Array

Emilio Di Giacomo e Walter Didimo

Limite delle variabili

- L'utilizzo di variabili "semplici" come quelle viste fino ad ora non è sufficiente per risolvere problemi in cui si debbano gestire collezioni di dati di dimensione non nota a priori
- Consideriamo ad esempio il seguente problema:
  - vogliamo far inserire all'utente una sequenza di stringhe e poi un numero intero positivo *n*
  - vogliamo visualizzare all'utente le sole stringhe inserite che hanno lunghezza maggiore di *n*

Limite delle variabili

- Il problema può avere due varianti:
  - la lunghezza della sequenza da fare inserire all'utente è fissata prima di scrivere il programma
  - la lunghezza della sequenza da fare inserire all'utente è scelta dall'utente stesso (e quindi non nota all'atto della scrittura del codice)
- Consideriamo inizialmente la prima variante e supponiamo che la lunghezza della sequenza sia quattro:
  - in questo caso potremmo risolvere il problema con gli strumenti che conosciamo

Soluzione 1

```
import fond.io.*;

public class SequenzaStringheDimensioneFissa1{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    OutputWindow out = new OutputWindow();

    // inserimento stringhe
    String s1 = in.readString("Inserire stringa 1");
    String s2 = in.readString("Inserire stringa 2");
    String s3 = in.readString("Inserire stringa 3");
    String s4 = in.readString("Inserire stringa 4");

    // inserimento intero positivo
    int k = in.readInt("Inserire un intero positivo");

    ...
}
```

Soluzione 1

```
import fond.io.*;

public class SequenzaStringheDimensioneFissa1{
  public static void main(String[] args){
    ...

    // stampa stringhe di lunghezza maggiore di k
    out.writeln("Stringhe piu' lunghe di " + k + ":");
    if (s1.length()>k)
      out.writeln(s1);
    if (s2.length()>k)
      out.writeln(s2);
    if (s3.length()>k)
      out.writeln(s3);
    if (s4.length()>k)
      out.writeln(s4);
    }
}
```

Limiti della soluzione vista

- La soluzione illustrata necessita di definire 4 variabili distinte per memorizzare le stringhe della sequenza
- Inoltre, la soluzione illustrata deve ripetere le stesse cose (lettura, verifica lunghezza e stampa) per ognuna delle stringhe inserite
- Cosa succede se voglio fissare la lunghezza della sequenza a 100, o addirittura a 100.000?
  - è molto dispendioso scrivere un codice in cui definisco esplicitamente 100.000 variabili ed in cui ripeto 100.000 istruzioni “quasi” uguali!!

Variante con lunghezza non fissata

- Nel caso in cui la lunghezza della sequenza non fosse fissata nel programma, ma decisa dall'utente, sarebbe addirittura impossibile scrivere il programma usando i soli strumenti che abbiamo
  - non sappiamo quante variabili bisogna usare
  - non sappiamo quante volte ripetere le operazioni di lettura, verifica della lunghezza e stampa

Gli array

- Per risolvere problemi come quello precedente, i linguaggi di programmazione forniscono una struttura dati nota come 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*⟩]

