Array
capitolo 7

Problema

- Scrivere un programma che:
  - legga dallo standard input una sequenza di dieci numeri in virgola mobile, uno per riga
  - chieda all'utente un numero intero **index** e visualizzi il numero che nella sequenza occupava la posizione indicata da **index**
- Occorre ***memorizzare tutti i valori della sequenza***

Come memorizzarli?

- Potremo:
  - usare dieci variabili diverse per memorizzare i valori
    - x1, x2, x3, x4, x5, x6, x7, x8, x9, x10
  - selezionarli con una lunga sequenza di alternative
    - if (index == 1) System.out.println(x1);
    - …
    - else if (index==10)) System.out.println(x10);
  - ***Ma se i valori dovessero essere mille?***
  - ***O se il numero di valori dovesse essere chiesto all'utente come dato iniziale?***

Soluzione

- Ciò che veramente ci serve è ***un contenitore di dati di dimensione arbitraria***, una dimensione che possa essere ***decisa durante l'esecuzione del programma***
  - Questo non si può fare con un semplice insieme di variabili: il numero di enunciati di dichiarazione di variabili deve essere noto al programmatore
  - Inoltre, il contenitore **non** deve essere analogo a un insieme matematico: ci interessa ***che preservi le posizioni dei dati al suo interno***, come una stringa

© traveler1116/iStockphoto.

Immaginate un treno di vagoni numerati per posizione, ciascuno dei quali contiene un carico dello stesso tipo, ma che puo' variare come quantita'

Le strutture dati

- Strumenti concettuali e concreti per organizzare l'informazione
  - Struttura dati astratta (Abstract Data Type): descrive i dati e le funzionalità che la struttura deve avere
  - Struttura dati: implementa le funzionalità indicate dalla ADT

- Algoritmi e Strutture Dati sono fondamentali per progettare soluzioni computazionali efficaci ed efficienti

Obiettivi

- Imparare a conoscere punti di forza e debolezza delle varie strutture dati
  - Capacità di scegliere la struttura dati più adeguata ai dati e al tipo di elaborazione che dovrete eseguire
- Imparare a conoscere (alcuni) algoritmi per ricercare e ordinare i dati
  - Capacità di sfruttare gli algoritmi migliori in base al contesto in cui opererete per ottenere soluzioni più efficienti
- Implementare le strutture dati: capire cosa c'è sotto il "cofano"
  - Ottimizzare l'utilizzo o inventare qualcosa di nuovo!

Le sequenze (o liste) di dati

- ADT Sequenza (o Lista)
  - Lineare: ogni elemento ha un predecessore e un successore (a parte il primo e l'ultimo)
  - Operazioni: inserimento, rimozione, ricerca

- Strutture dati che la implementano:
  - Array
  - Lista concatenata

Array

- Lo strumento messo a disposizione dal linguaggio Java (e in molti altri linguaggi di programmazione) per memorizzare una sequenza di dati si chiama ***array*** (che significa "schiera" o "sequenza ordinata")

- In Java, un **array** è ***un oggetto*** che rappresenta ***una sequenza posizionale di dati omogenei*** (cioè tutti dello stesso tipo), che vengono detti elementi o componenti dell'array

Sommario

- Come costruire un array
- Come utilizzare un array
- Come conoscere la lunghezza di un array
- Errori tipici
- Array come parametri

Costruire un array

- Un array in Java è un **oggetto**
  - Come ogni oggetto, deve essere costruito con l'operatore **new**, dichiarando il **tipo di dati** che potrà contenere

```
new double[10];
```

- Nella costruzione il tipo di dati è seguito da una **coppia di parentesi quadre** che contiene la **dimensione** dell'array, cioè il numero di elementi che potrà contenere
- **In Java, le parentesi quadre si usano solo per gli array**

Costruire un array

- Il tipo di dati di un array può essere qualsiasi tipo di dati valido in Java
  - Uno dei tipi di **dati fondamentali** o una **classe**
  - Potremo avere quindi array di numeri **interi**, di numeri in **virgola mobile**, di **stringhe**, di **conti bancari**...

Costruire un array

- Si dice anche che l'array è costituito da celle o posizioni, nelle quali sono memorizzati i suoi elementi
  - **le posizioni sono identificate da numeri interi non negativi**
  - esiste il dato in prima posizione (la posizione 0), quello in seconda posizione (la posizione 1), ecc.
  - quindi l'array **NON** rappresenta un insieme matematico, anche perché i dati presenti nell'array possono essere replicati in diverse posizioni

0	1	2	3	4	5	6	7
A	B	C	A	A	F	D	Z

Riferimento a un array

- Come avviene dopo la costruzione di qualunque oggetto, l'operatore **new** restituisce un ***riferimento*** all'array appena creato, che può essere memorizzato in una ***variabile oggetto*** dello stesso tipo

```
double[] values = new double[10];
```

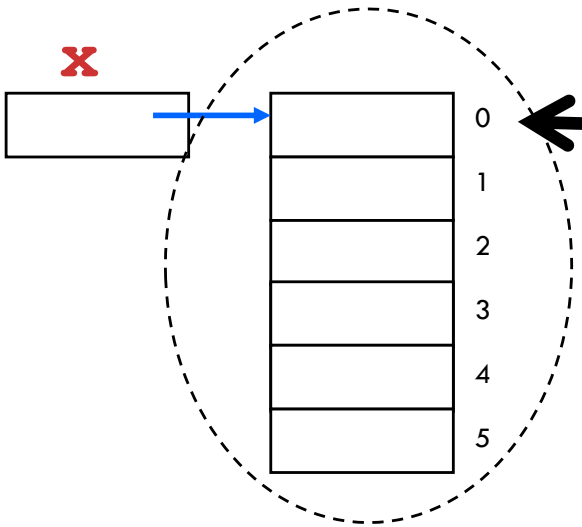
```
// si può fare in due passi
double[] values;
...
values = new double[10];
```

- **Attenzione**: nella definizione della variabile oggetto devono essere presenti le parentesi quadre, ma NON deve essere indicata la dimensione dell'array; la variabile potrà riferirsi solo ad array di quel tipo, ma **di qualunque dimensione**
  - Non esiste, quindi, il **tipo di dato** "array di dieci **double**", bensì il tipo di dato "array di **double**"

Riferimento a un array

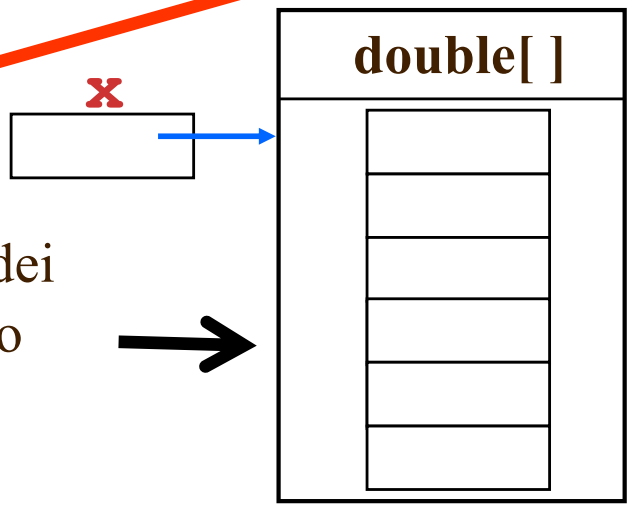
```
double[] x = new double[6];
```

- Una variabile `x` che si riferisce a un array è
  una variabile oggetto che contiene un riferimento all'oggetto array



Rappresentazione schematica di un oggetto di tipo array: la prima cella è quella più in alto

Se interessa anche il tipo dei dati memorizzati, l'oggetto array si può (meglio) raffigurare così



Utilizzare un array

- Al momento della costruzione, tutti gli elementi dell'array vengono inizializzati a un valore, seguendo ***le stesse regole viste per le variabili di esemplare prive di inizializzazione esplicita***

```
double[] values = new double[10];
```

  - ***Tutti gli elementi sono inizializzati a 0 in questo caso***

- Si accede al contenuto di una cella dell'array indicando tra parentesi la posizione voluta
  - Accesso al contenuto della posizione i dell'array values
    - values[i]

Utilizzare un array

- Si può ***leggere*** il contenuto di una cella dell'array, per usarlo in un'espressione

```
double doubleVar = 5 + values[3];
```

- La stessa sintassi si usa per ***scrivere*** un valore in una cella dell'array, ponendola nella parte sinistra di un enunciato di assegnazione

```
values[9] = 3.4 + Math.sqrt(2);
```

Utilizzare un array

```
double[] values = new double[10];
double oneValue = values[3]; // vale 0
values[9] = 3.4;
```

- Il numero utilizzato per accedere a un particolare elemento dell'array si chiama ***indice***
- L'indice può assumere un valore compreso tra **0** (***incluso***) e la **dimensione** dell'array (***esclusa***), cioè segue le stesse convenzioni viste per le posizioni dei caratteri in una stringa
  - il primo elemento ha indice 0
  - l'ultimo elemento ha indice (***dimensione*** - 1)
- **A tutti gli effetti sintattici e semantici, un elemento di un array equivale a una singola variabile dello stesso tipo**
  - `values[3]` è, quindi, una variabile di tipo `double`

Utilizzare un array

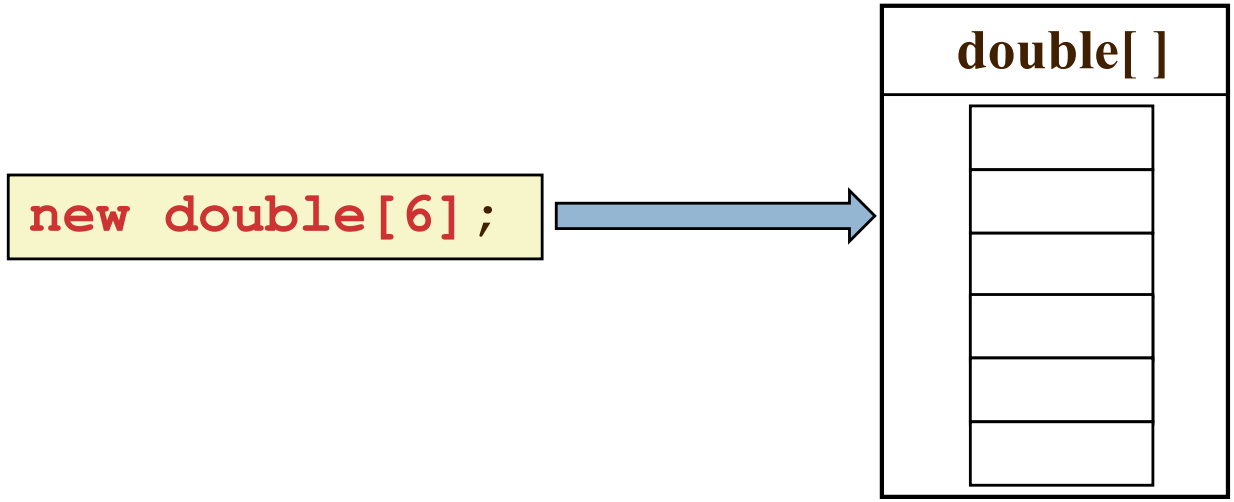
- L'indice di un elemento di un array può, più in generale, essere un'espressione di tipo **`int`**

```
double[] values = new double[10];
int a = 4;
values[a + 2] = 3.2; // modifica il
                     // settimo elemento
                     // (ricordo che parto da indice 0)
```

Utilizzare un array

- Cosa succede se si accede a un elemento dell'array usando un indice sbagliato (maggiore o uguale alla dimensione, o negativo) ?
  - L'ambiente di esecuzione (cioè l'interprete) lancia un'eccezione di tipo **ArrayIndexOutOfBoundsException**
    - In modo simile a quanto abbiamo visto con le stringhe!

