Array in Java

Emilio Di Giacomo e Walter Didimo

Gli array

- Anche in Java, come in C, esistono gli array
- Un array è una sequenza di variabili:
  - tutte le variabili di un array hanno lo stesso tipo di dato (il tipo dell'array), e si chiamano anche elementi dell'array
  - ogni variabile ha associato un indice (numero intero non negativo)
  - la dimensione (o lunghezza) dell'array è il numero delle sue variabili

Array in Java

- In Java un array è un oggetto
- Rispetto ad un oggetto “standard” presenta alcune differenze:
  - non è un’istanza di una classe predefinita
  - non possiede metodi

Array in Java

- Come gli altri oggetti viene creato con l'operatore *new*
  - all'atto della creazione va specificato sia il tipo che la dimensione

    new ⟨*tipo di dato*⟩ [⟨*dimensione*⟩]
  - Esempio: creazione di un array di tipo *String* di dimensione *5*:

    new String [5]

Array in Java

- Come per la creazione di oggetti l'operatore new restituisce un riferimento all'array creato
- Tale riferimento deve essere salvato in un'opportuna variabile per poter poi utilizzare l'array
- Quale deve essere il tipo di questa variabile?

Array in Java

- Per ogni tipo *T* è automaticamente definito il tipo *T[]* chiamato array di tipo T
- Una variabile di tipo *T[]* può memorizzare un riferimento ad un array il cui tipo è *T*
- Esempio

  String[] s = new String[5];

- L'istruzione precedente crea un array di stringhe di dimensione *5* e memorizza un riferimento ad esso in una variabile *s*

Accesso agli elementi di un array

- Un array di tipo *T* di dimensione *n* è costituito da *n* variabili di tipo *T*
- Ognuna di tali variabili può memorizzare un dato di tipo *T* e può essere acceduta (in lettura o scrittura) indipendentemente dalle altre
- L'accesso ad una certa variabile avviene specificando un indice
  - Ogni elemento di un array è infatti associato ad un indice compreso tra *0* e *n-1*

⟨*riferimento array*⟩ [⟨*espressione*⟩]

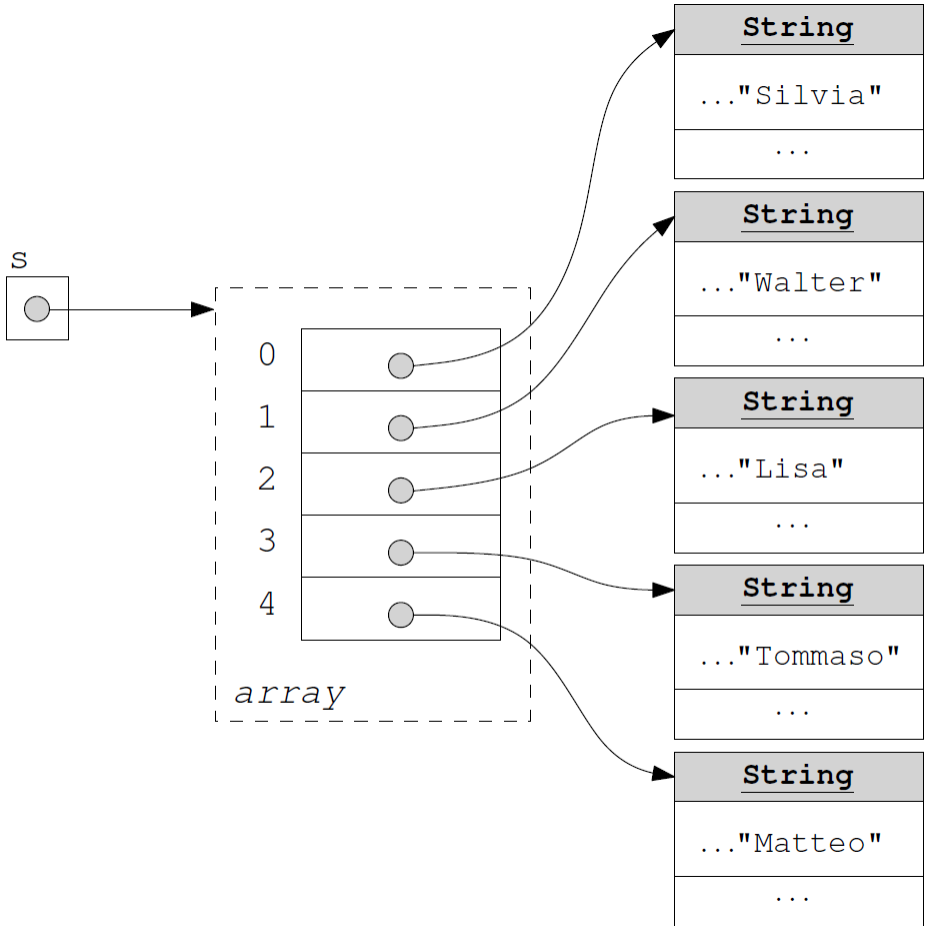
Accesso agli elementi di un array

- Esempio:

```
String[] s = new String[5];
s[0] = "Silvia";
s[1] = "Walter";
s[2] = "Lisa";
s[3] = "Tommaso";
s[4] = "Matteo";
```

- Il brano precedente crea un array di stringhe di dimensione 5 e assegna ad ogni elemento di tale array un riferimento ad una stringa