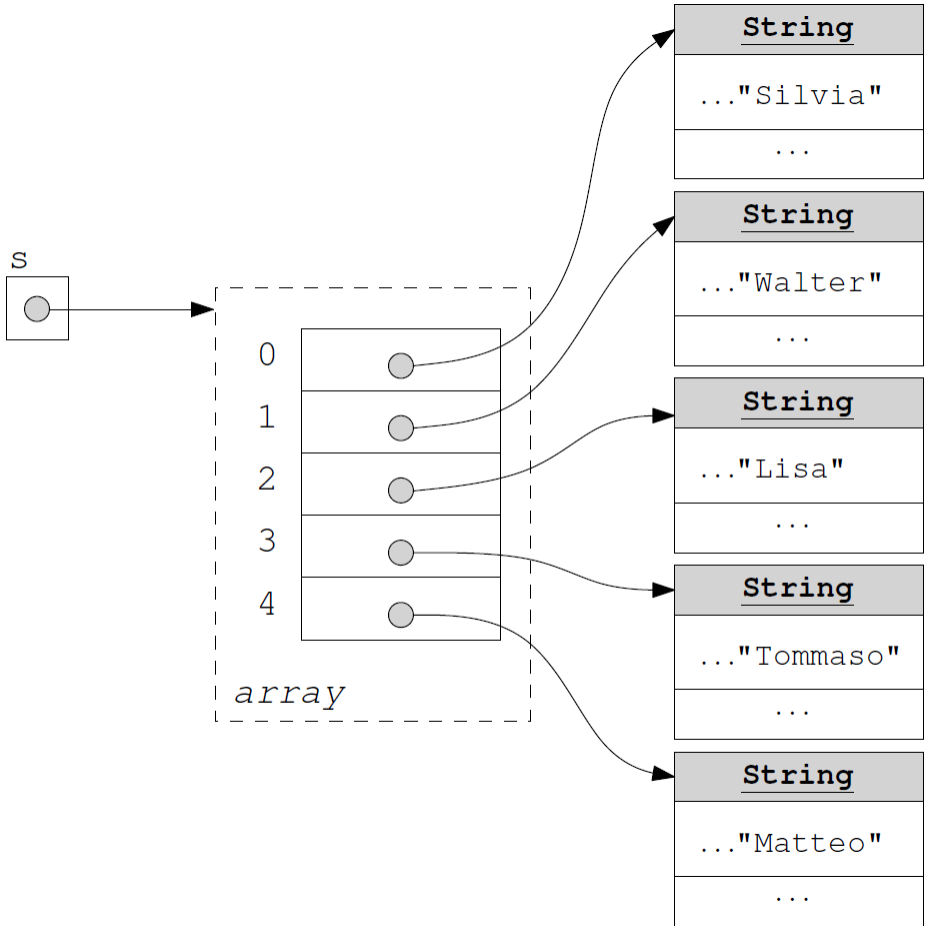
Accesso agli elementi di un array

- 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

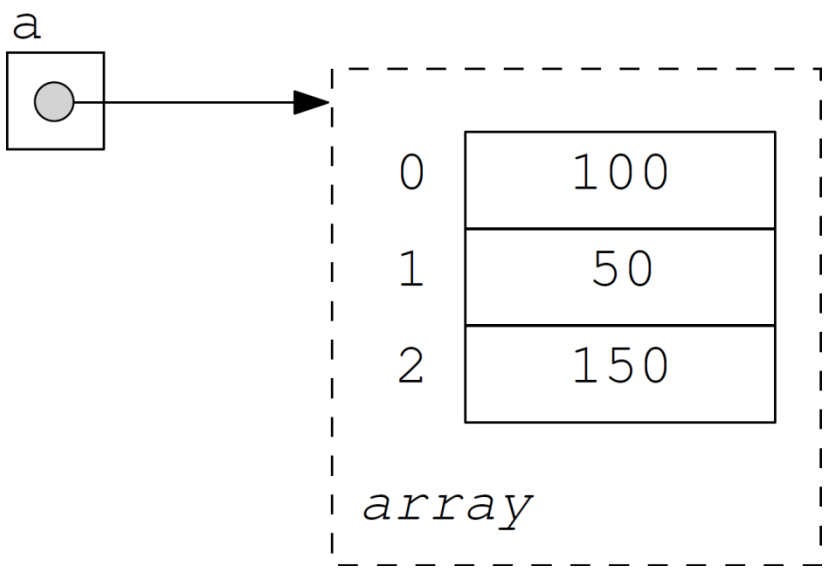
Schema di un array



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];
```



Riconsideriamo il nostro problema

- Utilizzando gli array possiamo risolvere il problema della sequenza di stringhe
- Creiamo un array di dimensione opportuna per memorizzare la sequenza di stringhe
- Sia la lettura delle stringhe che la valutazione della loro lunghezza (con eventuale stampa) avviene scandendo gli elementi dell'array

Soluzione 2

```
import fond.io.*;

public class SequenzaStringheDimensioneFissa2{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    OutputWindow out = new OutputWindow();

    // creazione array per memorizzare le stringhe
    final int DIM = 4;
    String[] s = new String[DIM];

    // inserimento stringhe
    for (int i = 0; i<DIM; i++)
      s[i] = in.readString("Inserire stringa "+i);

    ...
}
```

Soluzione 2

```
import fond.io.*;

public class SequenzaStringheDimensioneFissa2{
  public static void main(String[] args){
    ...

    // inserimento intero positivo
    int k = in.readInt("Inserire un intero positivo");

    // stampa stringhe di lunghezza maggiore di k
    out.writeln("Stringhe piu' lunghe di " + k + ":");
    for (int i = 0; i<DIM; i++)
      if (s[i].length()>k)
        out.writeln(s[i]);
    }
}
```

Soluzione 2: commenti

- La soluzione 2 è più compatta della soluzione 1
  - non c'è bisogno di ripetere più volte le stesse istruzioni
- Il codice inoltre è facilmente modificabile nel caso la lunghezza della sequenza cambi:
  - per passare da *4* a *100* stringhe è sufficiente cambiare il valore della costante *DIM*
- È inoltre possibile adattare facilmente il codice al caso in cui la lunghezza della sequenza viene scelta dall'utente

Soluzione 3

```
import fond.io.*;

public class SequenzaStringheDimensioneFissa2{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    OutputWindow out = new OutputWindow();

    // creazione array per memorizzare le stringhe
    int dim = in.readInt("Quante stringhe vuoi inserire?");
    String[] s = new String[dim];

    // inserimento stringhe
    for (int i = 0; i<dim; i++)
      s[i] = in.readString("Inserire stringa "+i);

    ...
}
```

Soluzione 3

```
import fond.io.*;

public class SequenzaStringheDimensioneFissa2{
  public static void main(String[] args){
    ...

    // inserimento intero positivo
    int k = in.readInt("Inserire un intero positivo");

    // stampa stringhe di lunghezza maggiore di k
    out.writeln("Stringhe piu' lunghe di " + k + ":");
    for (int i = 0; i<dim; i++)
      if (s[i].length()>k)
        out.writeln(s[i]);
    }
}
```

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 da 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!!
```