Gli array sono una struttura dati ad accesso casuale



- **Struttura ad accesso casuale**
  - Cioè **il tempo di accesso (in lettura o scrittura) a uno dei suoi elementi non deve dipendere dal valore dell'indice associato a tale elemento**
  - Questa proprietà ci sarà molto utile nella valutazione delle prestazioni degli algoritmi in merito al tempo di esecuzione

La dimensione di un array

- Un array è un oggetto un po' strano…
  - non ha metodi pubblici, né statici né di esemplare

- L'unico elemento pubblico di un oggetto di tipo array è la sua dimensione, a cui si accede attraverso la sua **variabile pubblica di esemplare length** (attenzione, non è un metodo!)

```
double[] values = new double[10];
int a = values.length; // a vale 10
```

- Una variabile **pubblica** di esemplare sembrerebbe una violazione delle regole…

La dimensione di un array

- In realtà, **length** è una variabile pubblica ma è **final**, quindi ***non può essere modificata***, può soltanto essere ispezionata
  - Questo paradigma è, in generale, considerato accettabile nell'OOP

```
double[] values = new double[10];
values.length = 15; // ERRORE IN COMPILAZIONE
```

- In alternativa si sarebbe potuto fornire un metodo pubblico per accedere alla variabile privata

```
double[] values = new double[10];
int a = values.getLength(); // non è così!!
```

  - la soluzione progettuale scelta è meno elegante ma fornisce lo stesso livello di protezione dell'informazione ed è più veloce in esecuzione

460016 Soluzione del problema iniziale

```
import java.util.Scanner;
public class SelectValue
{  public static void main(String[] args)
   {
      Scanner console = new Scanner(System.in);
      double[] values = new double[10];
      // usiamo values.length anziché ripetere il
      // numero 10, non usare "numeri magici" !!
      for (int i = 0; i < values.length; i++){
         values[i] = console.nextDouble();
      }
      System.out.println("Inserisci un numero:");
      int index = console.nextInt();
      if (index < 0 || index >= values.length){
         System.out.println("Valore errato");
      }
      else{
         System.out.println(values[index]);
      }
      console.close();
   }
}
```

Variante

```
import java.util.Scanner;
public class SelectValue2
{  public static void main(String[] args)
   {  Scanner console = new Scanner(System.in);
      System.out.println("Dimensione?");
      int dim = console.nextInt();
      double[] values = new double[dim];
      for (int i = 0; i < values.length; i++){
         System.out.println(“Introduci un valore”);
         values[i] = console.nextDouble();
      }
      System.out.println("Inserisci un numero:");
      int index = console.nextInt();
      if (index < 0 || index >= values.length){
         System.out.println("Valore errato");
      }
      else{
         System.out.println(values[index]);
      }
      console.close();
   }
}
```

Errori di limiti negli array

- Uno degli errori più comuni con gli array è l'utilizzo di un *indice che non rispetta i vincoli*
  - il caso più comune è l'uso di un indice uguale alla dimensione dell'array, che è il primo indice non valido…

```
double[] values = new double[10];
values[10] = 2; // ERRORE IN ESECUZIONE
```

- Come abbiamo visto, l'interprete Java segnala questo errore con il lancio di un'eccezione

Errori di mancata inizializzazione

- Attenzione… quando creo un array di oggetti tutti i riferimenti sono null!

```
BankAccount[] accounts = new BankAccount[10];

System.out.println("Saldo nel conto 0"+
                        accounts[0].getBalance());
```

Exception in thread "main" java.lang.NullPointerException

- Prima di invocare metodi devo aver inizializzato le singole celle!

```
BankAccount[] accounts = new BankAccount[10];
accounts[0] = new BankAccount(100);
System.out.println("Saldo del cliente 0"
                           +accounts[0].getBalance());
```

Inizializzazione di un array

- Quando si assegnano i valori agli elementi di un array si può procedere così

```
int[] primes = new int[3];
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
```

ma se si conoscono tutti gli elementi da inserire si può usare questa sintassi

```
int[] primes = { 2, 3, 5};
```

E' possibile la creazione di oggetto senza **new**, che è implicito

Passare un array come parametro

- Spesso si scrivono metodi **statici** che ricevono ***array*** come ***parametri espliciti***

```
public static double sum(double[] values)
{
   if (values.length == 0){
      return 0;//questo if è inutile... perché?
   }
   double sum = 0;
   for (int i = 0; i < values.length; i++){
      sum = sum + values[i];
   }
   return sum;
}
// è comodo che un array abbia length
// altrimenti dovrei passare 2 parametri
```

Passare un array come parametro

- Ricevendo una copia della variabile riferimento, nel metodo invocato **è anche possibile modificare gli elementi** dell'array: il metodo chiamante vedrà gli elementi dell'array

```
public static void incrementAll(double[] values)
{  // se l'array ha lunghezza zero, non fa nulla
   // senza protestare... Scelta ragionevole
   for (int i = 0; i < values.length; i++){
      values[i]++; // si comporta come una variabile
   }
}
```

Estratto di codice dal metodo main:

```
double[] valori = {1,5,10,32};
incrementAll(valori);
for(int i=0; i<valori.length; i++){
   System.out.println(valori[i]);
}
//stampa: 2,6,11,33
```

Usare un array come valore restituito

- Un metodo può anche restituire un (riferimento a un) array

```
public class Util
{  public static String[] getDayNames()
   {  String[] days = {"Lunedì",
                       "Martedì", "Mercoledì",
                       "Giovedì", "Venerdì",
                       "Sabato", "Domenica"};
      return days;
   }
}
```

Attenzione: al termine dell'esecuzione del metodo, la variabile locale **days** "muore", ma l'oggetto array che è stato creato rimane in memoria!
Il suo indirizzo è assegnato alla variabile (**dayNames**) che, nell'invocante, raccoglie il valore restituito dal metodo

- Esempio

```
String[] dayNames = Util.getDayNames();
System.out.println(dayNames[1]);
```

Sintassi sugli array: riassunto

Costruzione di un array

- Sintassi: `new NomeTipo[lunghezza]`

- Scopo: costruire un array per contenere dati del tipo ***NomeTipo***; la ***lunghezza*** indica il numero di dati che saranno contenuti nell'array
  - Se ***lunghezza*** è negativa, viene lanciata l'eccezione **`NegativeArraySizeException`**

Costruzione di un array

```
new NomeTipo[lunghezza]
```

- Nota: ***NomeTipo*** può essere uno dei **tipi primitivi** di Java o il nome di **una classe**

- Nota: i singoli elementi dell'array vengono inizializzati con le stesse regole delle variabili di esemplare che siano prive di inizializzazione esplicita
  - **0** per variabili numeriche e caratteri, **false** per variabili booleane, **null** per variabili oggetto

Variabile riferimento a un array

- Sintassi:

```
NomeTipo[] nomeRiferimento;
```

- Scopo: definire la variabile ***nomeRiferimento*** come variabile oggetto che potrà contenere un riferimento a un array di dati, ciascuno dei quali sarà di tipo ***NomeTipo***

- Le parentesi quadre [ ] sono necessarie e **non** devono contenere l'indicazione della dimensione dell'array

Accesso a un elemento di un array

- Sintassi: `riferimentoArray[indice]`
- Scopo: accedere all'elemento in posizione ***indice*** all'interno dell'array a cui ***riferimentoArray*** si riferisce, per conoscerne il valore o modificarlo

- Nota: il primo elemento dell'array ha indice 0, l'ultimo elemento ha indice (***dimensione*** - 1)
- Nota: se l'***indice*** non rispetta i vincoli, viene lanciata l'eccezione **ArrayIndexOutOfBoundsException**

Argomenti sulla riga di comando

public static void main(String[] args)

Argomenti sulla riga comandi

- Quando si esegue un programma Java, è possibile fornire dei parametri dopo il nome della classe che contiene il metodo **main**

```
java Program 2 33 Hello
```

- Tali parametri vengono letti dall'interprete Java (che li riceve dal sistema operativo) e trasformati in un array di stringhe che costituisce il parametro del metodo **main**

```
public class Program {
   public static void main(String[] args) {
      System.out.println(args.length);
      System.out.println(args[1]);
   } // qui sotto l'output del programma, se viene
}    // eseguito con la riga di comando vista sopra
```

```
3
33
```

Argomenti sulla riga comandi

- Se tra gli argomenti che si vogliono inserire c'e' una stringa composta da piu' parole, e' necessario metterla tra doppi apici

```
MacBook-Pro-di-Cinzia:lezioni2023 cinzia$ java ProgramArgs 10 33 ciao come stai
Il numero di parametri inseriti e' 5
parametro 1 : 10
parametro 2 : 33
parametro 3 : ciao
parametro 4 : come
parametro 5 : stai
MacBook-Pro-di-Cinzia:lezioni2023 cinzia$ java ProgramArgs 10 33 "ciao come stai"
Il numero di parametri inseriti e' 3
parametro 1 : 10
parametro 2 : 33
parametro 3 : ciao come stai
MacBook-Pro-di-Cinzia:lezioni2023 cinzia$
```

