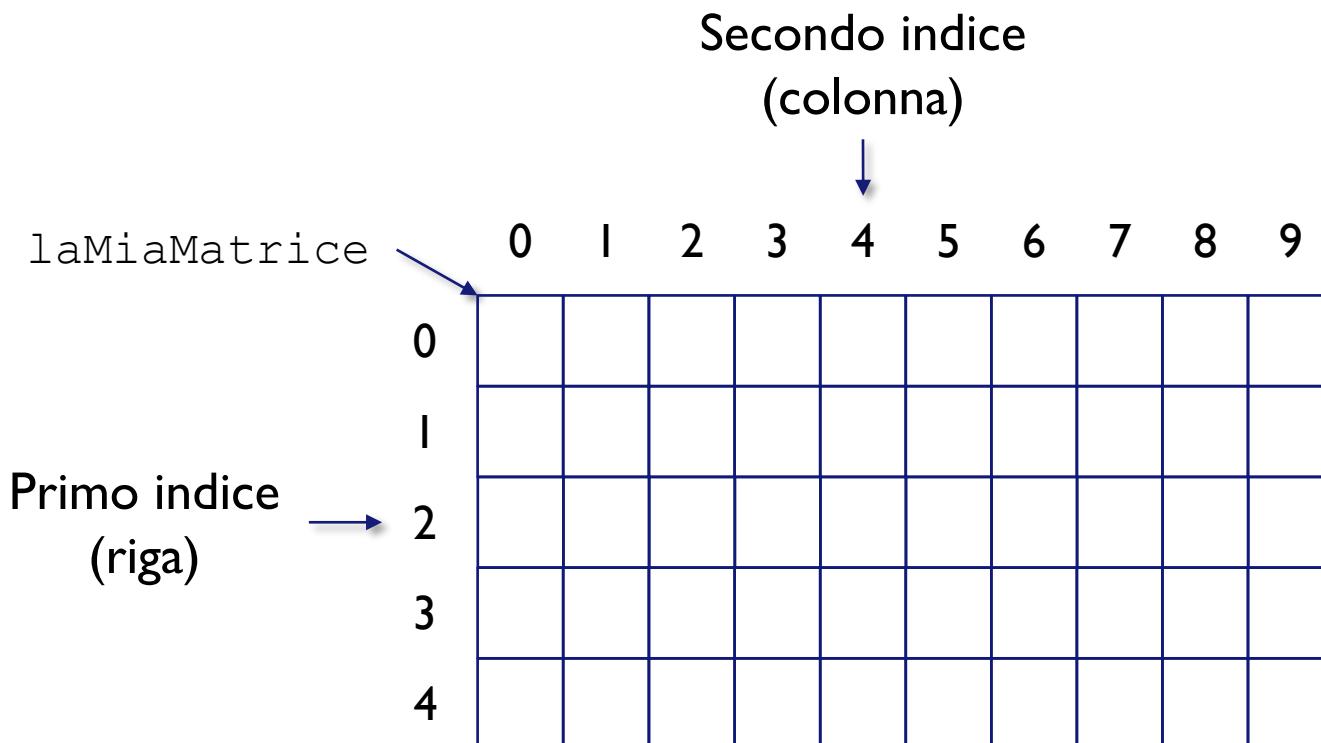
public class ConfrontoArrays {
    public static void main(String[] args) {
        int[] a1 = { 1, 2, 3, 4, 5, 6 };
        int[] a2 = { 1, 2, 3, 4, 5, 6 };
        int[] a3 = { 1, 2, 3, 4 };

        System.out.println("a1 uguale a a2? "
            + (Arrays.equals(a1, a2) ? "Sì" : "No"));
        System.out.println("a1 uguale a a3? "
            + (Arrays.equals(a1, a3) ? "Sì" : "No"));
    }
}
```

## Arrays multi-dimensionali

È possibile definire array ***multi-dimensional*** (matrici). Questi tipi di dato sono caratterizzati da due o più indici.

Esempio: array bidimensionale (due indici).



## Arrays multi-dimensionali

In Java, una variabile **array multi-dimensionale** viene dichiarata con il nome del tipo dei suoi elementi, seguito da un numero di coppie di parentesi quadre **[ ]** uguale al numero di dimensioni dell'array.

Esempi:

```
String[][] persone = new String[4][10];
long[][][] valori = new long[7][34][3];
float[][][] [] numeri = new float[125][9][15][240];
```

## Arrays multi-dimensionali

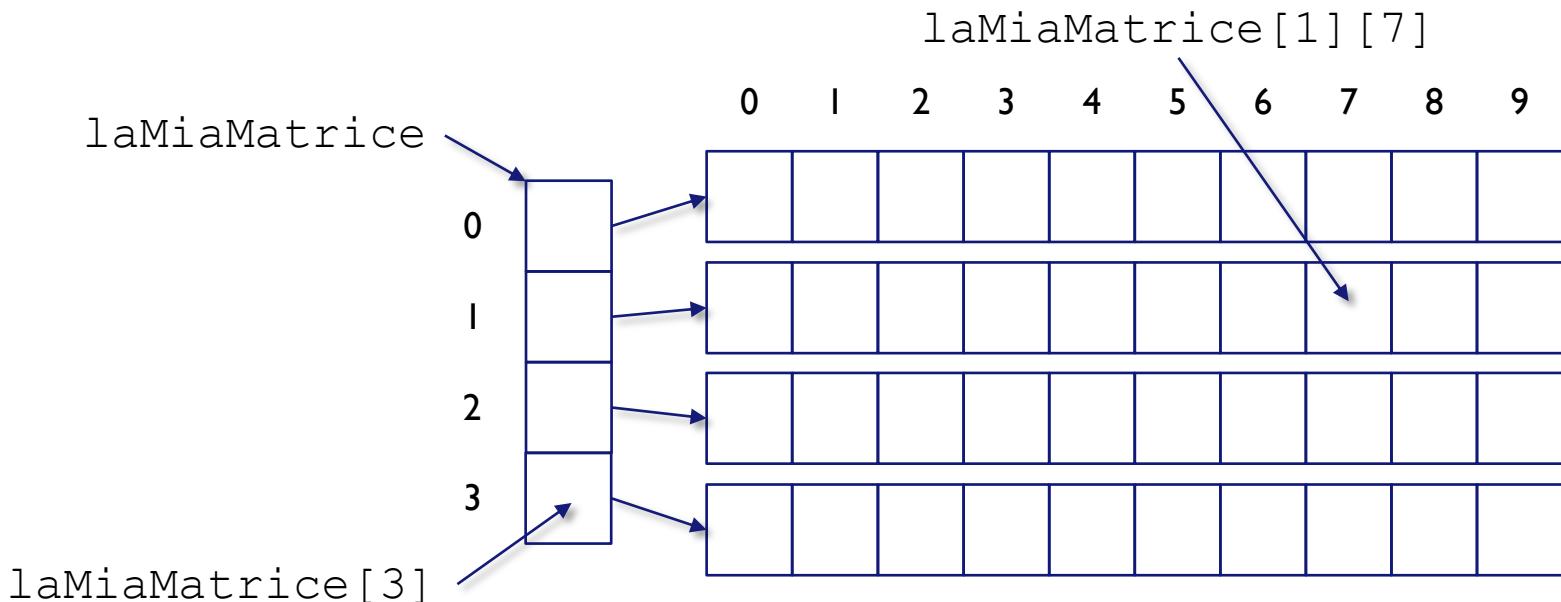
È possibile utilizzare un inizializzatore d'array anche per gli array multi-dimensional:

```
public class EsempioMultiDim {  
    public static void main(String[] args) {  
        String[][] nomi = { { "Mr. ", "Mrs. ", "Ms. " },  
                            { "Smith", "Jones", "Brown" } };  
        System.out.println(nomi[0][0] + nomi[1][0]); // Mr. Smith  
        System.out.println(nomi[0][2] + nomi[1][1]); // Ms. Jones  
    }  
}
```



# Arrays multi-dimensionali

In Java, un array multi-dimensionale è un array di array di ...



```
int righe = 4, colonne = 10;
double[][] laMiaMatrice = new double[righe] [colonne];
for (int i = 0; i < righe; i++)
    for (int j = 0; j < colonne; j++)
        laMiaMatrice[i] [j] = Math.random() * 100;
```

## Arrays multi-dimensionali: esempio

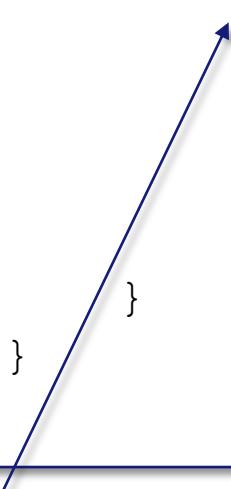
```
public class ArrayMultiDim {  
    public static void main(String[] args) {  
        int[][] nums = { { 99, 85, 98 }, { 98, 57, 78 },  
                        { 92, 77, 76 }, { 94, 32, 11 },  
                        { 99, 34, 22 }, { 90, 46, 54 },  
                        { 76, 59, 88 }, { 92, 66, 89 },  
                        { 97, 71, 24 }, { 89, 29, 38 } };  
  
        for (int i = 0; i < nums.length; i++) {  
            for (int j = 0; j < nums[i].length; j++) {  
                System.out.print(nums[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

