

Programmazione I-B 2022-23

Laboratorio T2

(penultima cifra matricola SPARI)

Attilio Fiandrotti

attilio.fiandrotti@unito.it

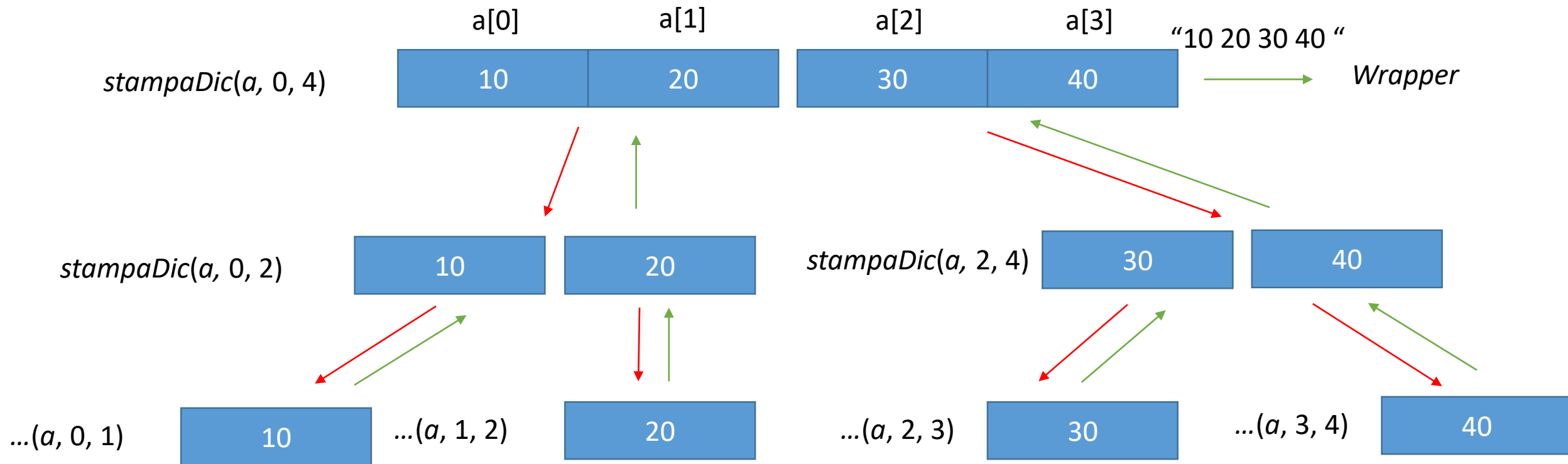
1 dicembre 2022

Outline

- Soluzione esercizi ricorsione dicotomica
- Le matrici in Java
- Esercizi sulle matrici e sulla ricorsione

Soluzione esercizi ricorsione dicotomica

ESERCIZIO ArrayDic – stampaDic()



```
$java ArrayDicTest  
10 20 30 40
```

Es1 – raddoppioPari()

Scrivere un metodo dicotomico ricorsivo *raddoppioPari()* tale che:

- ha un array di interi *a* ed un intero *v* come parametri;
- modifica *a* sfruttando l'aliasing in modo che ogni elemento di posizione pari **e** maggiore di *v* sia raddoppiato
- l'array *a* può essere null o contenere 0 elementi
- disponga dell'opportuno metodo wrapper

Es1 – raddoppioPari()

```
public static void raddoppioPari(int[] a, int v, int l, int r) {
    // Caso elementare: ho un (sotto)array a di un solo elemento
    if (r == l+1) {
        if ((l%2 == 0) && (l >= v)) {
            a[l] = 2 * a[l];
        }
    }
    // Caso generale: il (sotto)array a ha almeno due elementi
    else {
        int m = (r+l)/2;
        raddoppioPari(a, v, l, m);
        raddoppioPari(a, v, m, r);
    }
    // Esplicitato per ragioni didattiche
    return;
}

public static int[] raddoppioPari(int[] a, int v) {
    int[] b = null;

    if (a != null) {
        b = Arrays.copyOf(a, a.length);
        if (a.length > 0) {
            raddoppioPari(b, v, 0 /* l */, a.length /* r */);
        }
    }

    return b;
}
```

Es2 – raddoppioPariCnt()

Scrivere un metodo dicotomico ricorsivo *raddoppioPariCnt()* che:

- ha un array di interi a ed un intero v come parametri;
- modifica a sfruttando l'aliasing in modo tale ogni elemento di posizione pari e maggiore di v sia raddoppiato
- ritorna il numero di sostituzioni effettuate in a
- disponga dell'opportuno metodo wrapper

Es2 – raddoppioPariCnt()

```
public static int raddoppioPariCnt(int[] a, int v, int l, int r) {
    // Contatore di sostituzioni
    int cnt = 0;
    // Caso base: ho un (sotto)array a di un solo elemento
    if (r-l == 1) {
        if ((l%2 == 0) && (l >= v)) {
            a[l] = 2 * a[l];
            cnt = cnt + 1;
        }
        return cnt;
    }
    // Caso generale: il (sotto)array a ha almeno due elementi, procedo
    else {
        int m = (r+1)/2;
        int sostLeft = raddoppioPariCnt(a, v, l, m);
        int sostRight = raddoppioPariCnt(a, v, m, r);
        cnt = sostLeft + sostRight;
        return cnt;
    }
}
```

```
public static int raddoppioPariCnt(int[] a, int v) {
    int cnt = 0;
    if (a != null) {
        cnt = raddoppioPariCnt(a, v, 0, a.length);
    }
    return cnt;
}
```


Es3 – copiaPrecedente()

Scrivere un metodo ricorsivo dicotomico `copiaPrecedente()` che:

- ha un array di interi a come parametro
- restituisce a , in cui ogni elemento con **valore** pari è ricoperto dal valore dell'elemento che lo precede, se esiste
- disponga dell'opportuno metodo wrapper

Es3 – copiaPrecedente()

```
public static void copiaPrecedente(int[] a, int l, int r) {
    if (r-l == 1) {
        // La prima condizione di raddoppio "a[l] ha un predecessore" é in AND logico con "a[l]%2 == 0"
        if((l >= 1) && (l % 2 == 0)) {
            a[l] = a[l-1];
        }
        return;
    }
    else {
        int m = (r+1)/2;
        copiaPrecedente(a, l, m);
        copiaPrecedente(a, m, r);
        return;
    }
}

public static int[] copiaPrecedente(int[] a) {
    int[] b = null;
    if (a != null) {
        b = Arrays.copyOf(a, a.length);
        copiaPrecedente(b, 0, b.length);
    }
    return b;
}
```