Argomenti sulla riga comandi

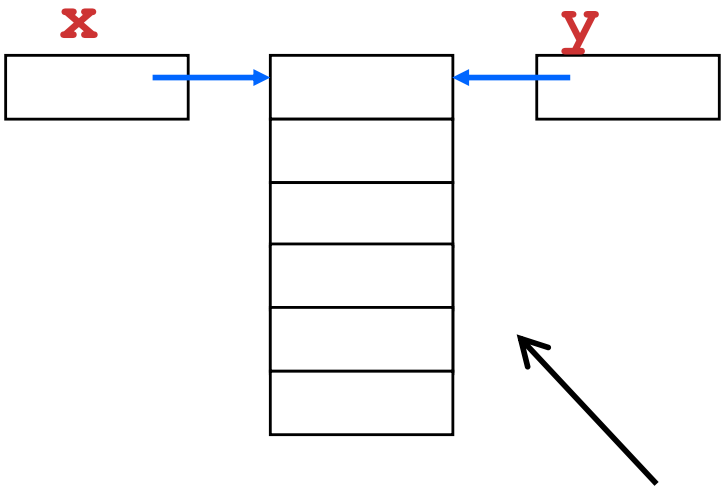
- Le specifiche della JVM dicono che
  - la lunghezza dell'array corrisponde sempre al numero di parametri forniti sulla riga comandi
  - se non vengono forniti parametri sulla riga comandi, viene passato al metodo **main** un array di lunghezza zero (perfettamente valido)

```
public class Program {
   public static void main(String[] args) {
      if (args.length == 0){
         System.out.println("Nessun parametro");
      }
   }
}
```

"Copiare" o clonare un array

Copiare un array

- Una variabile che si riferisce a un array è ***una variabile oggetto che contiene un riferimento all'oggetto array***. Copiando il contenuto della variabile in un'altra ***non si copia l'array***, ma si ottiene un altro riferimento allo ***stesso oggetto array***



Rappresentazione grafica "sintetica" dell'array

```
double[] x = new double[6];
double[] y = x;
```

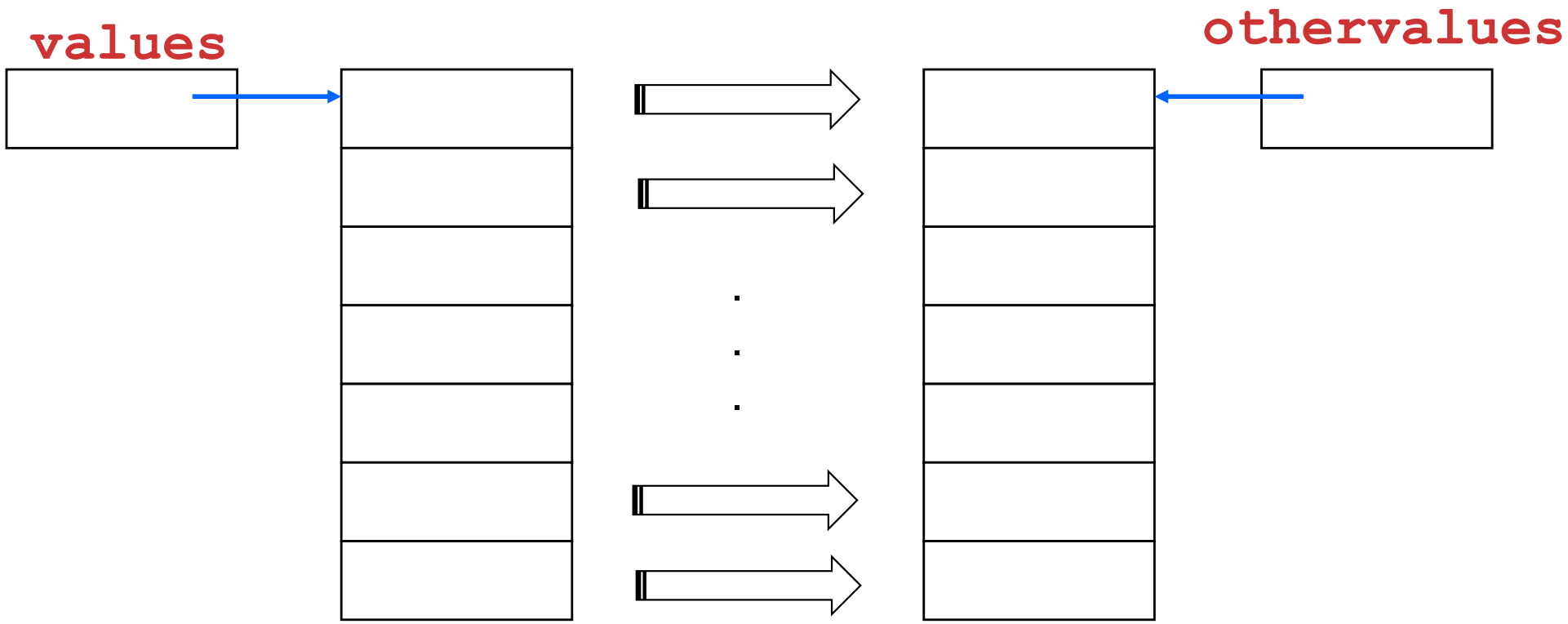
Clonare un array

- Se si vuole ***clonare*** un array, cioè ottenere ***una copia dell'array*** con identico contenuto, bisogna
  - *creare un nuovo array dello stesso tipo e con la stessa dimensione*
  - *copiare ogni elemento del primo array nel corrispondente elemento del secondo array*

```
double[] values = new double[10];
// inseriamo i dati nell'array
...
double[] otherValues = new double[values.length];
for (int i = 0; i < values.length; i++){
   otherValues[i] = values[i];
}
```

Questa clonazione è sempre possibile!!

Clonare un array



Attenzione alla minuscola!

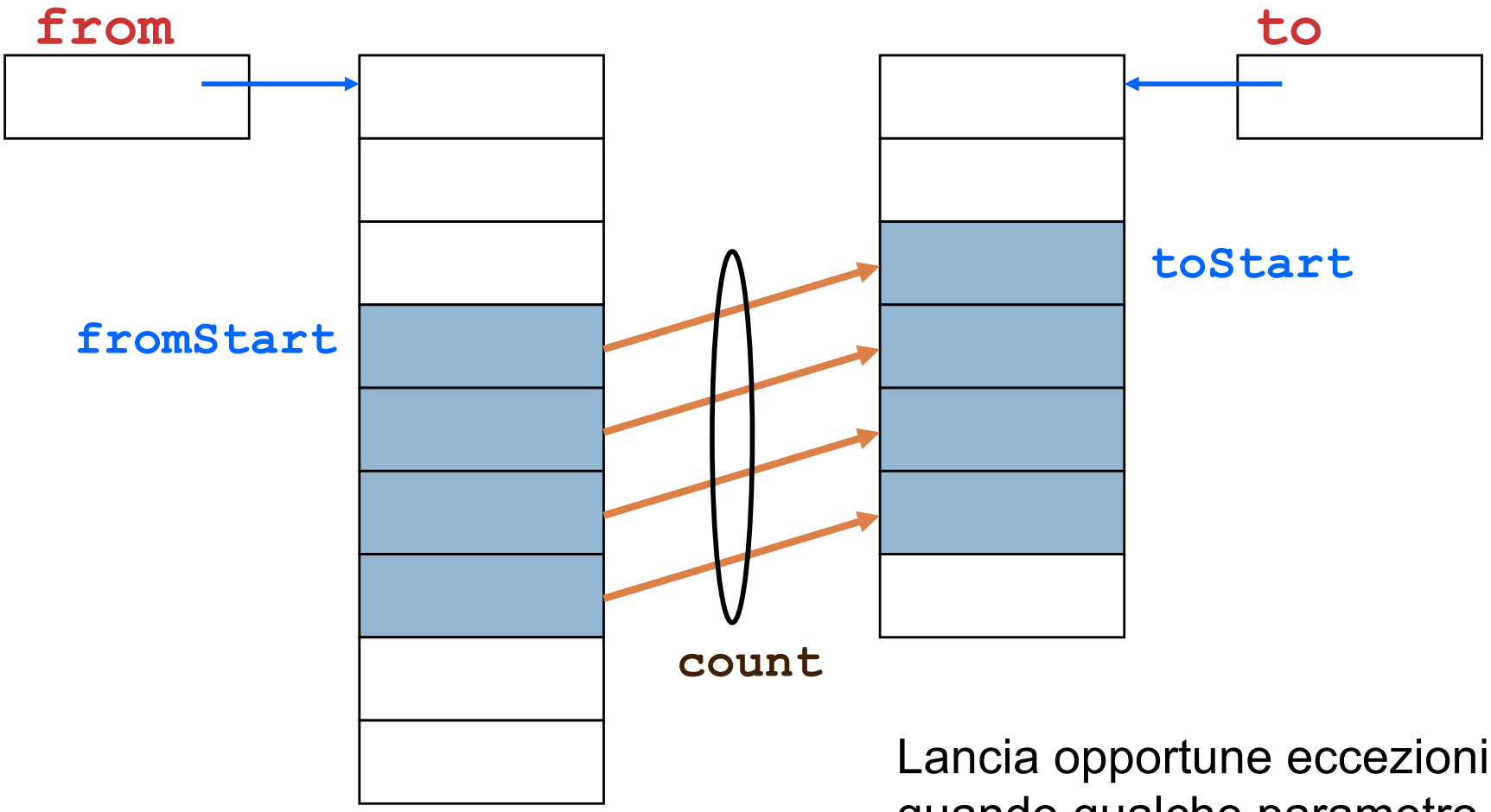
Clonare un array

- Invece di usare un ciclo, è possibile invocare il metodo statico **arraycopy** della classe **System**

- Il metodo **System.arraycopy** consente, più in generale, di copiare una **porzione** di un array in un altro array
  - Anche l'array di destinazione (quello in cui si copiano i dati) deve già esistere e deve essere grande a sufficienza
  - NB: si può usare all'esame!