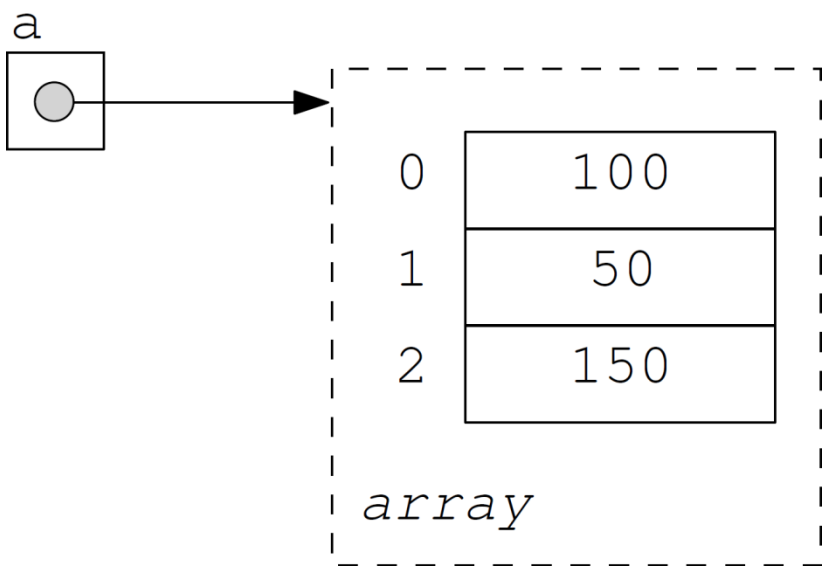
Schema di un array



Array di tipi primitivi

- Nell'esempio precedente abbiamo fatto riferimento ad un array di oggetti
- È possibile creare anche array di tipi primitivi

```
int[] a = new int[3];
a[0] = 100;
a[1] = 50;
a[2] = a[0]+a[1];
```



L’attributo length

- Gli array non hanno metodi, ma ogni array ha un attributo pubblico di nome *length*
- Tale attributo memorizza la lunghezza dell’array
- L’attributo è di sola lettura
  - non è possibile cambiare la dimensione di un array una volta creato
- A cosa serve l’attributo *length*?
  - ad esempio a conoscere la dimensione di un array ricevuto come parametro di un metodo

L’attributo length: esempio

```
public int sommaElementi(int[] a){
   int somma = 0;
   for (int i = 0; i<a.length; i++)
       somma += a[i];
   return somma;
}
```

- Il metodo precedente riceve un array di interi come parametro e restituisce la somma di tutti i suoi elementi
- Poiché l’array è creato esternamente al metodo l’unico modo per conoscere la sua lunghezza è tramite l’attributo *length*

Ancora sugli indici

- Abbiamo visto che gli elementi di un array a sono associati agli indici da *0* ad *a.length-1*
- Che cosa succede se tentiamo di accedere ad una posizione non valida, cioè ad un indice minore di *0* o maggiore di *a.length-1*?
- In fase di compilazione non si ha nessun problema
  - l'espressione usata come indice non viene valutata ma viene soltanto controllato che il tipo sia *int*
- In fase di esecuzione si ottiene un errore del tipo *ArrayIndexOutOfBoundException*

Inizializzazione di Array

- In Java quando un array viene creato i suoi elementi vengono inizializzati con il valore di default del loro tipo
  - ad esempio gli elementi di un array di interi vengono inizializzati con il valore *0*
  - mentre gli elementi di un array di stringhe vengono inizializzati con il valore *null*
- Ovviamente è possibile (e auspicabile) inizializzare esplicitamente gli elementi di un array

Inizializzazione di Array

- In Java è anche possibile inizializzare gli elementi di un array contestualmente alla sua creazione
  - in questo caso la creazione avviene con un meccanismo diverso dall'uso del *new*
- Esempio

String[] s={"Silvia", "Walter", "Lisa", "Tommaso", "Matteo"};

- *{"Silvia", "Walter", "Lisa", "Tommaso", "Matteo"}* è un letterale array

Letterali array

- Se *v1*, *v2*, ..., *vk* sono letterali di un tipo *T*, *{v1, v2, …, vk}* è un letterale array di tipo *T*
- I letterali array possono essere utilizzati per creare array ed inizializzarli all'atto della creazione
- L'assegnazione di un letterale array di tipo *T* ad una variabile di tipo *T[]* deve avvenire contestualmente alla definizione della variabile

```
int[] a={1,2,3} //OK
int[] b;
b={4,5,6} // Errore!!
```

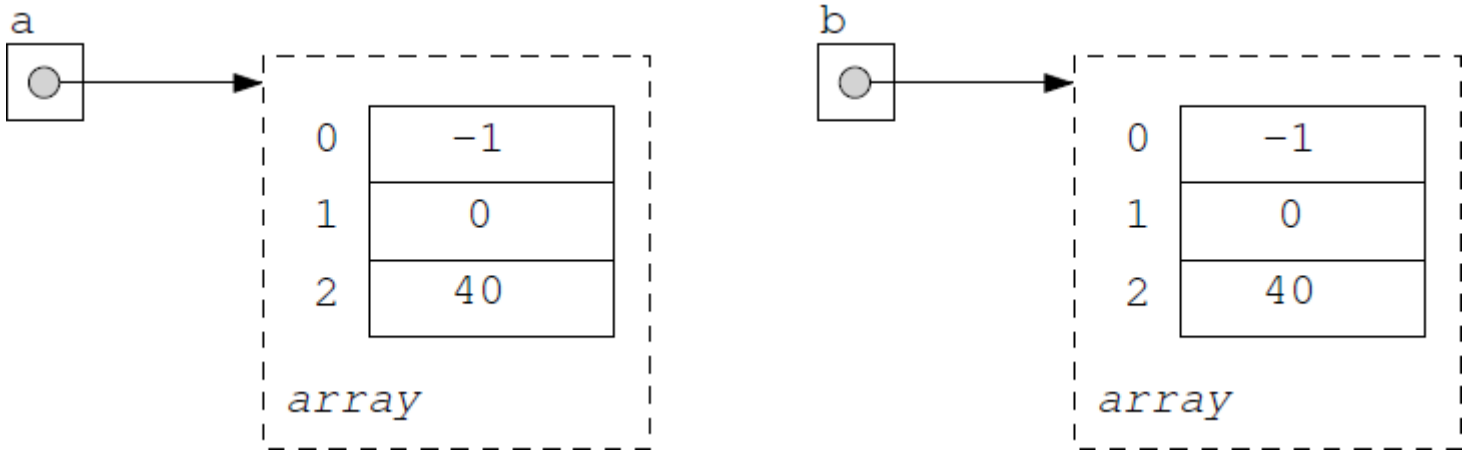
Copia di array

- Un'operazione piuttosto ricorrente nei programmi è la copia di un array
- Copiare un array *a* significa creare un nuovo array *b* i cui elementi hanno lo stesso valore di quelli di *a*
- La copia avviene in due passi:
  - si crea l'array *b* con la stessa dimensione di *a*
  - si copiano gli elementi di *b* uno alla volta in *a*