Es4 – `scambiaSuccessivoDic()`

Scrivere un metodo ricorsivo **dicotomico** *scambiaSuccessivoDic()* che:

- ha un array di interi a ed un intero v come parametri
- restituisce a , in cui ogni elemento con **valore** maggiore di v è scambiato con l'elemento immediatamente successivo, se esiste
- disponga dell'opportuno metodo wrapper

Es4 – scambiaSuccessivoDic()

```
public static void scambiaSuccessivoDic(int[] a, int v, int l, int r) {  
    if (r-l == 1) {  
        if((l+1 < a.length) && (a[l] > v)) {  
            int tmp = a[l];  
            a[l] = a[l+1];  
            a[l+1] = tmp;  
        }  
        return;  
    }  
    else {  
        int m = (r+l)/2;  
        scambiaSuccessivoDic(a, v, l, m);  
        scambiaSuccessivoDic(a, v, m, r);  
        return;  
    }  
}
```

equivale ad $l < a.length - 1$

```
public static int[] scambiaSuccessivoDic(int[] a, int v) {  
    int[] b = null;  
    if (a != null) {  
        int[] b = Arrays.copyOf(a, a.length);  
        scambiaSuccessivoDic(b, v, 0, b.length);  
    }  
    return b;  
}
```

Es4 – `scambiaSuccessivoCov()`

Scrivere un metodo ricorsivo **covariante** `scambiaSuccessivoCov()` che:

- ha un array di interi a ed un intero v come parametri
- restituisce a , in cui ogni elemento con **valore** maggiore di v è scambiato con l'elemento immediatamente successivo, se esiste
- disponga dell'opportuno metodo wrapper

Es4 – scambiaSuccessivoCov()

```
public static void scambiaSuccessivoCov(int[] a, int v, int i) {  
    //Notare che l'ultimo elemento valido dell'array non può avere successori  
    if (i == a.length - 1) {  
        return;  
    }  
    else {  
        scambiaSuccessivoCov(a, v, i+1);  
        if((a[i] > v) && (i+1 < a.length)) {  
            int tmp = a[i];  
            a[i] = a[i+1];  
            a[i+1] = tmp;  
        }  
        return;  
    }  
}  
  
public static int[] scambiaSuccessivoCov(int[] a, int v) {  
    int[] b = null;  
    if (a != null) {  
        b = Arrays.copyOf(a, a.length);  
        scambiaSuccessivoCov(b, v, 0);  
    }  
    return b;  
}
```

ricorsione per **i** crescente

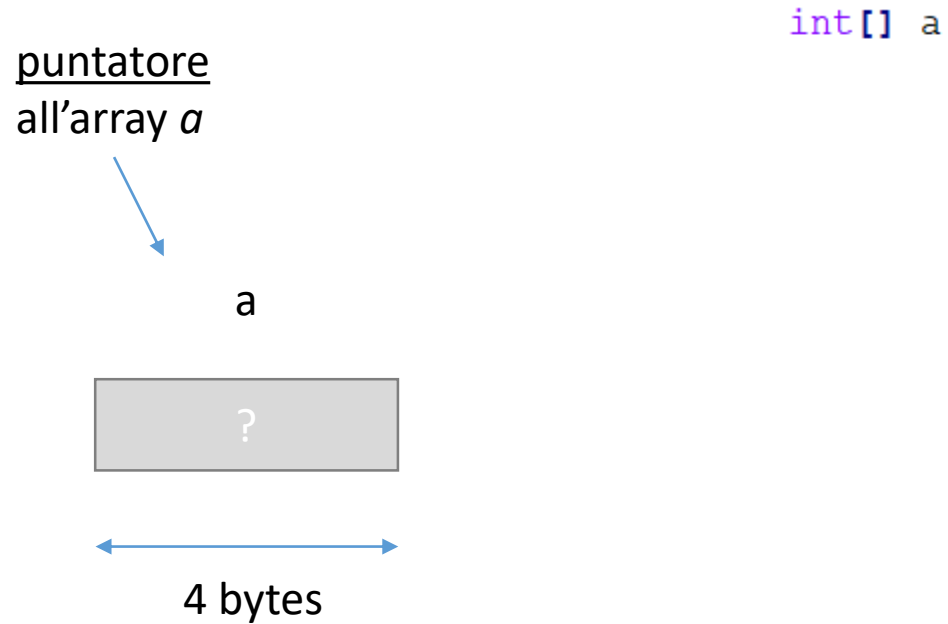
Le matrici in Java

Gli array in Java

- Primo esempio di *struttura dati*
- Ha dimensione fissa definita al momento della creazione dell'array
- Elementi omogenei per tipo (tutti int, o float, o boolean, etc.)
- Si accede all'elemento n-esimo tramite *puntatore* come *a[i]*
- Per un array di *n* elementi l'indice *i* va da 0 a n-1
- La lunghezza è disponibile a runtime come *a.length* (*!= C, C++, etc.*)

Gli array in Java

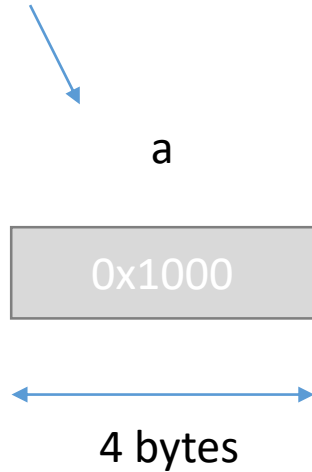
- Definiamo un array di 3 interi di valore 10, 20 e 30