Parametri di System.arraycopy

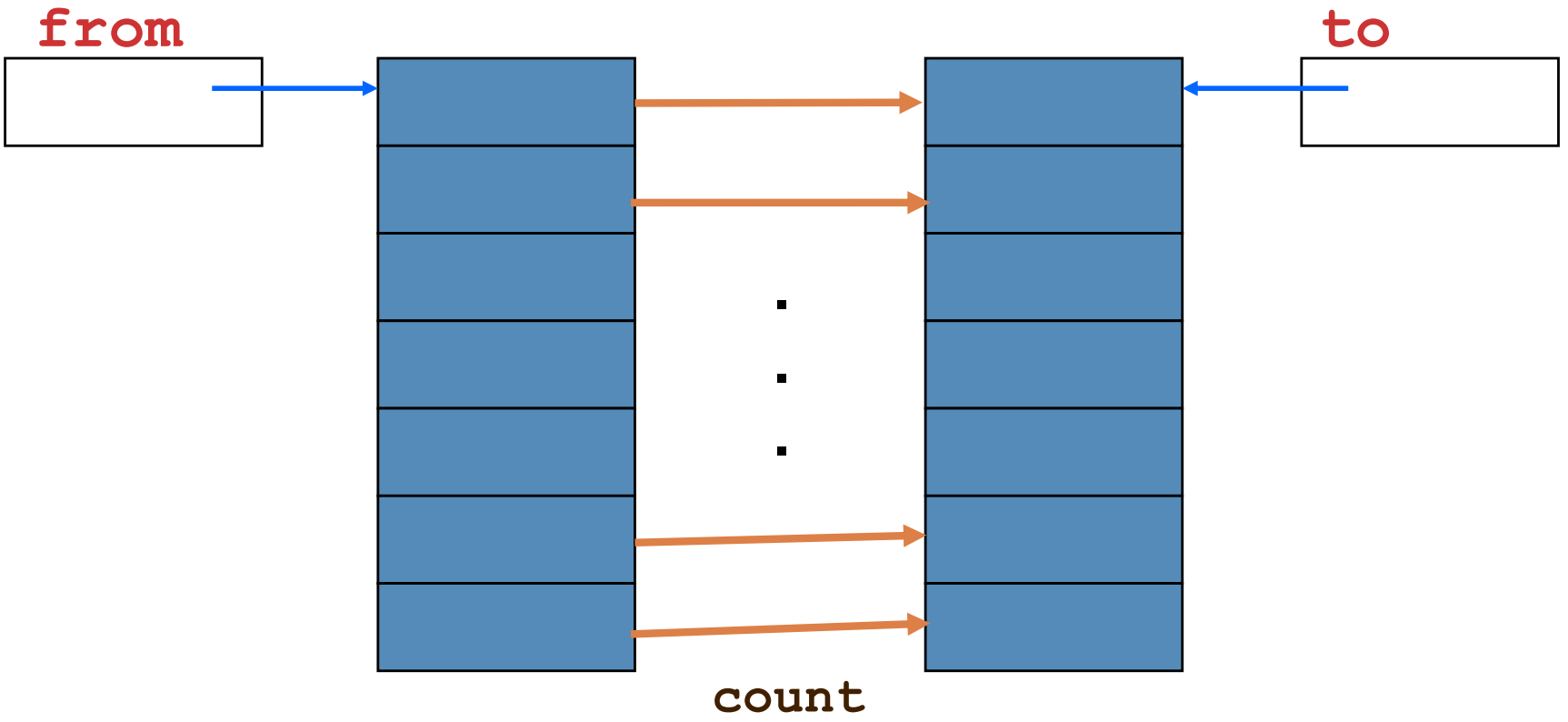
```
System.arraycopy(from, fromStart, to, toStart, count);
```



Lancia opportune eccezioni quando qualche parametro è sbagliato

Copiare un intero array con System.arraycopy

```
System.arraycopy(from, 0, to, 0, from.length);
```



Copio l'intero array "from"

Esempio

```
double[] values = new double[10];
// inseriamo i dati nell'array
...
double[] otherValues = new double[values.length];
System.arraycopy(values, 0, otherValues, 0, values.length);
```

Clonare un array

- E' anche possibile usare il metodo
  - **T[] Arrays.copyOf(T[] original, int newlength)**

```
double[] otherValues = Arrays.copyOf(values,values.length);
```

  - **Se newlength == values.length**
    - Copio tutto l'array
  - **Se newlength < values.length**
    - Tronco l'array alla dimensione fornita
  - **Se newlength > values.length**
    - Copio tutto l'array e nelle celle in più vengono messi valori di default secondo le solite regole
  - NB: non si può usare all'esame!

Clonare un array

- È anche possibile usare il metodo **clone**

```
double[] otherValues = (double[])values.clone();
```

- **Attenzione**: il metodo **clone** restituisce un riferimento di tipo **Object**
- È necessario effettuare un **cast** per ottenere un riferimento del tipo desiderato
- In questo caso **double**[]
- NB: non si può usare all'esame!

Array riempiti solo in parte

Array riempiti solo in parte

- Scrivere un programma che:
  - legga dallo standard input una sequenza di numeri in virgola mobile, uno per riga, ***finché i dati non sono finiti***
    - ad esempio, i dati terminano inserendo una riga vuota (non posso far chiudere lo standard input perché serve ancora…)
  - chieda all'utente un numero intero **index** e visualizzi il numero che nella sequenza occupava la posizione indicata da **index**
- La differenza rispetto al caso precedente è che ora, nel momento in cui il programma deve creare l'array, ***non si sa quanti saranno i dati*** introdotti dall'utente

Array riempiti solo in parte

- Problema: ***per costruire un array è necessario indicarne la dimensione***, che è una sua proprietà **final**
  - ***gli array in Java non possono crescere!***

- Possibile soluzione : costruire un array di ***dimensioni sufficientemente grandi*** da poter accogliere una sequenza di dati di lunghezza "ragionevole", cioè tipica per il problema in esame
  - Vedremo poi una soluzione migliore

Array riempiti solo in parte

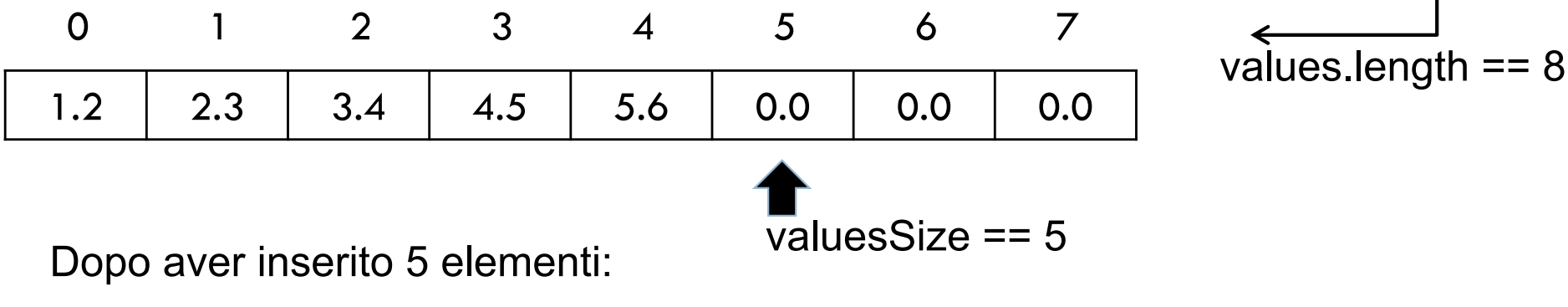
- Nuovo problema: al termine dell'inserimento dei dati da parte dell'utente, in generale, non tutto l'array conterrà dati validi
  - ***è necessario tenere traccia di quale sia l'indice dell'ultimo elemento dell'array che contiene dati validi***

```
Scanner c = new Scanner(System.in);
double[] values = new double[1000]; // oppure 10000 ?
int valuesSize = 0;
while (true) // sembra infinito... ma poi c'è break...
{  String s = c.nextLine();// non usiamo nextDouble perché
                           // cerchiamo una riga vuota
   if (s.equals("")){ // oppure if (s.length() == 0)
      break; // riga vuota!
   }
   values[valuesSize] = Double.parseDouble(s);
   valuesSize++;
}
// valuesSize è l'indice del primo dato non valido
// nell'array e deve sempre essere non negativo e
// non maggiore della dimensione dell'array;
// è anche il numero di dati inseriti!!
// nota: qual è il suo valore iniziale? Osserva bene
System.out.println("Inserisci un numero:");
int index = Integer.parseInt(c.nextLine());
if (index < 0 || index >= valuesSize){
   System.out.println("Valore errato");
}
else{
   System.out.println(values[index]);
}
```

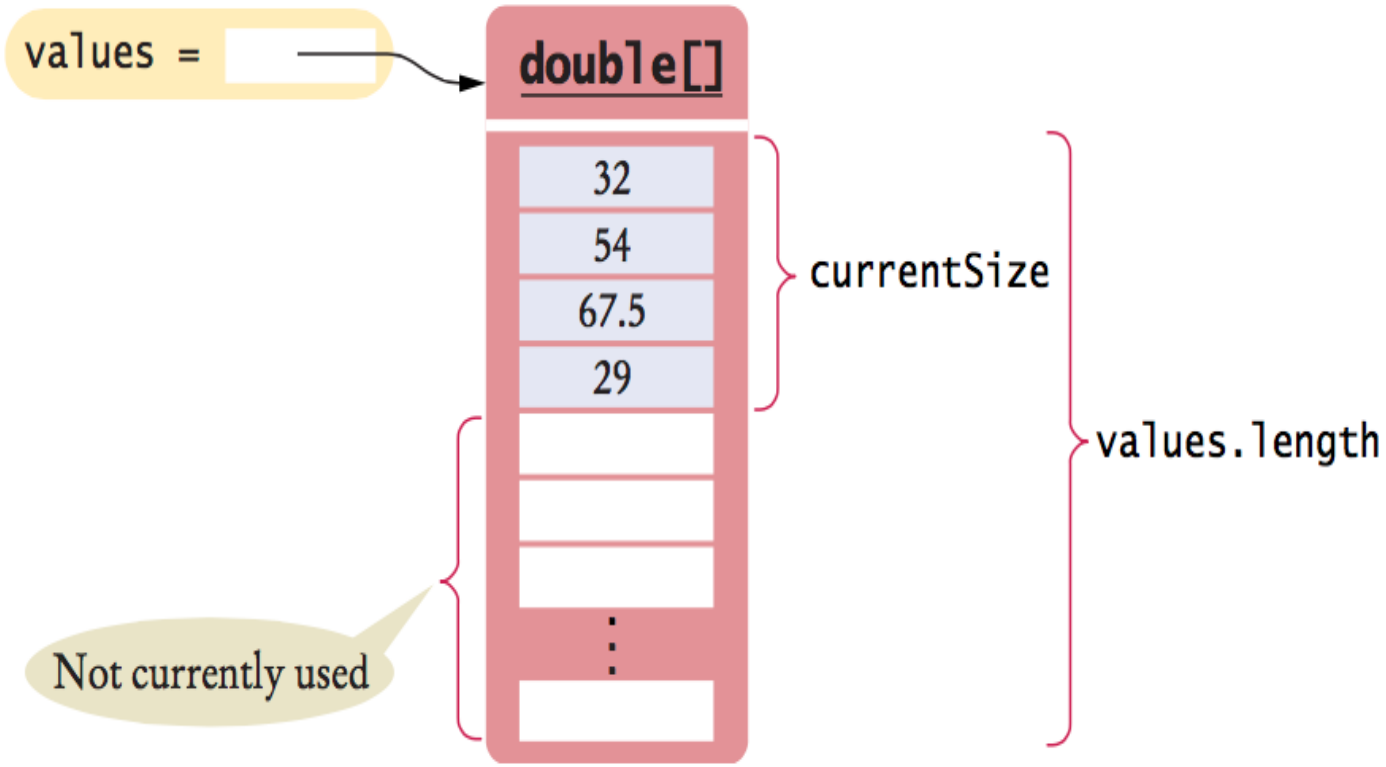
Dimensione fisica vs dimensione logica

- **Dimesione fisica**
  - **values.length** è il numero di valori ***memorizzabili*** ed è la **dimensione fisica dell'array**
- **Dimesione logica**
  - **valuesSize** è il numero di valori ***memorizzati*** ed è la **dimensione logica dell'array**

```
double[] values = new double[8];
```