Copia di array

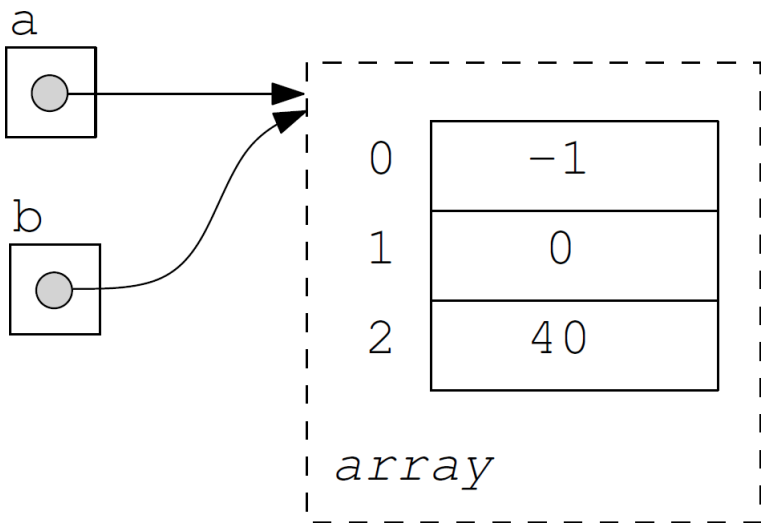
- Si supponga che *a* sia un array di tipo *int*:

```
int [] b = new int [a.length];
for (int i=0; i<a.length; i++)
        b[i] = a[i];
```



Copia di array

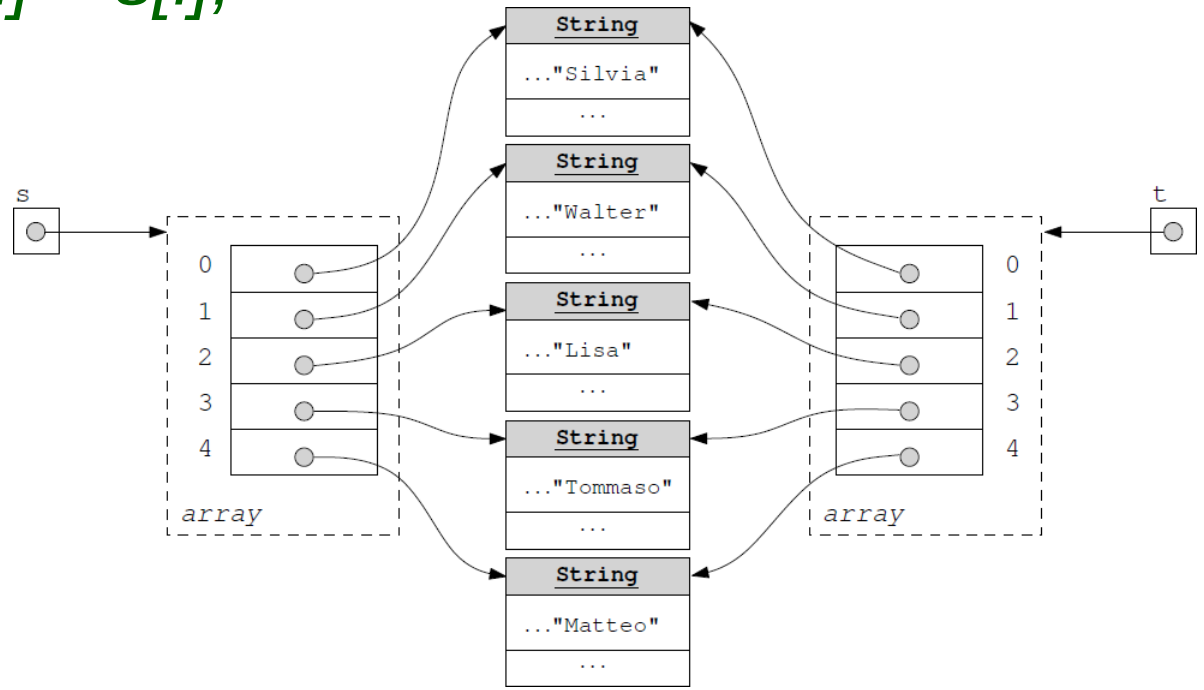
- Nota: l'istruzione *b=a* NON effettua la copia dell'array
- In questo caso si copia soltanto il riferimento all'array



Copia di array

- Si supponga che *s* sia un array di stringhe

```
String [] t = new String [s.length];
for (int i=0; i<s.length; i++)
    t[i] = s[i];
```



Esempi di uso degli array

- Mostreremo adesso un esempio di uso degli array in Java
- Vogliamo scrivere una classe *ElencoPrenotazioni*:
  - un oggetto istanza di questa classe permette di rappresentare le prenotazioni relative ad un volo;
  - ogni prenotazione consiste del nome e cognome del viaggiatore e del numero di posto ad egli assegnato;
  - l'elenco ha un numero massimo di prenotazioni (apri al numero di posti sull'aereo). Tale numero viene impostato all’atto della creazione dell'elenco.