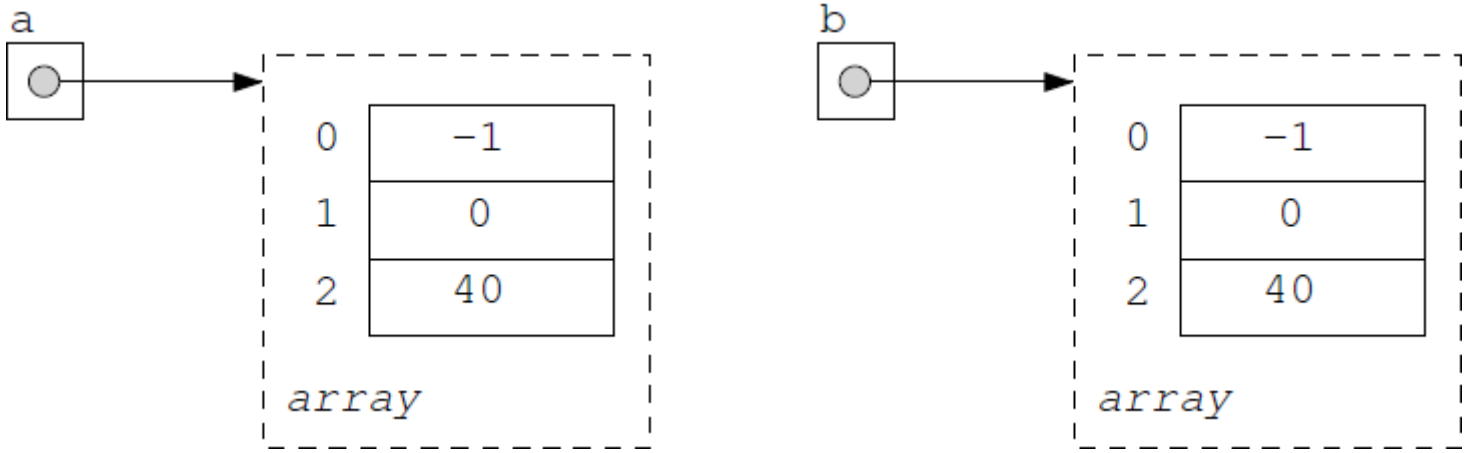
Copia di array

- 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 *a*, uno alla volta, in *b*

Copia di array

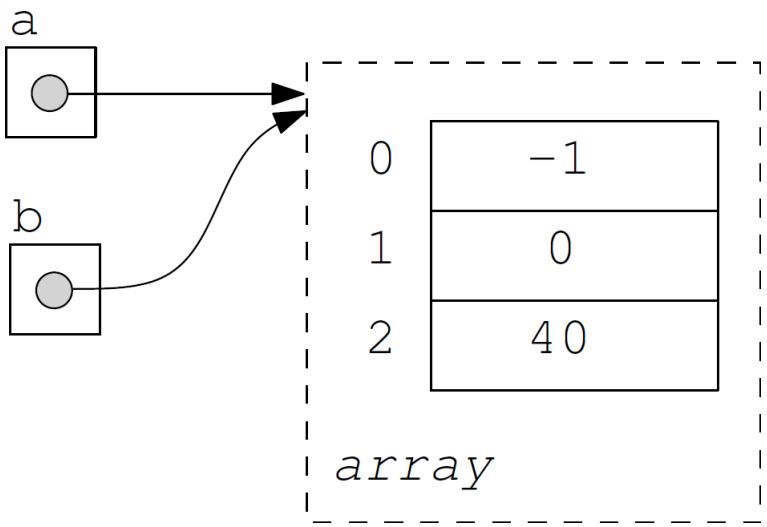
- Si supponga che *a* sia un array di tipo *int*:

```
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



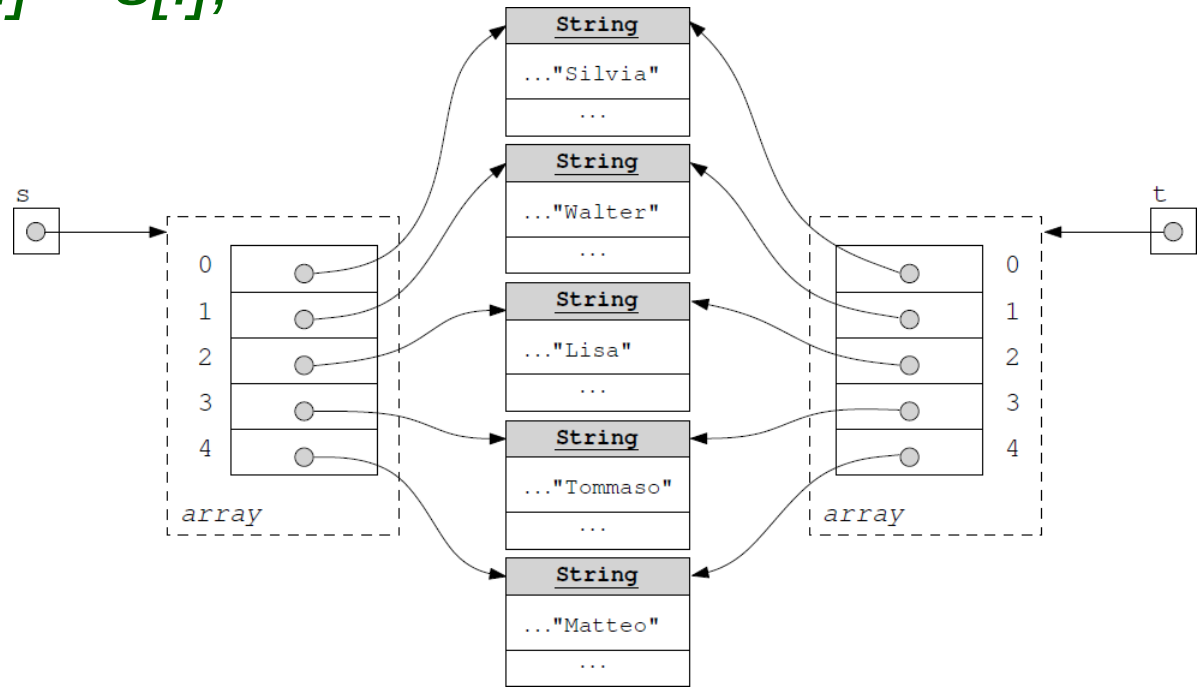
Copia di array

- 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];



Tecniche di base sugli array

- Vogliamo adesso descrivere alcune tecniche di base per la manipolazione di array tra cui:
  - l'individuazione del valore massimo o minimo in una sequenza
  - la ricerca di uno specifico elemento in una data sequenza
  - il confronto di due sequenze di elementi
- Ci concentreremo su sequenze di interi, ma quanto diremo vale più in generale

La classe SequenzaDiInteri

- Definiamo una classe di nome *SequenzaDiInteri*, in base alle seguenti specifiche:
  - un oggetto di tipo *SequenzaDiInteri* rappresenta una qualsiasi sequenza finita di numeri interi;
  - un oggetto di tipo *SequenzaDiInteri* viene creato attraverso un costruttore che riceve come parametro l'intera sequenza che l'oggetto deve rappresentare, espressa sotto forma di array di interi;

La classe SequenzaDiInteri

- la classe definisce i seguenti metodi di istanza:
  - *public int massimo()*: restituisce il valore più grande tra quelli presenti nella sequenza rappresentata dall'oggetto ricevente;
  - *public boolean contiene(int k)*: restituisce il valore *true* se *k* è un valore della sequenza e *false* in caso contrario;
  - *public boolean equals(SequenzaDiInteri altraSeq)*: restituisce il valore *true* se la sequenza rappresentata dall'oggetto ricevente è identica a quella rappresentata dall'oggetto *altraSeq*, e restituisce *false* in caso contrario

La classe SequenzaDiInteri

- Quali sono gli attributi della classe *SequenzaDiInteri*?
- Per memorizzare una sequenza di interi definiamo una variabile di istanza il cui tipo è un array di interi
- Chiameremo questa variabile *seq*

SequenzaDiInteri: scheletro