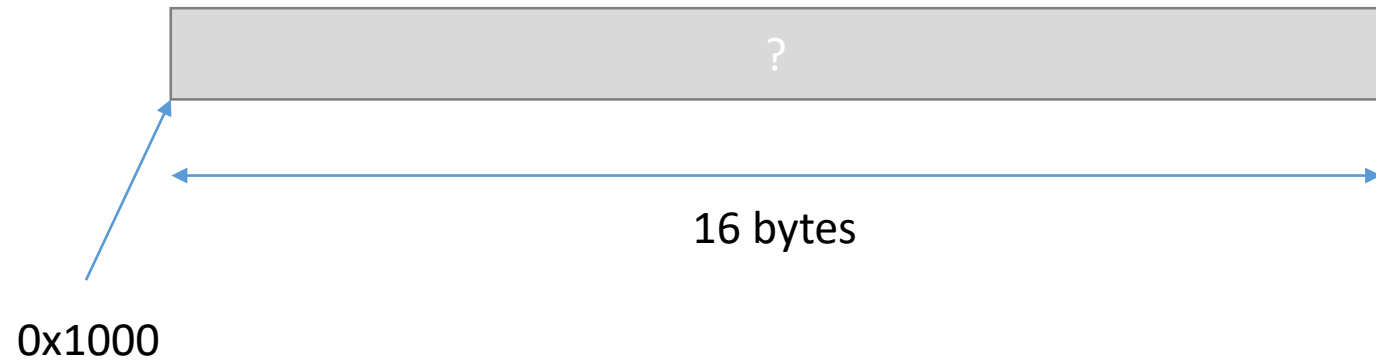
Gli array in Java

- Definiamo un array di 3 interi di valore 10, 20 e 30

puntatore
all'array *a*



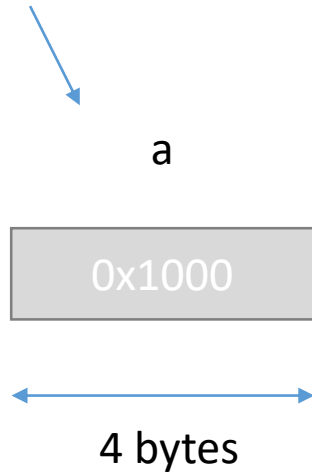
```
int[] a = new int[3];
```



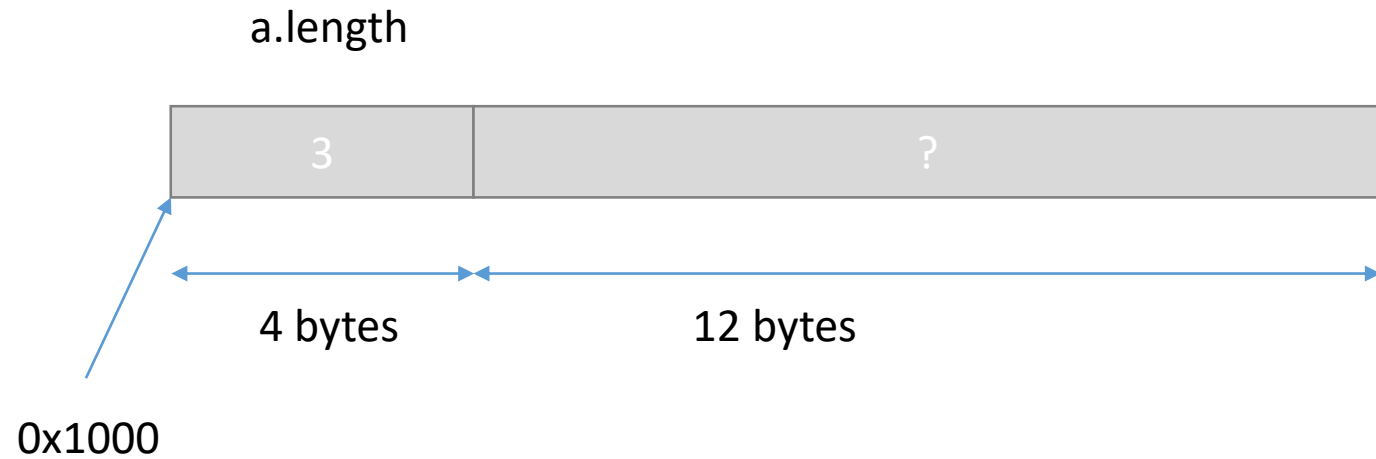
Gli array in Java

- Definiamo un array di 3 interi di valore 10, 20 e 30

puntatore
all'array *a*

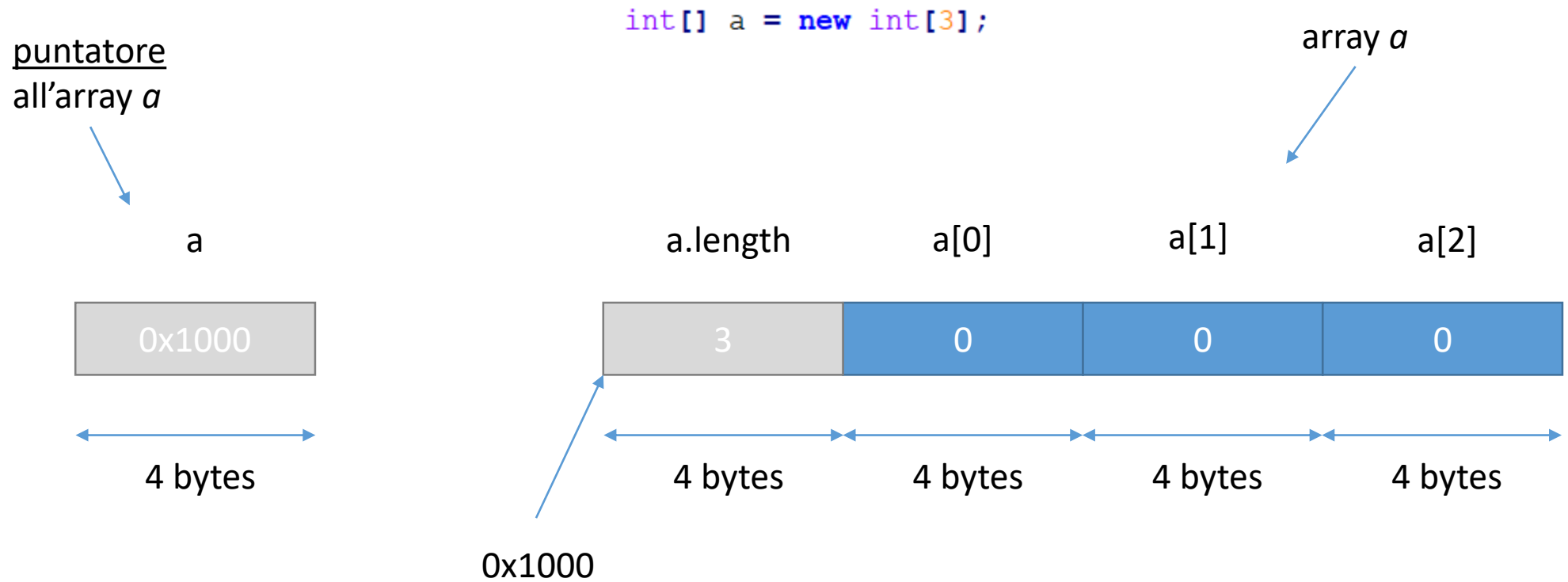


```
int[] a = new int[3];
```



Gli array in Java

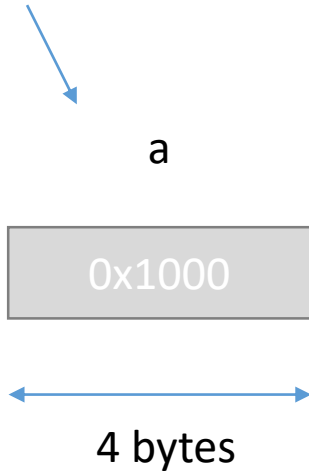
- Definiamo un array di 3 interi di valore 10, 20 e 30