 A red triangular warning icon with a white exclamation mark inside, positioned above the nested loops. Two red arrows point from the warning icon towards the code: one arrow points from the warning icon to the closing brace of the inner loop (the '}' after 'j++'), and another arrow points from the warning icon to the opening brace of the outer loop (the '{' before 'i = 0').

Per percorrere gli arrays multi-dimensional si usano cicli annidati.

## Arrays multi-dimensionali: esempio

```
public class ArrayMultiDim {  
    public static void main(String[] args) {  
        int[][] nums = { { 99, 85, 98 }, { 98, 57, 78 },  
                        { 92, 77, 76 }, { 94, 32, 11 },  
                        { 99, 34, 22 }, { 90, 46, 54 },  
                        { 76, 59, 88 }, { 92, 66, 89 },  
                        { 97, 71, 24 }, { 89, 29, 38 } };  
  
        for (int i = 0; i < nums.length; i++) {  
            int[] rigaCorrente = nums[i];  
            for (int j = 0; j < rigaCorrente.length; j++) {  
                System.out.print(rigaCorrente[j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```



Una riga dell'array.

## Arrays multi-dimensionali e foreach

```
public class ArrayMultiDimForeach {  
    public static void main(String[] args) {  
        int[][] nums = { { 99, 85, 98 }, { 98, 57, 78 },  
                        { 92, 77, 76 }, { 94, 32, 11 },  
                        { 99, 34, 22 }, { 90, 46, 54 },  
                        { 76, 59, 88 }, { 92, 66, 89 },  
                        { 97, 71, 24 }, { 89, 29, 38 } };  
  
        for (int[] riga : nums) {  
            for (int valore : riga) {  
                System.out.print(valore + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

## Utilities per gli arrays

Utilizzando la proprietà `length` di un array è possibile ottenere il valore della sua dimensione (lunghezza).

```
int lunghezza = mioArr.length;
```

Per copiare un array è possibile utilizzare la procedura `arraycopy()` della classe `System`.

```
int[] mioArrCopia = new int[mioArr.length];
System.arraycopy(mioArr, 0, mioArrCopia, 0, mioArr.length);
```

## Utilities per gli arrays: la classe Arrays

La funzione `equals()` permette di confrontare due arrays monodimensionali.

```
boolean uguali = Arrays.equals(arr1, arr2);
```

La funzione `copyOf()` permette di copiare un array monodimensionale (o parte di esso).

```
int[] mioArrCopia = Arrays.copyOf(mioArr, mioArr.length);
```

La procedura `sort()` permette di ordinare, in ordine crescente, gli elementi di un array monodimensionale.

```
Arrays.sort(mioArr);
```

Importante: ricordarsi di importare la classe `Arrays` (`java.util.Arrays`).

## Utilities per gli arrays: esempio

```
import java.util.Arrays;

public class OrdinaNumeri {
    public static void main(String[] args) {
        int[] numeri = { 13, 19, 5, 99, 71 };

        // Copia l'array originale
        int[] numeriOrdinati = Arrays.copyOf(numeri, numeri.length);
        // Ordina la copia
        Arrays.sort(numeriOrdinati);

        // Mostra l'array originale e quello ordinato
        System.out.println("Non ordinati: " + Arrays.toString(numeri));
        System.out.println("Ordinati: " + Arrays.toString(numeriOrdinati));

        // Confronta i due arrays
        System.out.println("Sono uguali? "
                + Arrays.equals(numeri, numeriOrdinati));
    }
}
```

# Riepilogo

- Tipi di dato strutturato e non strutturato
- Gestione della memoria tipi di dato primitivi e di riferimento
- Gestione della memoria (allocazione e rilascio)
- Array monodimensionale (dichiarazione, istanziazione e utilizzo)
- Inizializzatore di array
- Array e cicli
- Istruzione `for` avanzata (`foreach`)
- Copia di un array (shallow copy vs deep copy)
- Confronti di arrays
- Array multi-dimensionali
- Utilities per gli arrays