Dopo aver inserito 5 elementi:


values =
double[]
32
54
67.5
29
currentSize
values.length
Not currently used

Array riempiti solo in parte

- Si dice che un array utilizzato in questo modo è "**riempito solo in parte**", anche se sarebbe meglio dire "**riempito in una sua porzione iniziale**" (eventualmente vuota o eventualmente coincidente con l'intero array)

- **valuesSize** è una variabile **esterna** all'array, ***non esistono oggetti primitivi di tipo "array riempito solo in parte"***

- L'array riempito solo in parte è, quindi, semplicemente **una modalità di utilizzo di un normale array**, a cui viene associata, dal punto di vista logico (ma non strutturale), una variabile di tipo **int**

Array riempiti solo in parte

- La soluzione presentata ha però ancora una debolezza
  - se la ***previsione*** del programmatore sul numero massimo di dati inseriti dall'utente è sbagliata, il programma si arresta con un'eccezione di tipo **ArrayIndexOutOfBoundsException**

```
while (true)
{   String s = c.nextLine();
    if (s.equals("")) break;
    values[valuesSize] = Double.parseDouble(s);
    valuesSize++;
}
```

- Ci sono due possibili soluzioni
  - ***impedire l'inserimento di troppi dati, segnalando l'errore all'utente***
  - ***"ingrandire" l'array quando ce n'è bisogno***

Array riempiti solo in parte

```
// modifichiamo il ciclo di lettura
// per impedire l'inserimento di troppi dati
...
while(true)
{  String s = c.nextLine();
   if (s.equals(""))
      break;
   if (valuesSize == values.length) // è pieno
   {  System.out.println("Basta: troppi dati!");
      // eventualmente si può lanciare eccezione
      break;
   }
   values[valuesSize] = Double.parseDouble(s);
   // attenzione a dove si mette la verifica:
   // metterla qui sarebbe concettualmente errato
   // perché?

   valuesSize++;
}
...
```

Array riempiti solo in parte

```
// modifichiamo il ciclo di lettura
// per impedire l'inserimento di troppi dati
...
while(true)
{  String s = c.nextLine();
   if (s.equals(""))
      break;

   values[valuesSize] = Double.parseDouble(s);

   // ERRATO!!!
   if (valuesSize == values.length) // è pieno
   {  System.out.println("Basta: troppi dati!");
      // eventualmente si può lanciare eccezione
      break;
   }
   valuesSize++;

}
...
```

Cambiare dimensione a un array

- Abbiamo già visto come in Java sia ***impossibile*** aumentare (o diminuire) la dimensione di un array

- Ciò che si può fare è:
  - creare un nuovo array più grande di quello "pieno"
    - ad esempio il doppio, ma potrebbe bastare un elemento in più
  - copiarne il contenuto
  - "abbandonare" il vecchio array, usando poi quello nuovo

Cambiare dimensione a un array

- si parla di ***array dinamico***, sottintendendo che in realtà si tratta di una ***gestione dinamica dell'array***: non può esserci ambiguità perché **in Java non esistono array** dinamici, cioè **di dimensione modificabile**
- si dice anche che si ***ridimensiona l'array***, di nuovo senza ambiguità

Cambiare dimensione a un array

```
if (valuesSize == values.length) // è pieno
{  double[] newValues = new double[values.length * 2];
   for (int i = 0; i < values.length; i++){
      newValues[i] = values[i];
   }
   values = newValues;
// valuesSize ovviamente non cambia;
// values non punta più al vecchio
// array, che viene abbandonato
}
```

Perché `* 2` e non `+ 1`?
Per motivi di **efficienza**, come **vedremo**.
Naturalmente funziona anche con + 1.

Il metodo statico *resize*

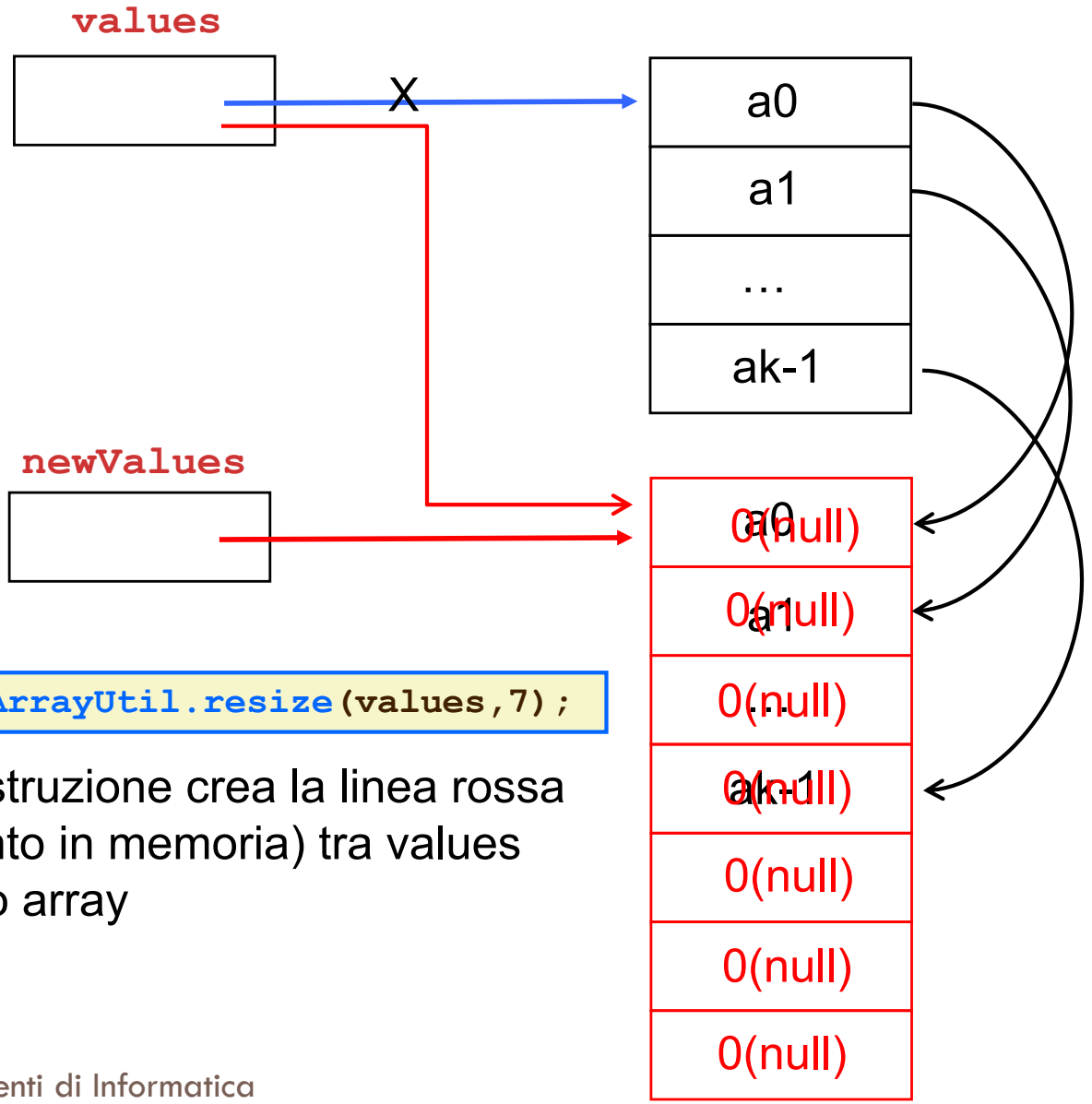
- Possiamo progettare un metodo di utilità che restituisca un array "ingrandito" a partire da quello che viene fornito

```
public class ArrayUtil
{
   public static double[] resize(double[] oldArray,int newLength)
   {  if (newLength < oldArray.length) {
       // gestire la situazione come piu' opportuno
      }
      double[] newArray = new double[newLength];
      for (int i = 0; i < oldArray.length; i++){
         newArray[i] = oldArray[i];
      }

      return newArray;
   } // la "coda" del nuovo array rimane riempita
     // con i valori predefiniti
}
```

Si usa cosi'

```
double[] values = {1, 2.3, 4.5};
values = ArrayUtil.resize(values, 5);
values[4] = 5.2;
```

Visualizzazione di resize



Il vecchio array viene "abbandonato"