```
public class SequenzaDiInteri{
   private int[] seq; // la sequenza rappres. dall'oggetto

   /* costruttore: per creare un oggetto che rappresenta
   la sequenza a di interi passata come parametro */
   public SequenzaDiInteri(int[] a){...}

   /* restituisce il massimo valore nella sequenza */
   public int massimo(){...}

   /* restituisce true se la sequenza contiene k */
   public boolean contiene(int k){...}

   /* restituisce true se la sequenza ricevente equivale
   a quella rappresentata da altraSeq */
   public boolean equals(SequenzaDiInteri altraSeq){...}
}
```

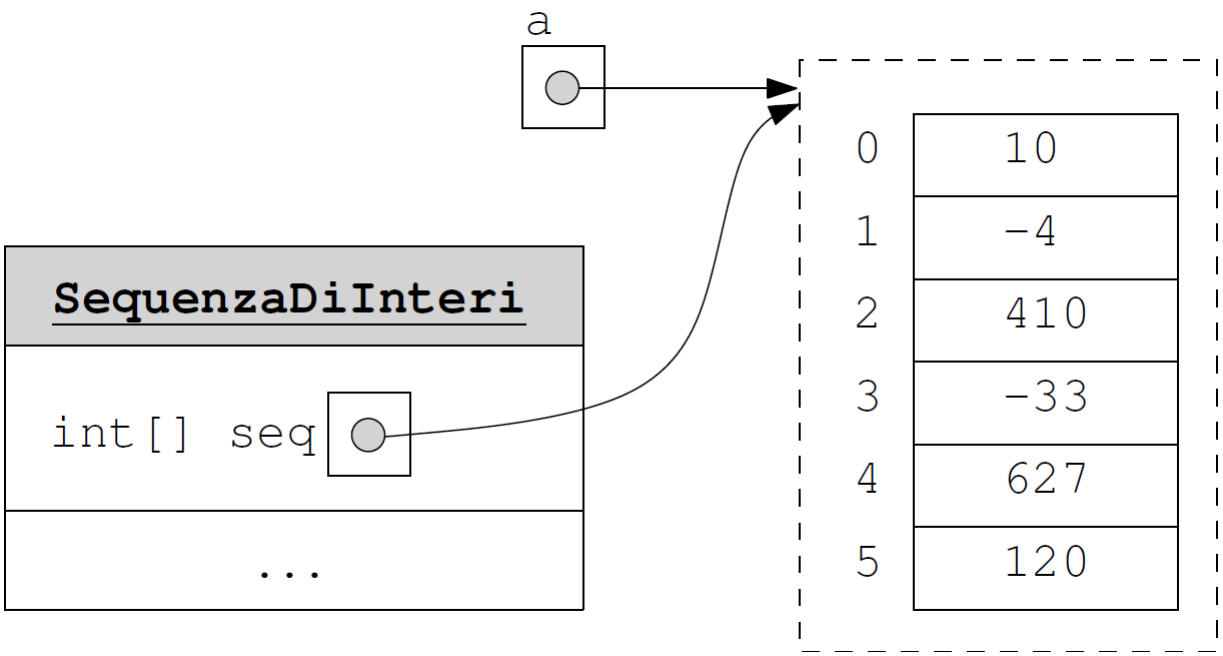
Costruttore

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