Gli array in Java

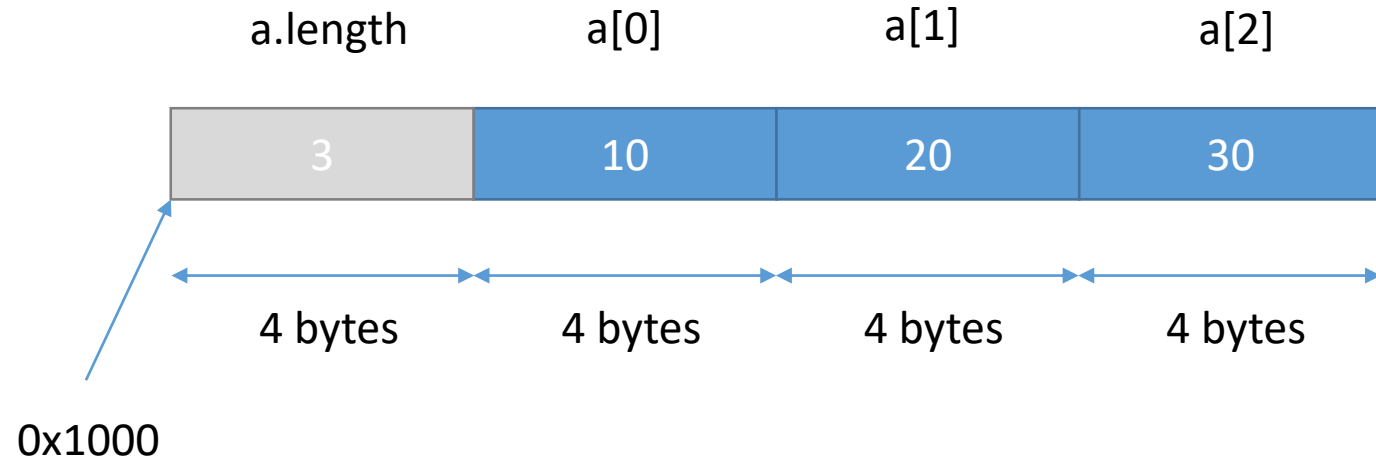
- Definiamo un array di 3 interi di valore 10, 20 e 30

puntatore
all'array *a*



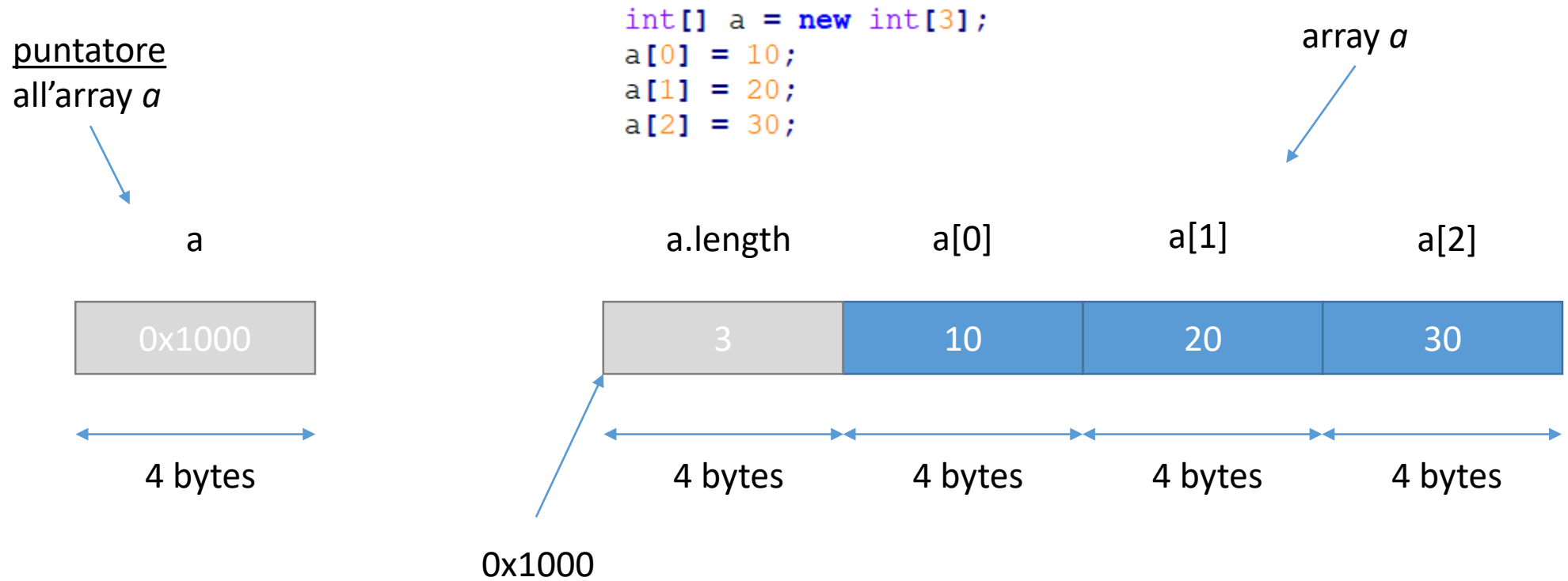
```
int[] a = new int[3];  
a[0] = 10;  
a[1] = 20;  
a[2] = 30;
```