nuovo array restituito da resize

```
values = ArrayUtil.resize(values,7);
```

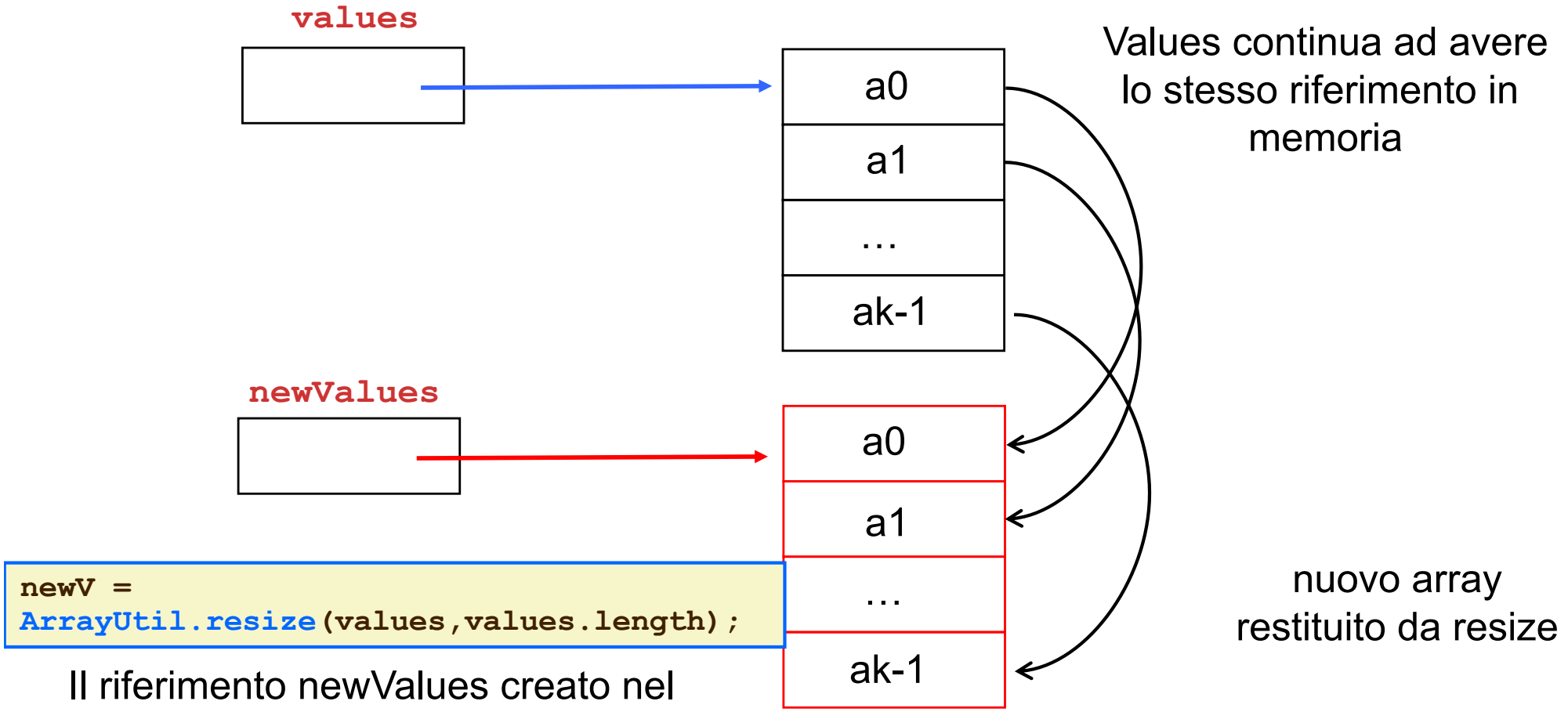
Questa istruzione crea la linea rossa (riferimento in memoria) tra values e il nuovo array

Cambiare dimensione a un array

- Lo stesso metodo può essere usato per **clonare** un intero array; in seguito, è consentito utilizzare sia il nuovo sia il vecchio

```
double[] values = {1, 2.3, 4.5};
double[] newV = ArrayUtil.resize(values, values.length);

newV[2] = 5.2;

if (values[2] == 4.5) System.out.println("OK");

if (newV[2] == 5.2) System.out.println("OK");
```

Resize



Values continua ad avere lo stesso riferimento in memoria

nuovo array restituito da resize

```
newV =
ArrayUtil.resize(values,values.length);
```

Il riferimento newValues creato nel metodo resize viene restituito e assegnato ad una nuova variabile newV

Garbage collector

- Che cosa succede all'array 'abbandonato'?
  - JVM (Java Virtual Machine) provvede a effettuare automaticamente la gestione della memoria (garbage collection) durante l'esecuzione di un programma

- Viene considerata memoria libera (quindi riutilizzabile) la memoria eventualmente occupata da oggetti che non abbiano più un riferimento nel programma

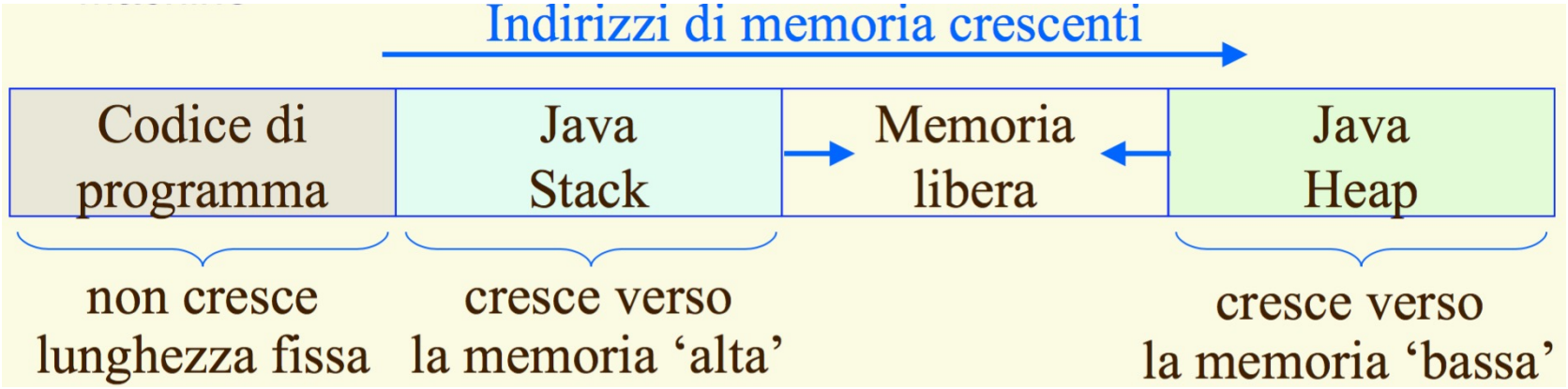
Allocazione della memoria in Java

- A ciascun programma java al momento dell'esecuzione viene assegnata un'area di memoria
- Una parte della memoria serve per memorizzare il codice; quest'area è statica, ovvero non modifica le sue dimensioni durante l'esecuzione del programma
- In un'area dinamica (ovvero che modifica la sua dimensione durante l'esecuzione) detta Java Stack vengono memorizzati i parametri e le variabili locali dei metodi

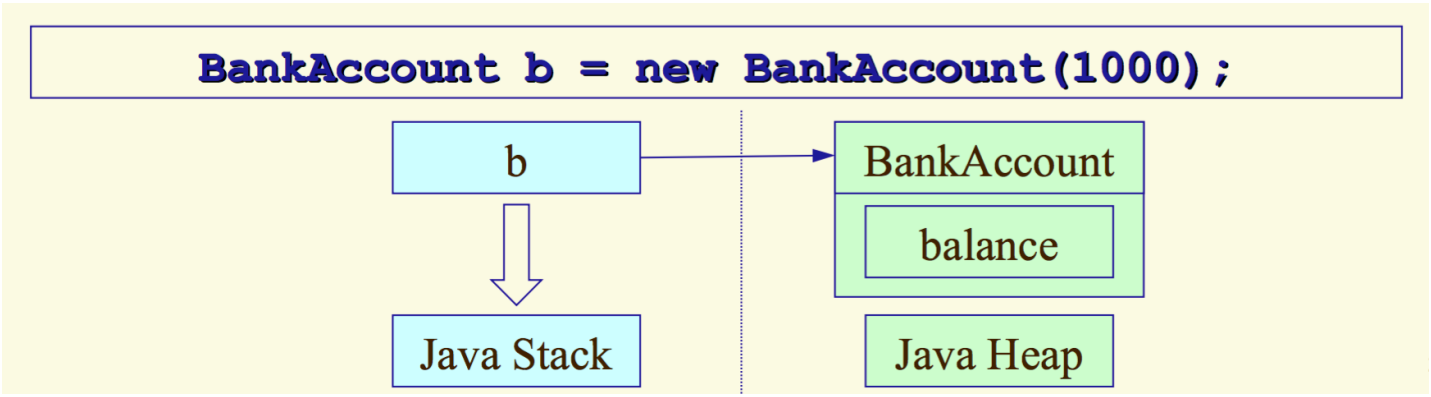
Allocazione della memoria in Java

- Durante l'esecuzione dei metodi di un programma vengono creati dinamicamente oggetti (allocazione dinamica) usando lo speciale operatore new:
  - BankAccount acct = new BankAccount();
  - crea dinamicamente un oggetto di classe BankAccount
- Per l'allocazione dinamica di oggetti Java usa un'area di memoria denominata Java Heap

Disposizione degli indirizzi di memoria nella JVM



- Le variabili parametro e locali sono memorizzate nel ***java Stack o runtime stack***
- Gli oggetti sono costruiti dall'operatore **new** nel ***java Heap***



Array riempito in parte come parametro

- Un metodo che voglia elaborare un array riempito solo in parte deve dichiarare due parametri:
  - **l'array**
  - **la sua dimensione logica**, perché quest'ultima non può essere dedotta ispezionando l'array, come invece avviene per la dimensione fisica
  - NB: se passo come parametro v.length elaboro array pieni!

```
public static double sum(double[] v, int vSize)
{   if (vSize > v.length){
       // qui lanceremo una eccezione!
    }
    double sum = 0;
    for (int i = 0; i < vSize; i++)
       sum = sum + v[i];
    return sum;
}
```

Take home message

- Array: contenitori per dati omogenei
  - Array di tipi primitivi
  - Array di oggetti
- Dimensione di un array
  - Fisica: definita al momento della dichiarazione
  - Logica: si usa una variabile per tener conto del riempimento dell'array
    - utilizzo come array riempito "in parte"
- Modificare la dimensione di un array
  - Di fatto si crea un nuovo array e si copiano i dati
  - Se attribuisco il nuovo riferimento al "vecchio" array è "come se" avessi aumentato la dimensione dell'array

Array bidimensionali e multidimensionali

Array bidimensionali

- Problema
  - stampare una tabella con i valori delle potenze $\mathbf{x}^{\mathbf{y}}$, per ogni valore di **x** tra 1 e 4 e per ogni valore di **y** tra 1 e 5

```
1     1     1     1     1
2     4     8    16    32
3     9    27    81   243
4    16    64   256  1024
```

Cerchiamo di risolverlo in modo generale, scrivendo metodi che possano ***elaborare un'intera struttura di questo tipo***

1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024

Matrici

- Una struttura di questo tipo, con dati organizzati in righe e colonne, si dice ***matrice*** o ***array bidimensionale***
- Un elemento all'interno di una matrice è identificato da *una coppia (ordinata) di indici*
  - un ***indice di riga***
  - un ***indice di colonna***
- In Java esistono gli array bidimensionali e, in generale, multidimensionali
  - ***sempre di dati omogenei***

Array bidimensionali in Java

- Dichiarazione di un array bidimensionale con elementi di tipo **int**

```
int[][] powers;
```

- Costruzione di array bidimensionale di **int** con **4** righe e **5** colonne

```
new int[4][5];
```

- Assegnazione di riferimento ad array bidimensionale

```
powers = new int[4][5];
```

- Accesso a un elemento di un array bidimensionale

```
powers[2][3] = 2;
int x = powers[0][1] + 5;
```

Oppure, viceversa, 4 colonne e 5 righe, basta mantenere poi la convenzione coerente: il significato "logico" dei due indici è esterno a Java

Array bidimensionali in Java

- Ciascun indice deve essere
  - di tipo `int`
  - non negativo
  - minore della dimensione corrispondente
- Per conoscere il valore delle due dimensioni
  - La prima dimensione è `powers.length;`
  - La seconda dimensione è `powers[0].length;`

  (perché un array bidimensionale è in realtà un array di array…)
- In un array tridimensionale (double[][][] powers), la terza dimensione è, analogamente

  `powers[0][0].length;`

Esempio

- Scrivere un programma che visualizzi una tabella con i valori delle potenze "x alla y", con x e y che variano indipendentemente tra 1 ed un valore massimo assegnato dall'utente.