```
public SequenzaDiInteri(int[] a){
    this.seq = a;
}
```

Costruttore

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



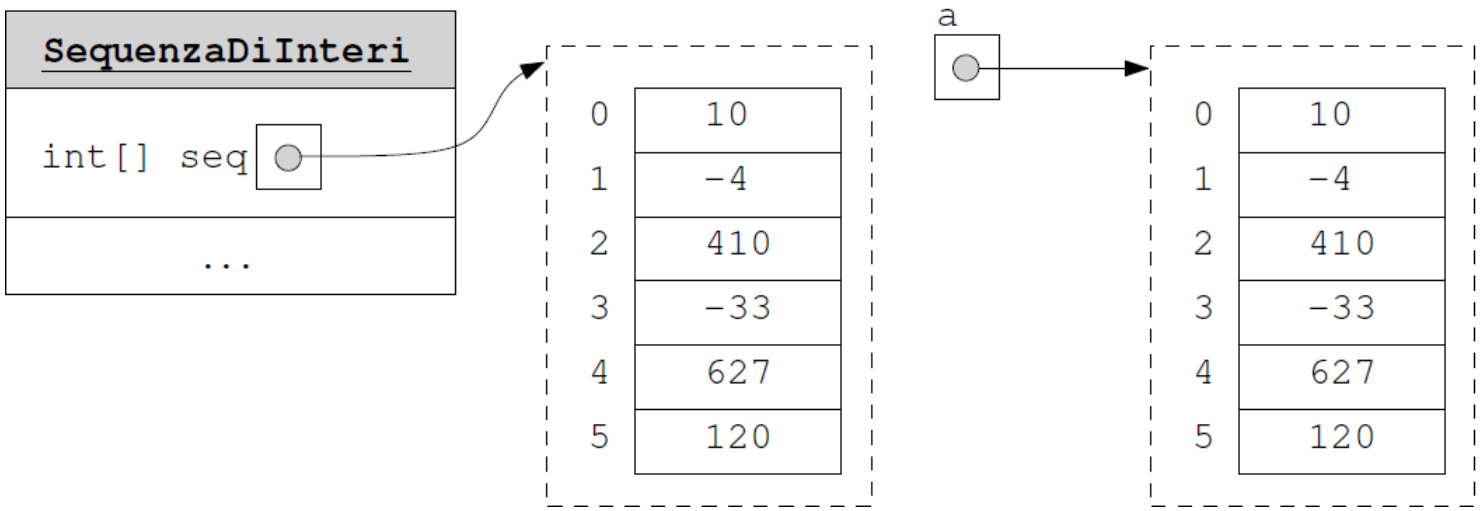
Costruttore

- Uno svantaggio di tale soluzione è che l'array referenziato da *seq* può essere modificato esternamente alla classe *SequenzaDiInteri*
- 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 SequenzaDiInteri(int[] a){
    this.seq = new int[a.length];
    for (int i = 0; i<a.length; i++)
        this.seq[i] = a[i];
}
```



Costruttore

- Con questa soluzione viene creato un nuovo array in cui vengono copiati gli elementi dell'array *a*
- L'array referenziato da *seq* non è visibile esternamente alla classe *SequenzaDiInteri*

Il metodo massimo

- *public int massimo()*
- Algoritmo:
  - manteniamo una variabile *max* per memorizzare il massimo
  - tale variabile viene inizializzata con il primo elemento della sequenza
  - scandiamo i restanti elementi dell'array
  - se l'elemento corrente è maggiore di *max*, quest'ultima viene aggiornata

Il metodo massimo

```
public int massimo (){
   // inizializzazione del massimo (passo 0)
   int max = this.seq[0];

   // scansione e confronto con gli altri elementi
   for (int i = 1; i<this.seq.length; i++)
       if (this.seq[i]>max)
              max = this.seq[i];
   return max;
}
```

Il metodo contiene

- *public boolean contiene(int k)*
- Algoritmo:
  - scandiamo gli elementi della sequenza
  - se l'elemento corrente è uguale al valore cercato *k* la ricerca termina e il metodo restituisce *true*
  - se la sequenza termina senza avere mai trovato il valore *k* il metodo restituisce *false*

Il metodo contiene

```
public boolean contiene(int k){
   boolean esiste = false;
   int i = 0;
   while (!esiste && i<this.seq.length){
       if (this.seq[i]==k)
             esiste = true; // elemento trovato
       i++;
   }
   return esiste;
}
```

Il metodo equals

- *public boolean equals(SequenzaDiInteri altraSeq)*
- Algoritmo:
  - verifichiamo innanzi tutto che la sequenza rappresentata da *this* e quella rappresentata da *altraSeq* abbiano la stessa lunghezza
  - verifichiamo poi che gli elementi di pari indice siano uguali

Il metodo equals

```
public boolean equals(SequenzaDiInteri altraSeq){
   boolean uguali;
   if (this.seq.length != altraSeq.seq.length)
       uguali = false;
   else{
       uguali = true;
       int i=0;
       while (uguali && i<this.seq.length){
              if (this.seq[i] != altraSeq.seq[i])
                     uguali = false;
              i++;
       }
   }
   return uguali;
}
```

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 salvate 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 alcuna 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