array *a*



Gli array in Java

- Definiamo un array di 3 interi di valore 10, 20 e 30

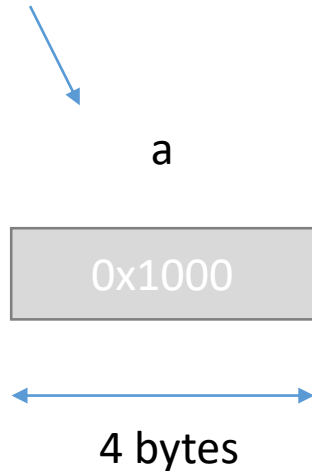


4 byte per puntatore + 16 bytes per array = 20 bytes

Gli array in Java

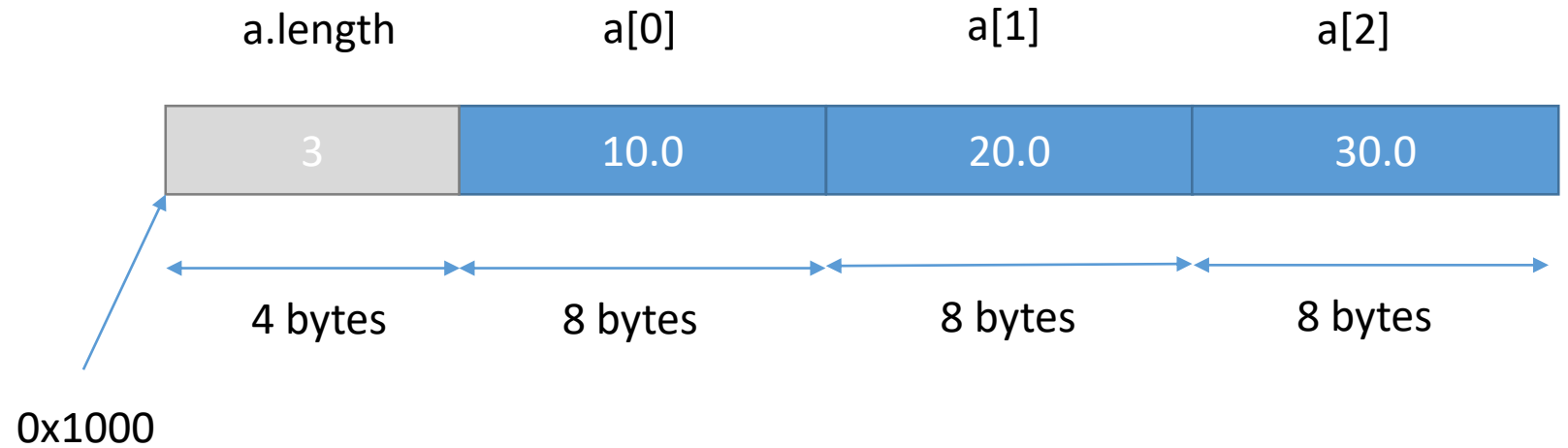
- Definiamo un array di 3 *double* di valore 10, 20 e 30

puntatore
all'array *a*



```
double[] a = new double[3];  
a[0] = 10.0;  
a[1] = 20.0;  
a[2] = 30.0;
```

array *a*



4 byte per puntatore + 28 bytes per array = 32 bytes

Le matrici

- Hanno dimensione fissa definita alla creazione

```
int m[][] = new int[3][3]
```

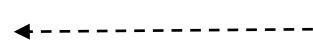
- Elementi omogenei per tipo (tutti int, o float, o boolean, etc.)
- Si accede all'elemento (i,j) -esimo come $a[i][j]$
- Per le righe, l'indice i va da 0 a $n-1$, per le colonne j va da 0 a $n-1$

Le matrici

- Interpretabili come *array bidimensionali*

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

$m[i][j]$



Elemento di m nella:

- riga i -esima
- colonna j -esima

		Indice colonna j			
		0	1	2	3
Indice riga i	0	1	2	3	4
	1	5	6	7	8
	2	9	10	11	12
	3	13	14	15	16
		m			

Le matrici in Java

- **Implementate come array di (puntatori ad) array**

Le matrici in Java

```
int[][] m
```

puntatore
alla matrice *m*



m



4 bytes

Le matrici in Java

```
int[][] m = {  
    {  
        '  
        '  
        '  
    }  
};
```

puntatore
alla matrice *m*



m



4 bytes

Array di puntatori

m.length

0d1000



4 bytes

4 bytes

4 bytes

4 bytes

4 bytes

Le matrici in Java

```
int[][] m = {{10, -2, 0, 15},  
             {  
             }  
};
```

puntatore
alla matrice *m*



m

0d1000



4 bytes

0d1020



m.length



0d1000



4 bytes

4 bytes

4 bytes

4 bytes

4 bytes

Le matrici in Java

```
int[][] m = {{10, -2, 0, 15},  
             null,  
             };
```

puntatore
alla matrice *m*

m

0d1000



4 bytes

0d1020

m.length

0d1000



4 bytes

4 bytes

4 bytes

4 bytes

4 bytes

Le matrici in Java

```
int[][] m = {{10,-2,0,15},  
             null,  
             {-8,12},  
             };
```

puntatore
alla matrice *m*

m

0d1000



4 bytes

0d1040



0d1020



m.length



0d1000



4 bytes

4 bytes

4 bytes

4 bytes

4 bytes

Le matrici in Java

```
int[][] m = {{10,-2,0,15},  
             null,  
             {-8,12},  
             {}};
```