Esempi di uso degli array

- La classe *ElencoPrenotazioni* sarà dotata dei seguenti costruttori e metodi:
  - *public ElencoPrenotazioni(int dim)*: crea un elenco che può contenere al massimo *dim* prenotazioni
  - *public boolean inserisci(String n, String c, int p)*: se l'aereo non è pieno, inserisce nell'elenco una nuova prenotazione con nome *n*, cognome *c* e numero di posto *p*; poi restituisce *true*. Se l'aereo è pieno restituisce *false*
  - *public int numeroPosto(String n, String c)*: restituisce il numero di posto della persona con nome *n* e cognome *c* se questa è presente nell'elenco; altrimenti restituisce *-1*
  - *public String toString()*: restituisce una descrizione testuale dell'elenco completo delle prenotazioni

La classe ElencoPrenotazioni

- Quali sono gli attributi della classe *ElencoPrenotazioni*?
- Per memorizzare i vari iscritti useremo tre array:
  - l'array *nome* (di tipo *String*) per memorizzare i nomi
  - l'array *cognome* (di tipo *String*) per memorizzare i cognomi
  - l'array *posto* (di tipo *int*) per memorizzare i numeri di posto
  - Gli elementi con lo stesso indice corrisponderanno alla stessa prenotazione

La classe ElencoPrenotazioni

- Useremo inoltre una variabile intera *count* per tener traccia del numero di prenotazioni nell'elenco:
  - in ogni momento le posizioni da *0* a *count-1* dei tre array conterranno dati validi
  - le posizioni da *count* in poi saranno disponibili per inserire nuove prenotazioni

ElencoIscritti: scheletro

```
public class ElencoPrenotazioni{
    private String[] nome;
    private String[] cognome;
    private int[] posto;
    private int count;

    /* costruttore: crea un elenco che può contenere al massimo dim prenotazioni
    */
    public ElencoPrenotazioni(int dim){...}


    /* se l'aereo non è pieno, inserisce nell'elenco una nuova prenotazione con
    nome n, cognome c e numero di posto p; poi restituisce true. Se l'aereo è
    pieno restituisce false. */
    public boolean inserisci(String n, String c, int p){...}

    /* restituisce il numero di posto della persona con nome n e cognome c se
    questa è presente nell'elenco; altrimenti restituisce -1 */
    public int numeroPosto(String n, String c){...}

    /* restituisce una descrizione testuale dell'elenco completo delle
    prenotazioni */
    public String toString(){...}
}
```

Costruttore

- Il costruttore deve creare tre array e memorizzarne i riferimenti nelle variabili di istanza *nome*, *cognome* e *posto*:
  - i tre array avranno tutti dimensione *dim*
- Inoltre deve inizializzare la variabile *count* a *0*

```
public ElencoPrenotazioni(int dim){
    this.nome = new String[dim];
    this.cognome = new String[dim];
    this.posto = new int[dim];
    this.count = 0;
}
```

Il metodo inserisci

- *public boolean inserisci(String n, String c, int p)*
- Come abbiamo detto l'idea è di far in modo che in ogni momento le posizioni da *0* a *count-1* contengano i dati degli iscritti, mentre quelle da *count* in poi siano disponibili
  - ne segue che il nuovo iscritto va inserito in posizione *count*, che va poi incrementato
- Occorre preliminarmente verificare che ci siano posizioni disponibili

Il metodo inserisci

```
public boolean inserisci(String n, String c, int p){
      boolean inserito = false;
      if(this.count<this.nome.length){
            this.nome[this.count] = n;
            this.cognome[this.count] = c;
            this.posto[this.count] = p;
            inserito = true;
            this.count++;
      }
      return inserito;
  }
```

Il metodo numeroPosto

- *public int numeroPosto(String n, String c)*
- Scandiamo le posizioni valide degli array *nome* e *cognome* alla ricerca delle stringhe *n* e *c*
- Se gli elementi correnti dei due array sono uguali ai valori cercati viene memorizzato il posto e la ricerca termina; il metodo restituisce *il* numero di posto
- Se la scansione termina senza avere mai trovato i valori cercati il metodo restituisce *-1*

Il metodo numeroPosto

```
public int numeroPosto(String n, String c){
   int posto = -1;
   int i=0;
   while(i<this.count && posto == -1){
        if(this.nome[i].equals(n) && this.cognome[i].equals(c))
               posto=this.posto[i];
        i++;
   }
   return posto;
}
```

Il metodo toString

```
public String toString(){
	String s="";
	for(int i=0;i<this.count;i++){
		s+=(i+1)+") "+this.cognome[i]+" "
		+this.nome[i]+" posto: "
		+this.posto[i]+"\n";

	}

	return s;
}
```

Il parametro del metodo main

- Siamo ora in grado di capire il parametro che compare nell'intestazione del metodo *main*

```
public static void main(String[] args)
```

- Tale parametro è un array di stringhe
  - convenzionalmente viene chiamato *args*
- Ma a cosa serve tale parametro? Chi passa al *main* il paramero attuale al momento della chiamata?

Il parametro del metodo main

- Il metodo *main* viene eseguito al momento dell'avvio del programma a seguito del comando *java C*
  - assumendo che *C* sia il nome della classe in cui è definito il *main*
- All'avvio del programma è possibile passare al *main* come parametro una sequenza di stringhe che verranno salvato nell'array *args*

Il parametro del metodo main

- In particolare, scrivendo

  java C ⟨*stringa* 1⟩ ⟨*stringa* 2⟩ ... ⟨*stringa k*⟩
- viene invocato il metodo *main* della classe *C* passandogli come parametro un array *args* di dimensione *k* tale che:
  - *args[0]* è pari a ⟨*stringa* 1⟩, *args[1]* è pari a ⟨*stringa* 2⟩,..., *args[k-1]* è pari a ⟨*stringa k*⟩
- Quando non si passa nessuna stringa, *args* ha dimensione *0*
- Le stringhe ⟨*stringa* 1⟩ ⟨*stringa* 2⟩ ... ⟨*stringa k*⟩ vengono chiamate parametri della linea di comando