- I dati relativi a ciascun valore di x compaiono su una riga, con y crescente da sinistra a destra e x crescente dall'alto in basso.

```
1     1     1     1     1
2     4     8    16    32
3     9    27    81   243
4    16    64   256  1024
```

```
import java.util.Scanner;

public class TableOfPowers
{
    public static void main(String[] args)
    {
        Scanner c = new Scanner(System.in);
        System.out.println("Calcolo dei valori di x alla y");

        System.out.println("Valore massimo di x:");
        int maxX = c.nextInt();

        System.out.println("Valore massimo di y:");
        int maxY = c.nextInt();

        int maxValue =
              (int)Math.round(Math.pow(maxX, maxY));

        int columnWidth =
          1 + Integer.toString(maxValue).length();

        // ora uso due metodi ausiliari

        int[][] powers = makeAndFillPowers(maxX, maxY);

        display(powers, columnWidth);
    }
(continua)
```

1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024

(continua)

```
   /*
        Questo metodo genera un array bidimensionale
        con i valori delle potenze di x alla y.
   */
   private static int[][] makeAndFillPowers(int x, int y)
   {
      int[][] mat = new int[x][y];

      for (int i = 0; i < x; i++) // riempie per righe
      {
         for (int j = 0; j < y; j++) // riempie per col
         {                           // calcola la riga i
            mat[i][j] =
               (int)Math.round(Math.pow(i + 1, j + 1));
         }

      }

      return mat;                    // attenzione all'uso
   }                                 // degli indici qui sopra:
                                     // perché? ragionare
```

(continua)

```
i=0
    j=0                                    i     j
       mat[0][0] = Math.pow(0+1,0+1) = 1^1
    j=1
       mat[0][1] = Math.pow(0+1,1+1) = 1^2
    j=2
       mat[0][2] = Math.pow(0+1,2+1) = 1^3
    j=3
       mat[0][3] = Math.pow(0+1,3+1) = 1^4
    j=4
       mat[0][4] = Math.pow(0+1,4+1) = 1^5

i=1
    j=0
       mat[1][0] = Math.pow(1+1,0+1) = 2^1 = 2
    j=1
       mat[1][1] = Math.pow(1+1,1+1) = 2^2 = 4
    j=2
       mat[1][2] = Math.pow(1+1,2+1) = 2^3 = 8
    j=3
       mat[1][3] = Math.pow(1+1,3+1) = 2^4 = 16
    j=4
       mat[1][4] = Math.pow(1+1,4+1) = 2^5 = 32
```

1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024

La visualizzazione deve procedere necessariamente per righe!

```
(continua)
   /*
       Questo metodo visualizza un array
       bidimensionale di numeri interi con colonne di
       larghezza fissa e valori allineati a destra.
   */
   private static void display(int[][] mm, int width)
   {
      for (int i = 0; i < mm.length; i++)
      {  for (int j = 0; j < mm[0].length; j++)
         {  String s = Integer.toString(mm[i][j]);
            while (s.length() < width){
               s = " " + s;
            }
            System.out.print(s);
         }
         System.out.println();
      }
   }
}
```

Esempio di visualizzazione fornendo i parametri iniziali 4 e 5

```
   1    1    1    1    1
   2    4    8   16   32
   3    9   27   81  243
   4   16   64  256 1024
```

Suddivisione in sottoproblemi

- La suddivisione in sotto-problemi poteva anche essere diversa. Ad esempio:
  - Potevo riempiere la matrice per colonne e poi per righe
    - e/o
  - la matrice viene creata nel main e viene passata a un metodo che la riempie (e restituisce "void")
    - e/o
  - il metodo display può calcolare autonomamente l'ampiezza delle colonne, gli basta ricevere la matrice e ispezionare il valore massimo (quello "in basso a destra")

Semplici algoritmi su array

La classe ArrayUtil

- Costruiremo una classe **ArrayUtil**
- Sarà una "**classe di utilità** " (come la classe **Math**) che contiene una collezione di **metodi statici** che realizzano algoritmi per l'elaborazione di array
- Per ora trattiamo array di numeri **interi**
  - Più avanti tratteremo array di **oggetti generici**

```
public class ArrayUtil
{ ...
 public static int[] resize(int[] oldArray, int newLength)
 { ...}
 public static ...
}
```

```
//in un'altra classe i metodi verranno invocati così
v = ArrayUtil.resize(v,2*v.length);
```

Generare array di numeri casuali

Numeri casuali

- Come facciamo a generare un numero casuale nel calcolatore? Ad esempio, per simulare il lancio di un dado?
  - sappiamo che il calcolatore fa sempre esattamente ciò che gli viene detto di fare mediante un programma, si comporta cioè in modo ***deterministico***, e non casuale!
- Esistono complesse teorie matematiche che forniscono algoritmi in grado di generare sequenze di numeri ***pseudo-casuali*** (cioè ***quasi casuali***), che sono un'ottima approssimazione di sequenze di numeri casuali
  - A partire, ad esempio, dall'istante di tempo in cui vengono eseguiti: un valore che cambia di volta in volta

Numeri casuali

- La classe **Math** mette a disposizione il metodo statico **random( )** per generare sequenze di numeri pseudo-casuali
- Ogni invocazione del metodo restituisce un numero reale pseudo-casuale nell'intervallo **[0, 1[** (cioè 0 compreso e 1 escluso…)

```
double x = Math.random();
```

- Per ottenere, ad esempio, numeri interi casuali compresi nell'intervallo **[a, b]**, basta fare un po' di calcoli…

```
int n = (int)(a + (1+b-a)*Math.random());
```

Esempio

```
public class AverageDice
{  public static void main(String[] args)
   {  double sum = 0;
      final int TRIALS = 100000;
      for (int i = 0; i < TRIALS; i++)
      {  // nuovo lancio del dado
         int d = (int)(1 + 6 * Math.random());
         sum = sum + d;
      }
      double avg = sum / TRIALS;
      System.out.println("Average: " + avg);
   }
}
// meglio: il valore di TRIALS viene
// fornito dall'utente attraverso System.in
```

Array di numeri casuali

- Usando **random** scriviamo nella classe **ArrayUtil** un metodo che genera array di numeri interi casuali, ciascuno dei quali ha un valore compreso tra 0 e n.

```
public static int[] randomIntArray(int length, int n)
{
   int[] a = new int[length];
   for (int i = 0; i < a.length; i++)
      // a[i] e` un num intero casuale tra 0 e n-1
inclusi
      a[i] = (int) (n * Math.random());
   return a;
}
```

La classe Random

- In java.util
- Creo un oggetto r di tipo Random
  - Costruttore senza parametri
  - Costruttore con seed (genera sempre la stessa sequenza)
- Invoco su r i metodi
  - nextInt()
  - nextInt(bound) : numero da 0 a bound-1
  - Equivalenti per altri tipi numerici
  - (mostra codice ProvaRandom)

Convertire array in stringhe

Stampare il contenuto di un array come stringa

- Se cerchiamo di stampare un array sullo standard output non riusciamo a visualizzarne il contenuto ...

```
int[] a = {1,2,3};
System.out.println(a)
```

```
[I@10b62c9
```

- Scriviamo nella classe **ArrayUtil** un metodo che **crei una stringa** contenente gli elementi di un array

```
public static String printArray(int[] v, int vSize)
{
   String s = "[";
   for (int i = 0; i<vSize; i++){
      s = s + v[i] + " ";
   }
   s = s + "]";
   return s;
}
```

Come Arrays.toString(myarray)

Stampare il contenuto di un array come stringa

```
int[] a = {1,2,3};
int aSize = a.length;
System.out.println(ArrayUtil.printArray(a,aSize));
```

[1 2 3]