puntatore
alla matrice *m*

m

0d1000



4 bytes

0d1060



0d1040



0d1020



m.length



0d1000



4 bytes

4 bytes

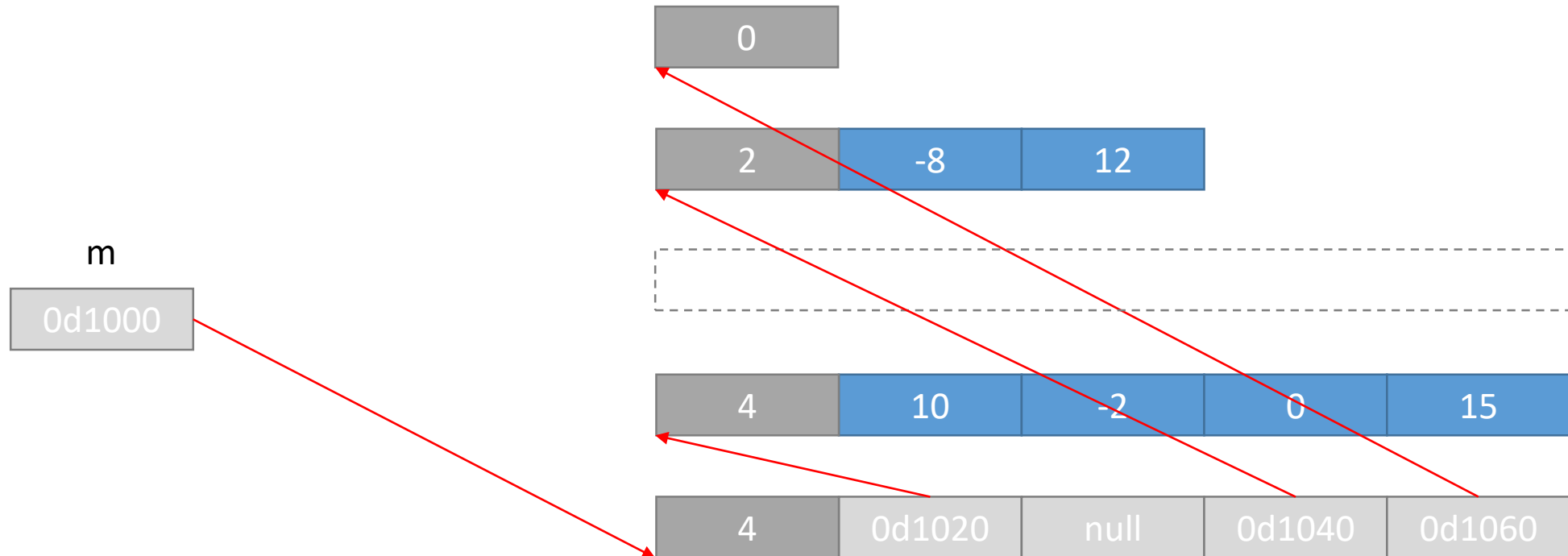
4 bytes

4 bytes

4 bytes

Le matrici in Java

```
int[][] m = {{10,-2,0,15},  
             null,  
             {-8,12},  
             {}};
```



Le matrici in Java

- Implementate come array di (puntatori ad) array
- Numero righe accessibile a runtime come *m.length*

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

m.length
== 4

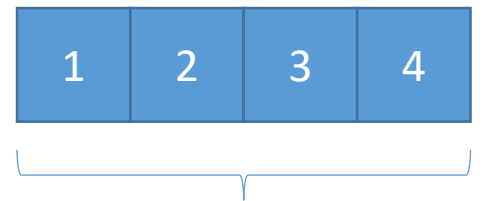
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

m

Le matrici in Java

- Implementate come array di (puntatori ad) array
- Numero righe accessibile a runtime come *m.length*
- Numero colonne riga *i*-esima accessibile a runtime come *m[i].length*

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



`m[0].length == 4`

Le matrici in Java

```
int[][] m = {{10, -2, 0, 15},  
             null,  
             {-8, 12},  
             {}};
```

- Implementate come array di (puntatori ad) array
- Numero righe accessibile a runtime come *m.length*
- Numero colonne riga *i*-esima accessibile a runtime come *m[i].length*
- Supporta righe di lunghezza differente («*ragged*»)
- Le righe possono essere vuote o *null*

10	-2	0	15
----	----	---	----

null

-8	12
----	----

empty

m

Le matrici in Java – regole di accesso

- Ipotesi: voglio accedere all'elemento $m[i][j]$
 1. Verificare puntatore alla matrice m sia valido $m \neq null$
 2. Verificare puntatore alla riga i -esima sia valido $m[i] \neq null$
 3. Accedere a colonna j -esima con indice j in $[0, m[i].length - 1]$

Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];

// Verifica puntatore m valido
if (m != null) {
    // Ciclo esterno con i sulle righe
    for(int i = 0; i < m.length; i = i+1) {
        // Verifica puntatore m[i] valido
        if (m[i] != null) {
            // Ciclo interno con j sulle colonne
            for(int j = 0; j < m[i].length; j = j+1) {
                System.out.print (m[i][j] + " ");
            }
        }
        // Stampo "a capo"
        System.out.println("");
    }
}
```

Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];
```

Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];

// Verifica puntatore m valido
if (m != null) {

}
```


Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];

// Verifica puntatore m valido
if (m != null) {
    // Ciclo esterno con i sulle righe
    for(int i = 0; i < m.length; i = i+1) {

    }
}
```

Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];

// Verifica puntatore m valido
if (m != null) {
    // Ciclo esterno con i sulle righe
    for(int i = 0; i < m.length; i = i+1) {
        // Verifica puntatore m[i] valido
        if (m[i] != null) {

        }

    }
}
```

Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];

// Verifica puntatore m valido
if (m != null) {
    // Ciclo esterno con i sulle righe
    for(int i = 0; i < m.length; i = i+1) {
        // Verifica puntatore m[i] valido
        if (m[i] != null) {
            // Ciclo interno con j sulle colonne
            for(int j = 0; j < m[i].length; j = j+1) {
                System.out.print (m[i][j] + " ");
            }
        }
    }
}
```

Le matrici in Java – stampa matrice (1)

```
int [][] m = new int[...][...];

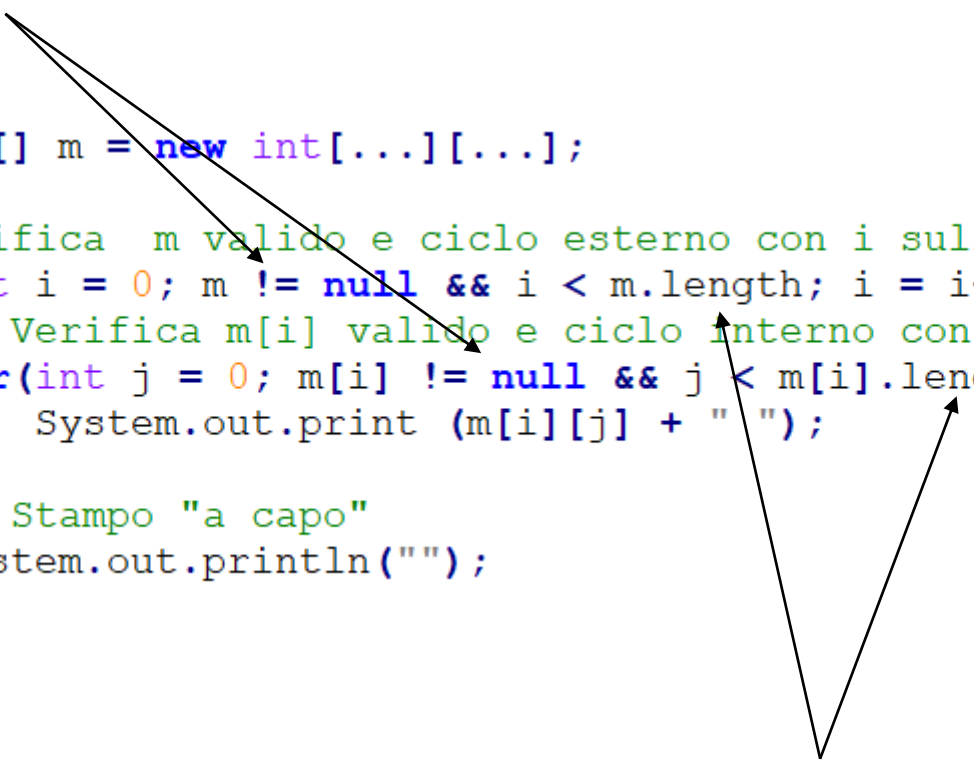
// Verifica puntatore m valido
if (m != null) {
    // Ciclo esterno con i sulle righe
    for(int i = 0; i < m.length; i = i+1) {
        // Verifica puntatore m[i] valido
        if (m[i] != null) {
            // Ciclo interno con j sulle colonne
            for(int j = 0; j < m[i].length; j = j+1) {
                System.out.print (m[i][j] + " ");
            }
        }
        // Stampo "a capo"
        System.out.println("");
    }
}
```