Esempio

- Immaginiamo di voler scrivere un programma che effettui la somma di due numeri, passatigli in input al momento della sua esecuzione
- Se ad esempio chiamassimo *Somma* la classe che realizza questo programma, potremmo scrivere:

  java Somma 150 200
- ricevendo in risposta il messaggio *350*

Esempio

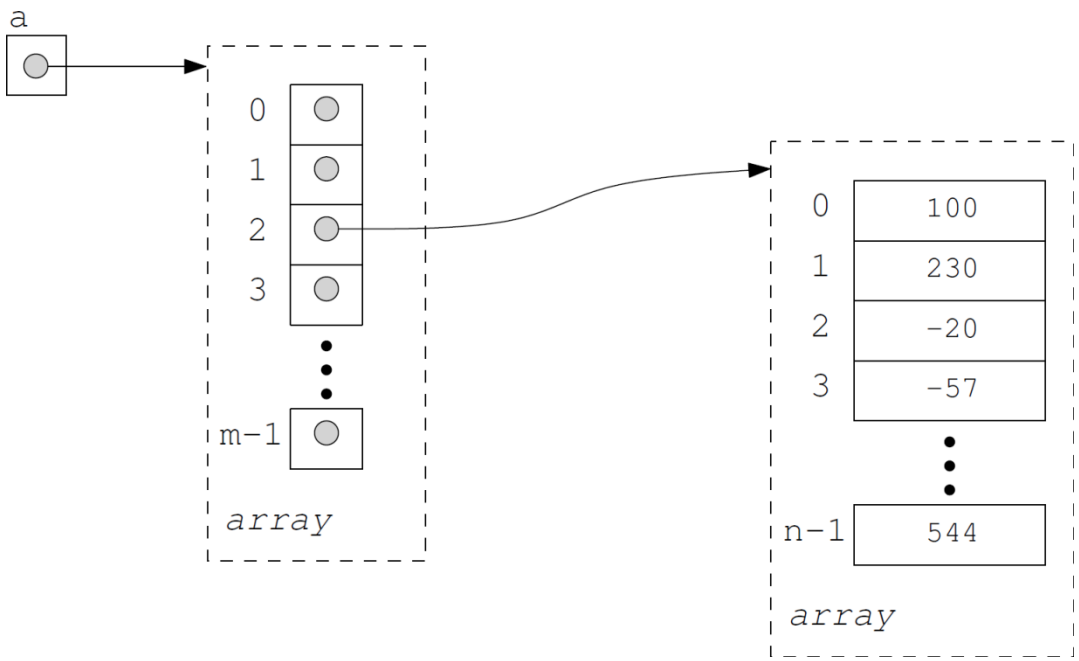
```
public class Somma{
  public static void main(String[] args){
    if (args.length<2)
      System.out.println("Num. param. errato!");
    else{
      Integer n1 = Integer.valueOf(args[0]);
      Integer n2 = Integer.valueOf(args[1]);
      System.out.println(n1.intValue()+n2.intValue());
    }
  }
}
```

Array di array

- Gli array che abbiamo visto finora sono caratterizzati dall'utilizzo di un solo indice per far riferimento agli elementi dell'array
- Questo tipo di array si chiamano anche array unidimensionali
- Visto che il tipo degli elementi di un array può essere qualunque, esso può essere a sua volta un tipo array

Array di array

- Possiamo ad esempio definire un array i cui elementi sono array di interi



Array di array

- Un array i cui elementi sono a loro volta degli array si chiama array di array o array bidimensionale
  - i suoi elementi possono infatti essere acceduti tramite due indici
- E possibile anche definire array di array di array, oppure array di array di array di array, ecc.
  - raramente si ha bisogno di array con più di due indici

Array bidimensionali matriciali

- Gli array bidimensionali vengono usati prevalentemente per modellare matrici
- Il loro potere espressivo è tuttavia superiore a quello di una matrice (vedremo più avanti)
- Per ora ci concentriamo su array bidimensionali matriciali, cioè “tabelle” con *m* righe e *n* colonne
- Un array bidimensionale matriciale di tipo *T* con *m* righe ed *n* colonne può essere creato con la seguente istruzione

  new T[m][n]

Array bidimensionali matriciali

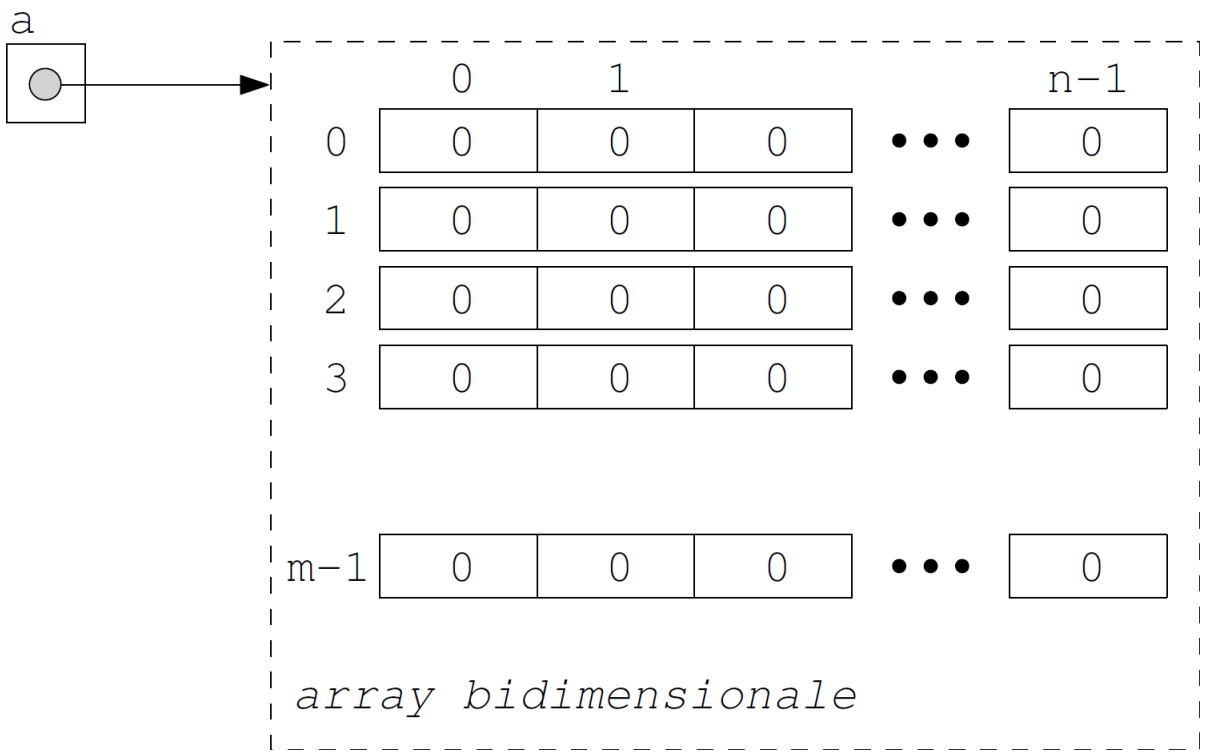
- Il tipo dell'array creato è array di array di tipo T oppure array bidimensionale di tipo T
- Tale tipo di dato si indica con *T[][]*
  - possiamo quindi definire una variabile di questo tipo per memorizzare il riferimento ad una array bididmensionale di tipo *T*
- Gli elementi dell'array vengono inizializzati con il valore di default del tipo *T*