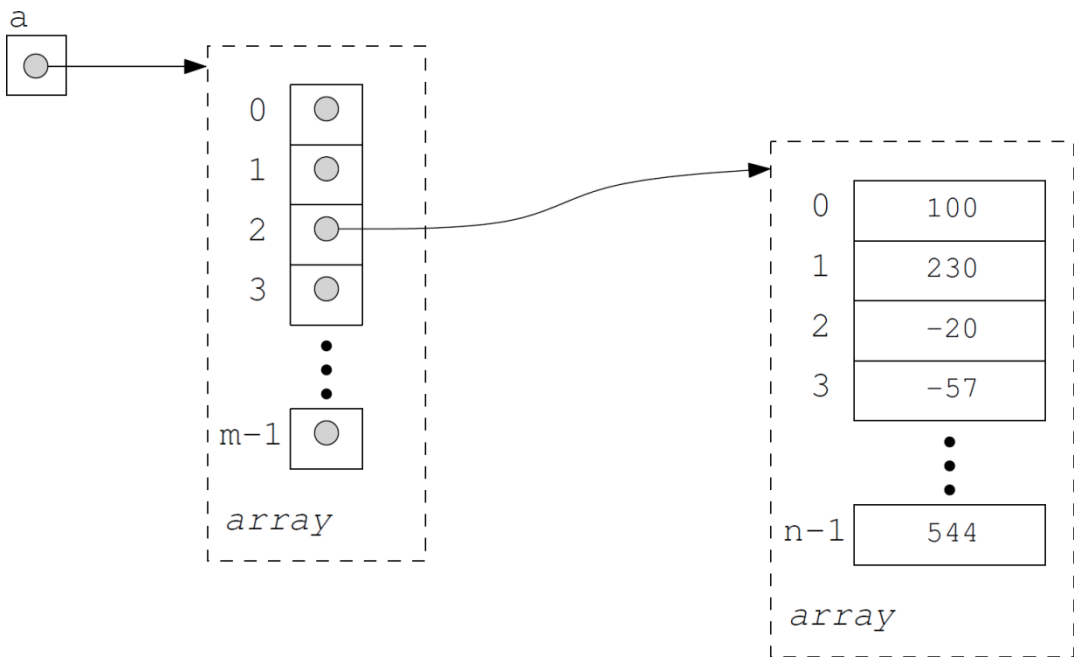
```
public class Somma{
  public static void main(String[] args){
    if (args.length<2)
      System.out.println("Num. param. errato!");
    else{
      Integer n1 = Integer.valueOf(args[0]);
      Integer n2 = Integer.valueOf(args[1]);
      System.out.println(n1.intValue()+n2.intValue());
    }
 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

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 o monodimensionali
- 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
- È 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

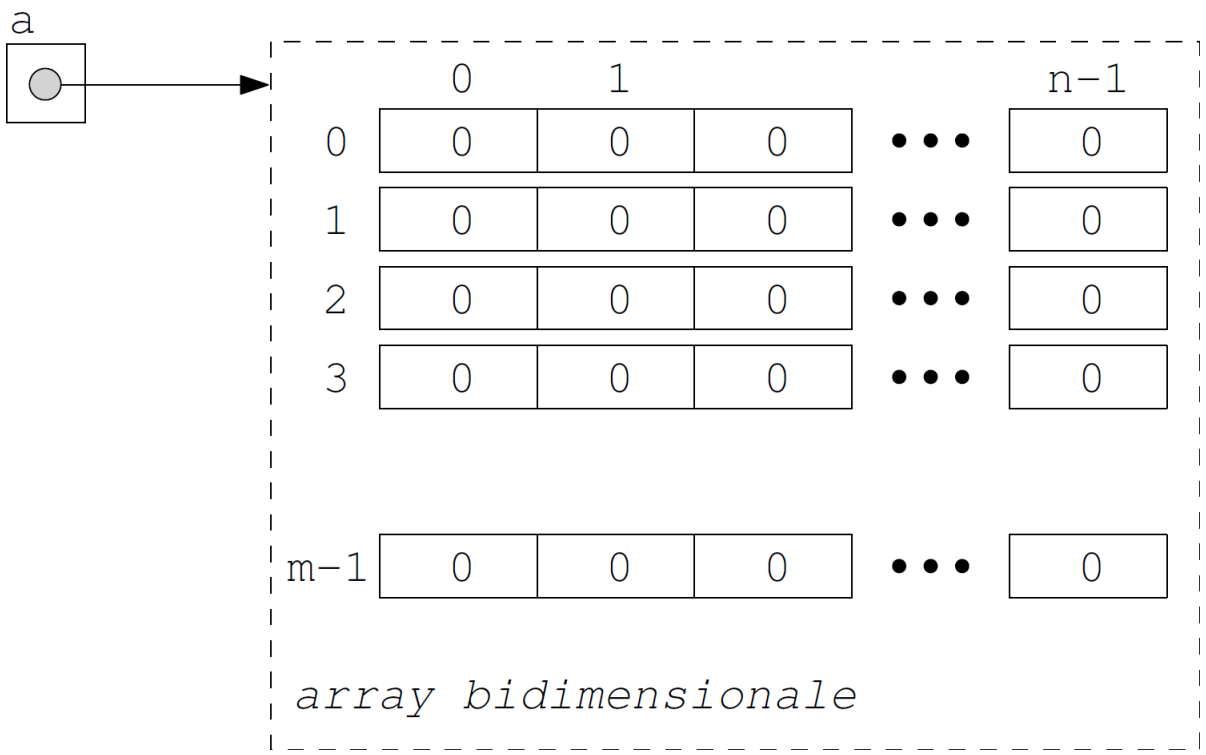
- Il tipo dell'array creato è array di array di tipo T oppure array bidimensionale di tipo T
- Tale tipo di dato si indica con *T[][]*
  - possiamo quindi definire una variabile di questo tipo per memorizzare il riferimento ad una array bididmensionale di tipo *T*
- Gli elementi dell'array vengono inizializzati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

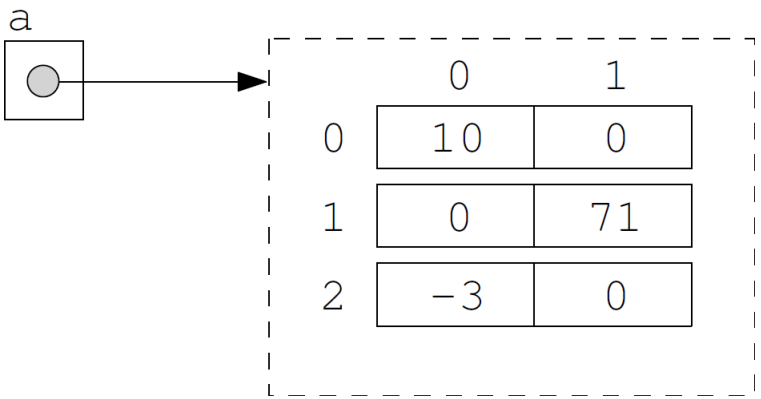


Accesso agli elementi

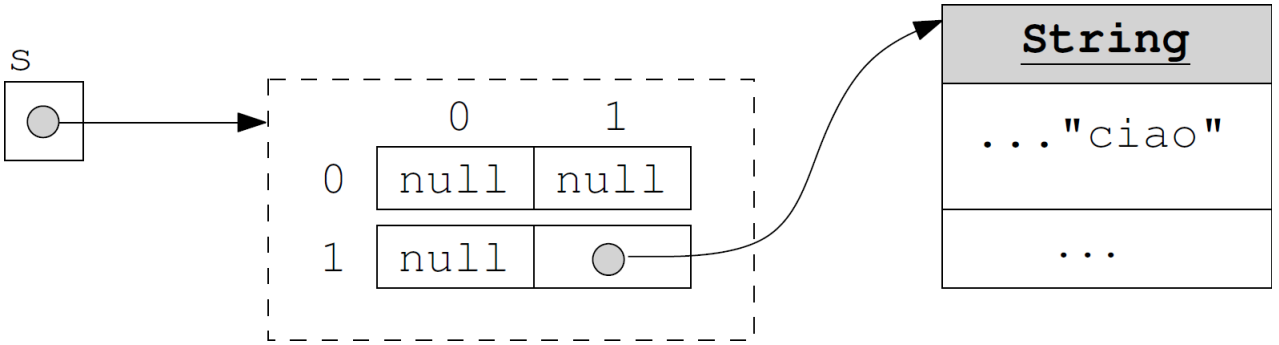
- Per accedere agli elementi di un array bidimensionale sono necessari due indici detti rispettivamente indice di riga e indice di colonna
- Ad esempio con *a[i][j]* si denota l'elemento di riga *i* e colonna *j* dell'array bidimensionale *a*
- Più precisamente:
  - *a[i]* denota l'array monodimensionale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

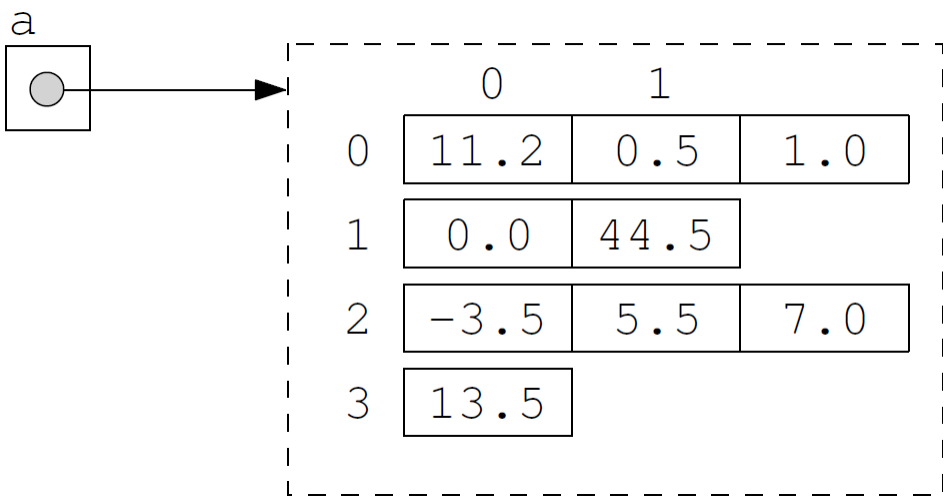
  int[][] a={{10, 0}, {0, 71}, {-3, 0}};
- *{{10, 0}, {0, 71}, {-3, 0}}* è un letterale array bidimensionale

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 scanditi 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]);
    }
}
```