Le matrici in Java – stampa matrice (2)

Prima verifico puntatore non *null*

```
int [][] m = new int[...][...];

// Verifica m valido e ciclo esterno con i sulle righe
for(int i = 0; m != null && i < m.length; i = i+1) {
    // Verifica m[i] valido e ciclo interno con j colonne
    for(int j = 0; m[i] != null && j < m[i].length; j = j+1) {
        System.out.print (m[i][j] + " ");
    }
    // Stampo "a capo"
    System.out.println("");
}
```



Dopo accedo ad attributo length

Le matrici in Java – stampa matrice (2)

Prima accedo ad attributo length

```
int [][] m = null;

// Verifica m valido e ciclo esterno con i sulle righe
for(int i = 0; i < m.length && m != null; i = i+1) {
    // Verifica m[i] valido e ciclo interno con j colonne
    for(int j = 0; j < m[i].length && m[i] != null; j = j+1) {
        System.out.print (m[i][j] + " ");
    }
    // Stampo "a capo"
    System.out.println("");
}
```

Dopo verifico puntatore non *null*



null pointer
exception !

ESERCIZIO Matrici – *stringfy()*

- Si implementi un metodo *stringfy()* che ritorni la rappresentazione String di una matrice di interi, dove:
 - ogni elemento di ogni riga sia seguito da uno spazio tranne l'ultimo elemento
 - l'ultimo elemento di ogni riga sia seguito da *a capo* «\n» tranne l'ultima riga
- Il metodo deve gestire righe *null*
- In caso di matrici *null*, tale metodo ritorni una stringa vuota

ESERCIZIO Matrici – *stringfy()*

```
public static String stringfy(int[][] m) {  
    String s = "";  
    if (m!=null){  
        for (int i=0; i < m.length; i++) {  
            if (m[i]!=null) {  
                for (int j=0; j < m[i].length; j++){  
                    s = s + m[i][j];  
                    if (j < m[i].length -1) {  
                        s = s + " ";  
                    }  
                }  
                s = s + "\n";  
            }  
        }  
    }  
    return s;  
}
```


ESERCIZIO Matrici – *creaTriangolare()*

- Scrivere un metodo *creaTriangolare()* che, dato un intero n , ritorni un (puntatore ad) una matrice di interi quadrata di lato n triangolare alta o bassa a seconda che il secondo parametro *alta* del metodo sia rispettivamente *true* o *false*.
- Gli elementi diversi da zero corrispondano ai numeri interi ≥ 1
- In caso n sia uguale a 0, il metodo ritorni una matrice vuota.

ESERCIZIO Matrici – *creaTriangolare()*

- Esempio per $n=4$

1	2	3	4
0	5	6	7
0	0	8	9
0	0	0	10

Triangolare alta

1	0	0	0
2	3	0	0
4	5	6	0
7	8	9	10

Triangolare bassa

ESERCIZIO Matrici – *creaTriangolare()*

```
public static int[][] creaTriangolare(int n, boolean alta) {
    int[][] ret = new int[n][n];

    int cnt = 1;
    // Ciclo esterno sulle righe con indice i
    for (int i = 0; i < ret.length; i = i+1){
        // Ciclo interno sulle righe con indice j
        for (int j = 0; j < ret[i].length; j = j+1){
            // A seconda del valore di alta, verifichiamo l'opportuna condizione
            if((alta == true && i <= j) || (alta == false && i >= j)) {
                ret[i][j] = cnt;
                cnt = cnt + 1;
            }
        }
    }

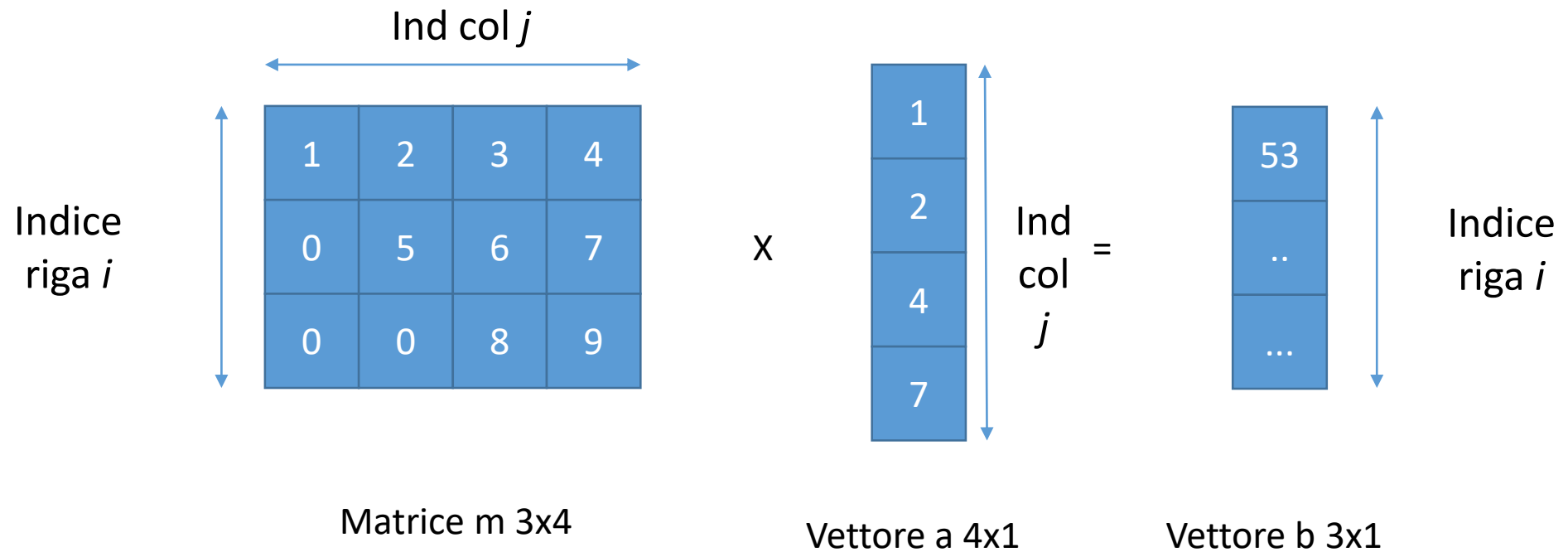
    return ret;
}
```