Array bidimensionali matriciali

int [][] a=new int [m][n];

- Ad esempio, l'istruzione precedente crea un array bidimensionale di interi con *m* righe e *n* colonne e ne memorizza il riferimento nella variabile *a*
- Gli elementi creati saranno inizializzati con il valore *0*
- La situazione è mostrata nella prossima slide
  - nel seguito rappresenteremo gli array bidimensionali come se fossero matrici

Array bidim.: rappresent. grafica

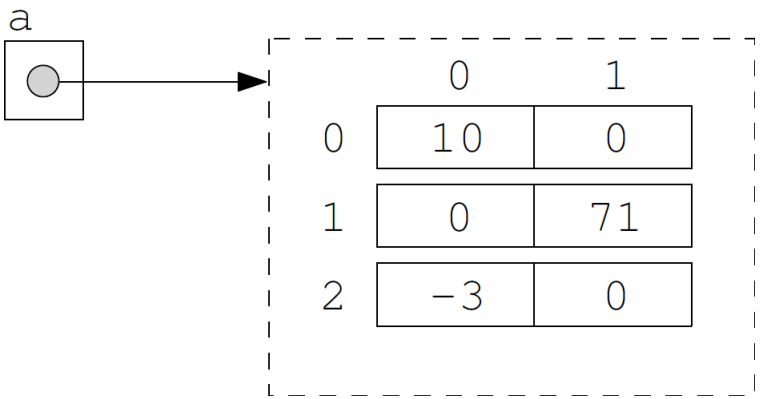


Accesso agli elementi

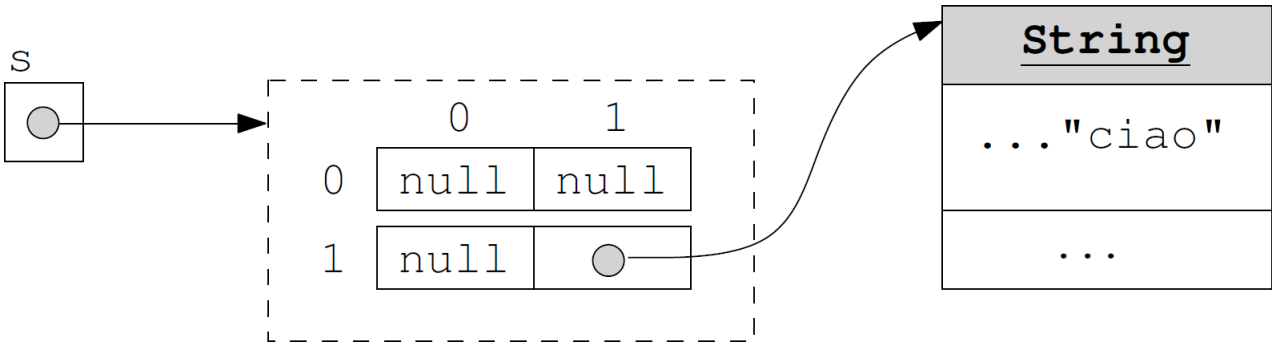
- Per accedere agli elementi di un array bidimensionale sono necessari due indici detti rispettivamente indice di riga e indice di colonna
- Ad esempio con *a[i][j]* si denota l'elemento di riga *i* e colonna *j* dell'array bidimensionale *a*
- Più precisamente:
  - *a[i]* denota l'array monodimensionale corrispondente alla riga i-esima
  - *a[i][j]* denota il j-esimo elemento dell'array *a[i]*

Esempio

```
int[][] a = new int[3][2];
a[0][0] = 10;
a[1][1] = 71;
a[2][0] = -3;
```



```
String[][] s = new String[2][2];
s[1][1] = "ciao";
```



Accesso agli elementi

- Nota bene: *a[i]* è a tutti gli effetti un riferimento ad un array monodimensionale
- è quindi possibile, ad esempio, scrivere *a[i].length* per conoscere la sua dimensione
- oppure passare *a[i]* ad un metodo che richiede come parametro un array monodimensionale dello stesso tipo di *a*

Letterali array bidimensionale

- Anche gli array bidimensionali possono essere creati attraverso letterali array
- Esempio:

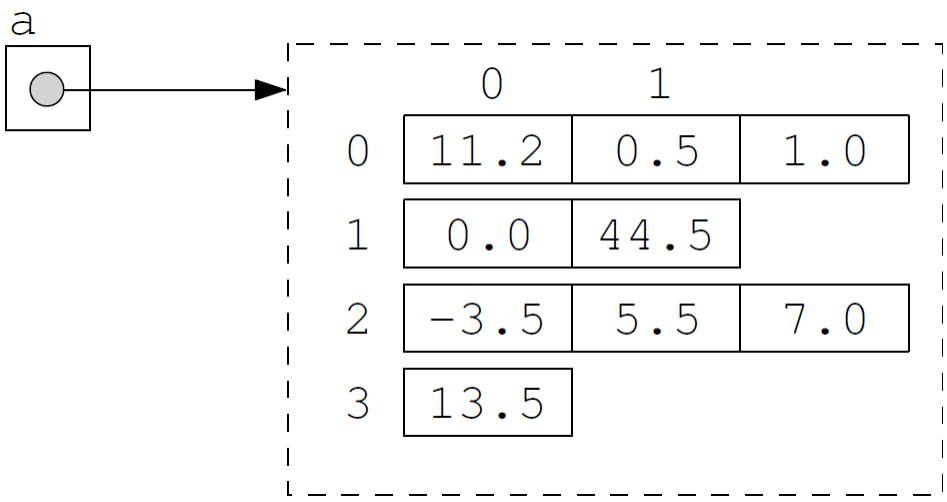
  int[][] a={{10, 0}, {0, 71}, {-3, 0}};
- *{{10, 0}, {0, 71}, {-3, 0}}* è un letterale array bidimensionale

Array bidimensionali non matriciali

- Abbiamo detto che gli array bidimensionali hanno un potere espressivo maggiore delle matrici
- Visto che le righe di un array bidimensionale sono a tutti gli effetti array monodimensionali, nulla vieta che tali array abbiano dimensioni diverse

Esempio

```
double[][] a = { {11.2, 0.5, 1.0}, {0.0, 44.5},
                 {-3.5, 5.5, 7.0}, {13.5} };
```



Esempio

- Lo stesso array può essere creato come segue:

```
double[][] a = new double[4][];
a[0] = new double[3];
a[1] = new double[2];
a[2] = new double[3];
a[3] = new double[1];
a[0][0] = 11.2;
a[0][1] = 0.5;
a[0][2] = 1.0;
a[1][0] = 0.0;
a[1][1] = 44.5;
a[2][0] = -3.5;
a[2][1] = 5.5;
a[2][2] = 7.0;
a[3][0] = 13.5;
```

Viene specificata solo una dimensione (numero di righe)

Creazione esplicita delle singole righe

Visitare gli elem. di un array bidim.

- Gli elementi di un array monodimensionale vengono tipicamente scanditi uno dopo l'altro tramite un ciclo
- Come possiamo scandire gli elementi di un array bidimensionale?
- Tipicamente essi vengono scandite per righe (dalla prima all'ultima) e all'interno di ogni riga per colonne (dalla prima all'ultima)
  - ciò può essere fatto con due cicli annidati

Visitare gli elem. di un array bidim.

```
public static void stampaElementi(double[][] a){
  for (int i = 0; i<a.length; i++)
    for (int j = 0; j<a[i].length; j++){
      System.out.print("Elem. in posiz. "+"("+i+","+j+"): ");
      System.out.println(a[i][j]);
    }
}
```

Visitare gli elem. di un array bidim.

- Nota: *a.length* denota la lunghezza dell'array referenziato da *a*, quindi il numero di righe
- *a[i].length* denota la lunghezza dell'array referenziato da *a[i]*, quindi il numero di elementi sulla riga *i*-esima
- Il codice precedente funziona sia con array matriciali che con array non matriciali

Es. di uso di array bidimensionali

- Vogliamo definire una classe di nome *Matrice* i cui oggetti rappresentano matrici di numeri reali
- La classe deve metterci a disposizione metodi per effettuare alcune operazioni tra matrici

Esempio: la classe Matrice

- La classe *Matrice* avrà i seguenti metodi e costruttori:
  - *Matrice(double[][] m)*: costruttore per creare un oggetto *Matrice*; l'array bidimensionale *m* specifica quale sia la matrice che l'oggetto creato dovrà rappresentare
  - *public boolean diagonale()*: restituisce *true* se la matrice rappresentata dall'oggetto ricevente (*this*) è diagonale. Se la matrice non è quadrata restituisce *false*.
  - *public Matrice trasposta()*: restituisce un nuovo oggetto *Matrice* la cui matrice rappresentata è la trasposta della matrice rappresentata dall'oggetto ricevente (*this*)

Esempio: la classe Matrice

- La classe *Matrice* avrà i seguenti metodi e costruttori:
  - *public static Matrice matriceDiagonaleABlocchi(Matrice a, Matrice b)*: date due matrici quadrate *a* e *b* restituisce un nuovo oggetto *Matrice* che rappresenta una matrice diagonale a blocchi i cui blocchi sono *a* e *b*. Se le matrici non sono quadrate restituisce *null*.
  - *String toString()*: restituisce una descrizione della matrice rappresentata dall'oggetto ricevente (*this*).

La classe Matrice: scheletro

```
public class Matrice{

   private double[][] mat;

   public Matrice(double[][] m){...}

   public boolean diagonale(){...}

   public Matrice trasposta(){...}

   public static Matrice matriceDiagonaleABlocchi(Matrice
   a, Matrice b){...}

   public String toString(){...}


}
```