Scandisce le colonne dell'array

Scandisce le righe dell'array

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

- Vediamo ora alcuni esempi d'uso di array bidimensionali
- Realizziamo un programma che:
  - fa inserire all'utente una matrice di numeri interi; l'utente deve scegliere sia le dimensioni (righe colonne) che i singoli elementi della matrice;
  - per ogni riga della matrice inserita, visualizza all'utente la somma degli elementi di tale riga.

SommaRigheMatrice: esempio

- Se la matrice inserita dall'utente fosse:

$$\begin{bmatrix} 4 & 2 & 11 & -2 \\ 10 & 5 & -3 & 0 \\ -1 & -3 & 6 & 1 \end{bmatrix}$$

- il programma dovrebbe stampare:

```
Somma elementi di riga 0 = 15
Somma elementi di riga 1 = 12
Somma elementi di riga 2 = 3
```

SommaRigheMatrice: codice

```
import fond.io.*;

public class SommaRigheMatrice{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    OutputWindow out = new OutputWindow();

    // crea una matrice di dimensioni specificate
    int m = in.readInt ("Numero di righe della matrice?");
    int n = in.readInt ("Numero di colonne della matrice?");
    int[][] matrice = new int[m][n];

    // fa inserire all'utente gli elementi della matrice
    for (int i = 0; i<matrice.length; i++)
      for (int j = 0; j<matrice[i].length; j++)
        matrice[i][j] = in.readInt("Elemento ("+i+","+j+")?");
    ...
  }
}
```