ESERCIZIO Matrici – *prodMatVet()*

- Si implementi un metodo *prodMatVet()* che, date la matrice m $A \times B$ ed il vettore a $B \times 1$ ($1 \times B$ per semplicità), ne calcoli il prodotto matriciale e ne memorizzi il risultato in un vettore b $A \times 1$ ($1 \times A$ per semplicità).
- Se ne verifichi il funzionamento anche rispetto alla validità di m ed a ed alla correttezza delle dimensioni di m ed a
- In caso di errore si ritorni una matrice vuota (per semplicità, si tralasci il caso di matrici m ragged)

ESERCIZIO Matrici – *prodMatVet()*

- Esempio per m con A==3 righe e B==4 col



ESERCIZIO Matrici – *prodMatVet()*

```
public static int[] prodMatVet(int[][] m, int[] a) {  
  
    int[] ret;  
  
    // Puntatori di m oppure a a null o dimensioni discordanti  
    if (m == null || a == null || m[0].length != a.length) {  
        ret = new int[0];  
    }  
    else {  
        // Vettore ritornato  
        ret = new int[m.length];  
  
        // Ciclo sulle righe di m  
        for (int i = 0; i < m.length && m[i] != null; i = i+1){  
            ret[i] = 0;  
            // Prodotto riga-vettore  
            for (int j = 0; j < m[i].length; j = j+1){  
                ret[i] = ret[i] + (m[i][j] * a[j]);  
            }  
        }  
    }  
  
    return ret;  
}
```

ESERCIZIO Matrici – *mediaRigaPrec()*

- Si implementi un metodo *mediaRigaPrec()* che, data la matrice di int m , ritorni un array di float a tale che la cella i -esima di a contenga la media della riga $(i+1)$ -esima di m .
- Se ne verifichi il funzionamento anche rispetto alla validità di m , in caso di errore si ritorni un puntatore a null

ESERCIZIO Matrici – *mediaRigaPrec()*

```
public static float[] mediaRigaPrec(int[][] m) {  
    float[] a = null;  
    if (m != null && m.length > 0) {  
        // Inizializziamo l 'array a  
        a = new float[m.length - 1];  
        // Cicliamo sulle righe di m (notare inizializzazione i)  
        for (int i = 1; i < m.length; i = i+1){  
            // Inizializzo accumulatore a[i-1]  
            a[i-1] = 0;  
            // Cicliamo sulle colonne di m[i]  
            if (m[i] != null && m[i].length > 0) {  
                for (int j = 0; j < m[i].length; j = j+1){  
                    a[i-1] = a[i-1] + m[i][j];  
                }  
                // Calcolo la media  
                a[i-1] = a[i-1] / (float)m[i].length;  
            }  
        }  
    }  
    return a;  
}
```


ESERCIZIO Matrici – sommaCol()

- Si implementi un metodo `sommaCol()` che, data la matrice di int m , ritorni un array di int a tale che la cella i -esima di a contenga la somma della colonna i -esima di m .
- Se ne verifichi il funzionamento anche rispetto alla validità di m , in caso di errore si ritorni un puntatore a null
- Si faccia ricorso ad un metodo `maxLunR()` per contare gli elementi della riga più lunga di m

ESERCIZIO Matrici – sommaCol()

```
public static int[] sommaCol(int[][] m) {
    int[] a = null;
    if (m != null) {
        // Troviamo la lunghezza massima delle righe di m
        int maxLun = maxLun(m);
        // Allochiamo l'array a
        a = new int[maxLun];
        // Scorrimento sulle colonne di m con indice j
        for(int j = 0; j < a.length; j = j + 1) {
            // Accumulatore elementi colonna j-esima di m
            a[j] = 0;
            // Scorrimento sulle righe di m con indice i
            for(int i = 0; i < m.length; i = i + 1) {
                if (m[i] != null && j < m[i].length) {
                    a[j] = a[j] + m[i][j];
                }
            }
        }
    }
    return a;
}

public static int maxLun(int[][] m) {
    int max = 0;
    if (m != null) {
        for (int i = 0; i < m.length; i = i + 1) {
            if (m[i] != null && max < m[i].length) {
                max = m[i].length;
            }
        }
    }
    return max;
}
```

ESERCIZIO Matrici – *maxLunCov()*

```
public static int maxLunCov(int[][] m, int i) {
    int len = 0;
    // Caso base
    if (i == 0) {
    }
    // Caso generale
    else{
        len = maxLunR(m, i-1);
        if (m[i] != null && len < m[i-1].length) {
            len = m[i].length;
        }
    }
    return len;
}

public static int maxLunCov(int[][] m) {
    int len = 0;
    if (m != null) {
        len = maxLunR(m, m.length -1);
    }
    return len;
}
```

ESERCIZIO Matrici – *maxLunDic()*

```
public static int maxLunDic(int[][] m, int max, int l, int r) {  
    // Caso base  
    if (r == l+1) {  
        if (m[l] == null)  
            return 0;  
        else  
            return m[l].length;  
    }  
    // Caso generale  
    else{  
        int p = (l + r) / 2;  
        System.out.println("p " +p);  
        int len_sx = maxLunDic(m, max, l, p);  
        int len_dx = maxLunDic(m, max, p, r);  
        if (len_sx > max)  
            max = len_sx;  
        if (len_dx > max)  
            max = len_dx;  
  
        return max;  
    }  
}
```