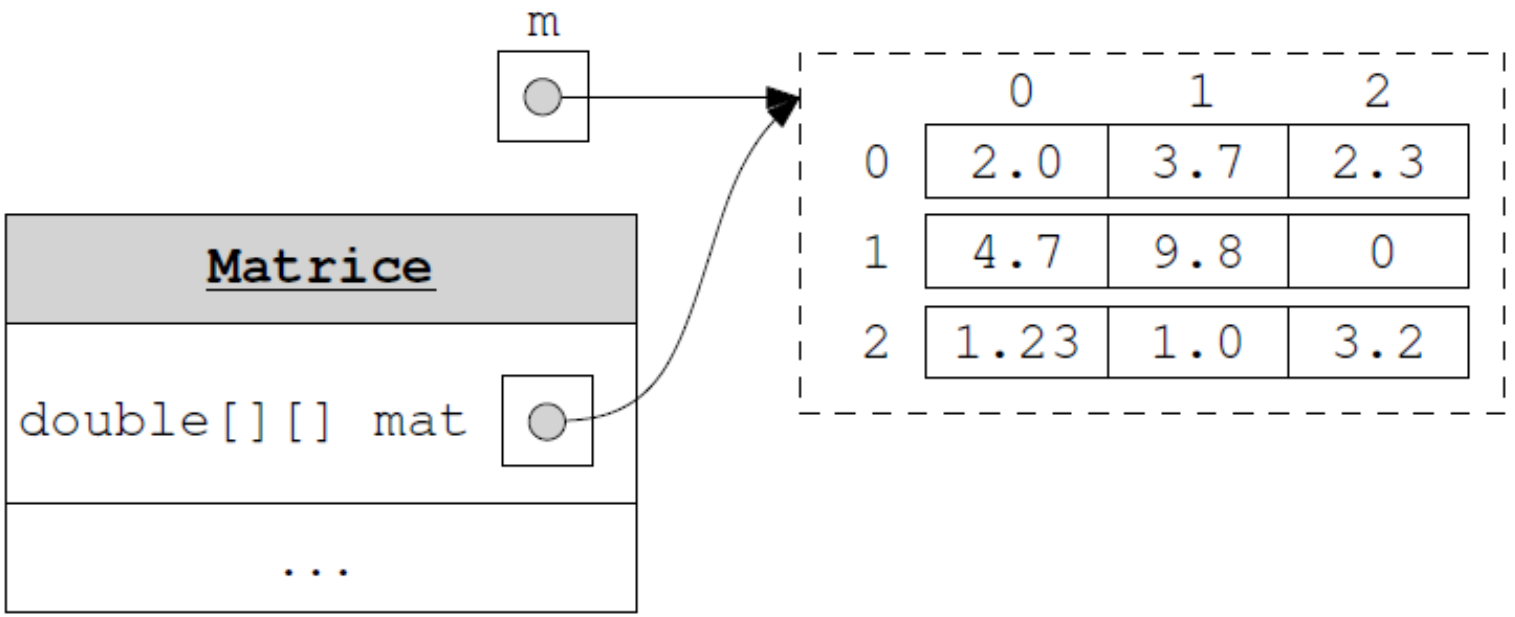
Costruttore

- Il costruttore deve inizializzare il valore della variabile di istanza *mat*, in modo che essa memorizzi il riferimento ad un array bidimensionale di *double* equivalente a quello passato come parametro
- Una prima implementazione:

```
public Matrice(double[][] m){
    this.mat = m;
}
```

Costruttore

- Si noti che nel caso precedente si copia in *mat* il riferimento all'array memorizzato in *m*



Costruttore

- Uno svantaggio di tale soluzione è che l'array referenziato da *mat* può essere modificato esternamente alla classe *Matrice*
- Ciò va contro il principio dell'incapsulamento in base al quale gli attributi di un oggetto dovrebbero essere privati e non modificabili esternamente alla classe che li definisce

Costruttore

- Una soluzione alternativa

```
public Matrice(double[][] m){
   this.mat=new double[m.length][m[0].length];

   for(int i=0;i<this.mat.length;i++)
       for(int j=0;j<this.mat[i].length;j++)
             this.mat[i][j]=m[i][j];
}
```

Il metodo diagonale

- Controlliamo innanzi tutto che la matrice sia quadrata; se non lo è restituiamo false
- Se la matrice è quadrata scandiamo tutti gli elementi se ne esiste uno che non si trova sulla diagonale ed è diverso da 0 la matrice non è diagonale

Il metodo diagonale

```
public boolean diagonale(){
   boolean diagonale=false;
   if(this.mat.length==this.mat[0].length){
        diagonale=true;
        int i=0;
        while(i<this.mat.length && diagonale){
                int j=0;
                while(j<this.mat[i].length && diagonale){
                        if(i!=j && this.mat[i][j]!=0)
                                diagonale=false;
                        j++;
                }
                i++;
        }
   }
   return diagonale;
}
```

Il metodo trasposta

- Creiamo prima un array bidimensionale *m* il cui numero di righe è pari al numero di colonne di *this.mat* e il cui numero di colonne è pari al numero di righe di *this.mat*
- Ogni elemento *m[i][j]* viene posto uguale a *this.mat[j][i]*
- Infine viene creato un oggetto *Matrice* usando *m* come parametro del costruttore

Il metodo trasposta

```
public Matrice trasposta(){
   double[][] m=new double[mat[0].length][mat.length];

   for(int i=0;i<m.length;i++)
        for(int j=0;j<m[i].length;j++)
               m[i][j]=this.mat[j][i];

   return new Matrice(m);
}
```

Il met. matriceDiagonaleABlocchi

- L’approccio è simile a quello del metodo precedente
- Il risultato viene prima memorizzato in un array bidimensionale *m* di opportune dimensioni
- Poi viene creato l'oggetto *Matrice* da restituire

Il met. matriceDiagonaleABlocchi

```
public static Matrice matriceDiagonaleABlocchi(Matrice a, Matrice b){
    Matrice risultato=null;

    int m1=a.mat.length;
    int m2=b.mat.length;

    int n1=a.mat[0].length;
    int n2=b.mat[0].length;

    if(m1==n1 && m2==n2){
          double[][] m=new double[m1+m2][n1+n2];
          for(int i=0;i<m.length;i++)
                    for(int j=0;j<m[i].length;j++)
                              if(i<m1 && j<n1)
                                        m[i][j]=a.mat[i][j];
                              else if(i>=m1 && j>=n1)
                                        m[i][j]=b.mat[i-m1][j-n1];
                              else
                                        m[i][j]=0;
          risultato=new Matrice(m);
    }

    return risultato;
}
```

Il metodo toString

```
public String toString(){
   String s="";
   for(int i=0;i<this.mat.length;i++){
        for(int j=0;j<this.mat[i].length;j++)
               s+=this.mat[i][j]+"\t";
        s+="\n";
   }
   return s;
}
```