SommaRigheMatrice: codice

```
import fond.io.*;

public class SommaRigheMatrice{
  public static void main(String[] args){
    ...
    // visualizza le somme degli elementi di ogni riga
    int somma;
    for (int i = 0; i<matrice.length; i++){
      // calcola la somma sulla riga i
      somma = 0;
      for (int j = 0; j<matrice[i].length; j++)
        somma += matrice[i][j];
      // visualizza la somma sulla riga i
      out.writeln("Somma riga " + i + " = " + somma);
    }
  }
}
```

Esempio 2: la classe Matrice

- Vogliamo definire una classe di nome *Matrice* i cui oggetti rappresentano matrici di numeri reali
- La classe deve metterci a disposizione metodi per effettuare operazioni tipiche tra matrici, quali la somma e il prodotto righe per colonne

Esempio 2: la classe Matrice

- La classe *Matrice* avrà i seguenti metodi e costruttori:
  - *Matrice(double[][] matrice)*: costruttore per creare un oggetto *Matrice*; l'array bidimensionale *matrice* specifica quale sia la matrice che l'oggetto creato dovrà rappresentare
  - *Matrice somma(Matrice altra)*: restituisce un nuovo oggetto *Matrice* la cui matrice rappresentata è pari alla somma delle matrici rappresentate dall'oggetto ricevente (*this*) e dall'oggetto *altra*
    - si assume che le due matrici abbiano lo stesso numero di righe e lo stesso numero di colonne

Esempio 2: la classe Matrice

- La classe *Matrice* avrà i seguenti metodi e costruttori:
  - *Matrice prodotto(Matrice altra)*: restituisce un nuovo oggetto *Matrice* la cui matrice rappresentata è il prodotto righe per colonne tra la matrice rappresentata dall'oggetto ricevente (*this*) e quella rappresentata dall'oggetto *altra*
    - Si assume che il numero di colonne della matrice dell'oggetto ricevente sia uguale al numero di righe della matrice dell'oggetto *altra*
  - *String toString()*: restituisce una descrizione della matrice rappresentata dall'oggetto ricevente (*this*)

La classe Matrice: scheletro

```
public class Matrice{
  /* memorizza la matrice rappresentata dall'oggetto */
  private double[][] mat;

  /* costruttore: crea un oggetto che rappresenta
  la matrice passata come parametro */
  public Matrice(double[][] matrice){...}

  /* restituisce la somma tra la matrice ricevente (this) e quella
   passata come parametro (altra)
   PRE: this e altra hanno le stesse dimensioni*/
  public Matrice somma(Matrice altra){...}

  /* restituisce il prodotto tra la matrice ricevente (this) e quella
   passata come parametro (altra)
   PRE: il numero di colonne di this coincide con il numero di righe
   di altra */
  public Matrice prodotto(Matrice altra){...}

  /* restituisce una stringa che descrive la matrice */
  public String toString (){...}
}
```

Costruttore

- Il costruttore deve creare una copia della matrice passata come parametro e memorizzarne il riferimento nella variabile di istanza *mat*
- Come già detto nel caso della classe *SequenzaDiInteri* è preferibile creare un nuovo oggetto copia di quello passato come parametro piuttosto che limitarsi a copiare il riferimento

Costruttore

```
public Matrice(double[][] matrice){
   int m = matrice.length;
   int n = matrice[0].length;
   this.mat = new double[m][n];
   for (int i = 0; i<m; i++)
       for (int j = 0; j<n; j++)
             this.mat[i][j] = matrice[i][j];
}
```

Il metodo somma

- Creiamo prima un array bidimensionale *matrice* delle stesse dimensioni di *this.mat* e *altra.mat*
- Ogni elemento *matrice[i][j]* viene posto uguale a *this.mat[i][j] + altra.mat[i][j]*
- Infine viene creato un oggetto *Matrice* usando *matrice* come parametro del costruttore

Il metodo somma

```
public Matrice somma(Matrice altra){
   int m = this.mat.length;
   int n = this.mat[0].length;
   double[][] matrice = new double[m][n];
   for (int i = 0; i<m; i++)
        for (int j = 0; j<n; j++)
               matrice[i][j] = this.mat[i][j]+altra.mat[i][j];
   Matrice somma = new Matrice (matrice); // risultato
   return somma;
}
```

Il metodo prodotto

- L’approccio è simile a quello del metodo precedente
- Il risultato viene prima memorizzato in un array bidimensionale *matrice* di opportune dimensioni
- La differenza sta nel fatto che per calcolare ciascun elemento della matrice risultato è necessario effettuare una sommatoria di prodotti

Il metodo prodotto

- Ricordiamo come si effettua il prodotto tra due matrici A (di dimensione $m \times n$) e B (di dimensione $n \times p$)
- Il risultato è una matrice C di dimensione $m \times p$
- L'elemento C_{ij} di C è dato da:

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj}$$

Il metodo prodotto

```
public Matrice prodotto(Matrice altra){
  int m = this.mat.length; // righe di this
  int n = this.mat[0].length; // colonne di this (righe di altra)
  int p = altra.mat[0].length; // colonne di altra
  double[][] matrice = new double[m][p];
  for (int i = 0; i<m; i++){
    for (int j = 0; j<p; j++){
      double somma = 0;
      for (int k = 0; k<n; k++)
        somma += this.mat[i][k]*altra.mat[k][j];
      matrice[i][j]=somma;
    }
  }
  Matrice prodotto = new Matrice(matrice); // risultato
  return prodotto;
}
```

calcola l'elemento in posizione (i,j) nella matrice prodotto