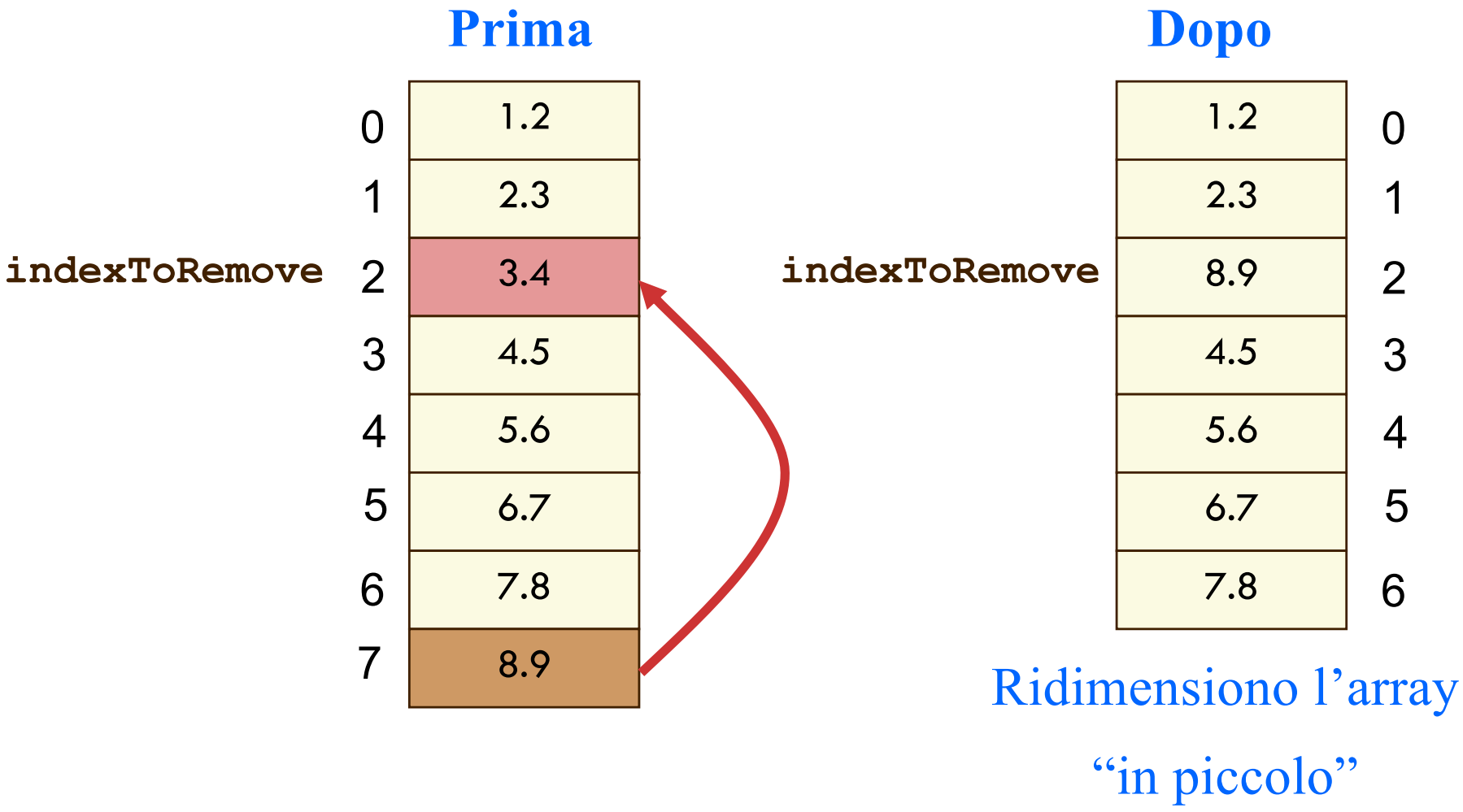
Eliminare/inserire un elemento

Eliminare un elemento

- L'eliminazione di un elemento da un array richiede due algoritmi diversi

  - **se l'ordine tra gli elementi dell'array non è importante** (cioè se l'array realizza il concetto astratto di insieme per quanto riguarda le posizioni), allora **è sufficiente copiare l'ultimo elemento dell'array nella posizione dell'elemento da eliminare e**
    - **ridimensionare l'array** se uso array riempiti completamente
    - **decrementare la dimensione** logica se uso la tecnica degli array riempiti soltanto in parte)

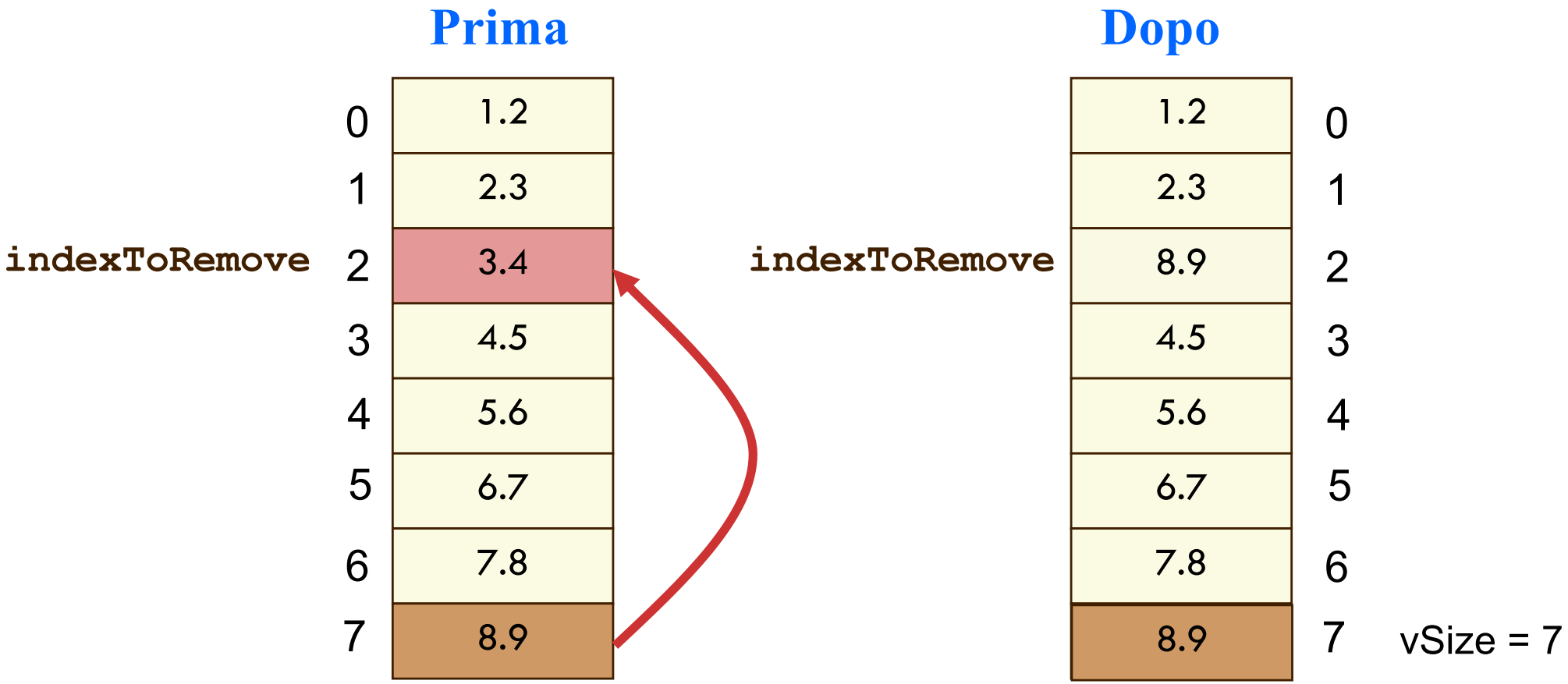
Eliminare un elemento: ordine degli elementi non importante



Ridimensiono l'array "in piccolo"

L'elemento 3.4 viene perso… se ho bisogno di usarlo devo salvarlo in una variabile prima di sovrascrivere la cella 2

Eliminare un elemento: ordine degli elementi non importante



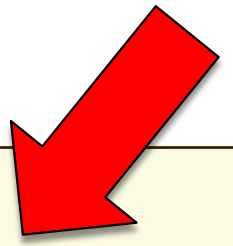
Se uso array riempito solo in parte
devo diminuire la dimensione logica

L'elemento 3.4 viene perso… se ho bisogno di usarlo devo salvarlo in una variabile prima di sovrascrivere la cella 2

Eliminare un elemento: ordine degli elementi non importante

```
double[] values = {1, 2.3, 4.5, 5.6};
int indexToRemove = 1;

values[indexToRemove] = values[values.length - 1];
values = resize(values, values.length - 1);

/*n.b. resize → metodo ausiliario
```

oppure

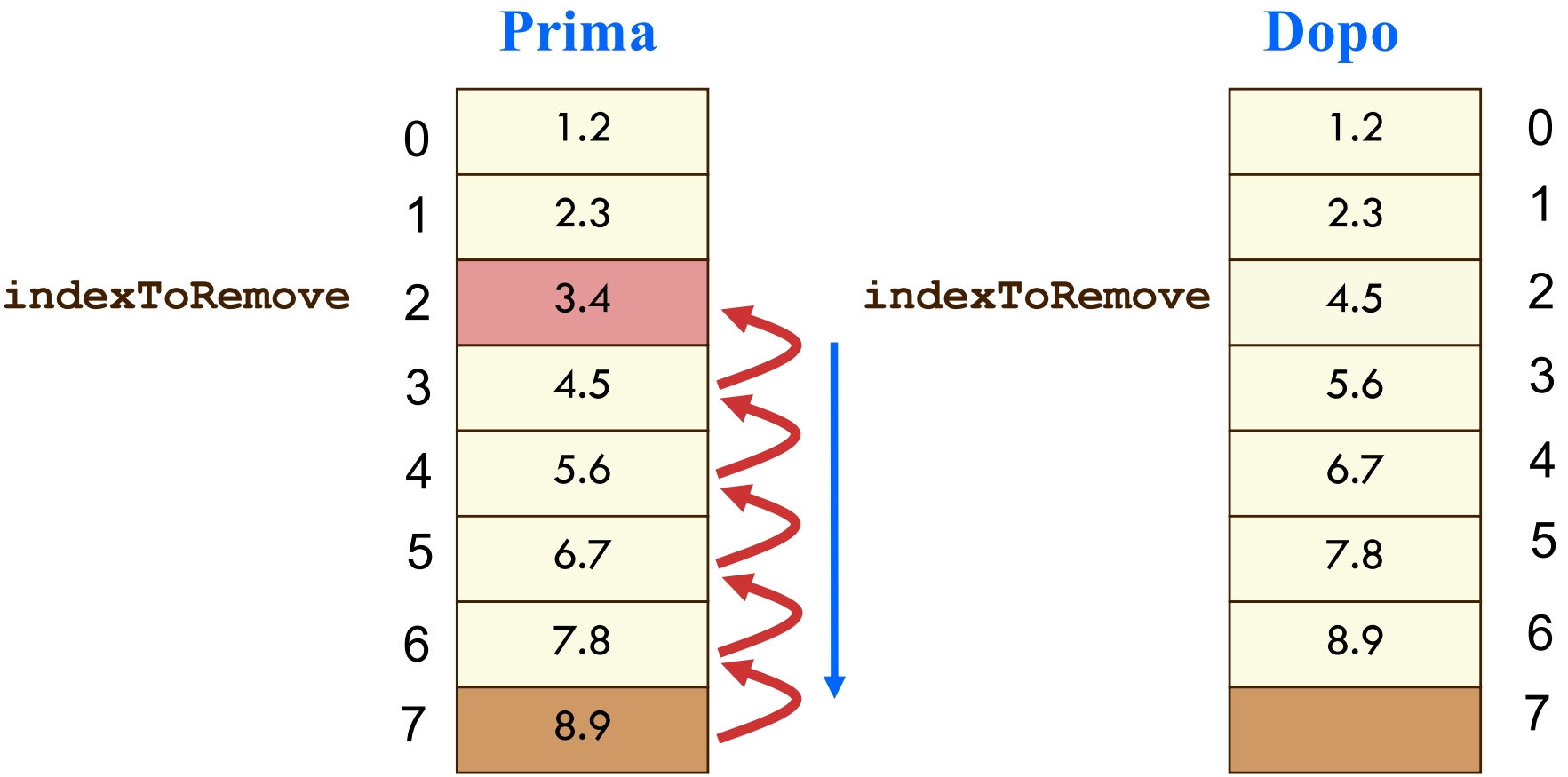


```
double[] values = {1, 2.3, 4.5, 5.6};
valuesSize = 4; // in questo caso e' pieno
int indexToRemove = 1;

values[indexToRemove] = values[valuesSize - 1];
valuesSize--;
```

Eliminare un elemento

- **Se l'ordine tra gli elementi deve, invece, essere mantenuto**, l'algoritmo è più complesso
  - **tutti gli elementi il cui indice è maggiore dell'indice dell'elemento da rimuovere devono essere spostati nella posizione con indice immediatamente inferiore,** per poi ridimensionare l'array (oppure usare la tecnica degli array riempiti soltanto in parte)
  - NB: si deve partire dalla posizione successiva all'indice dell'elemento da rimuovere e poi proseguire con indici crescenti

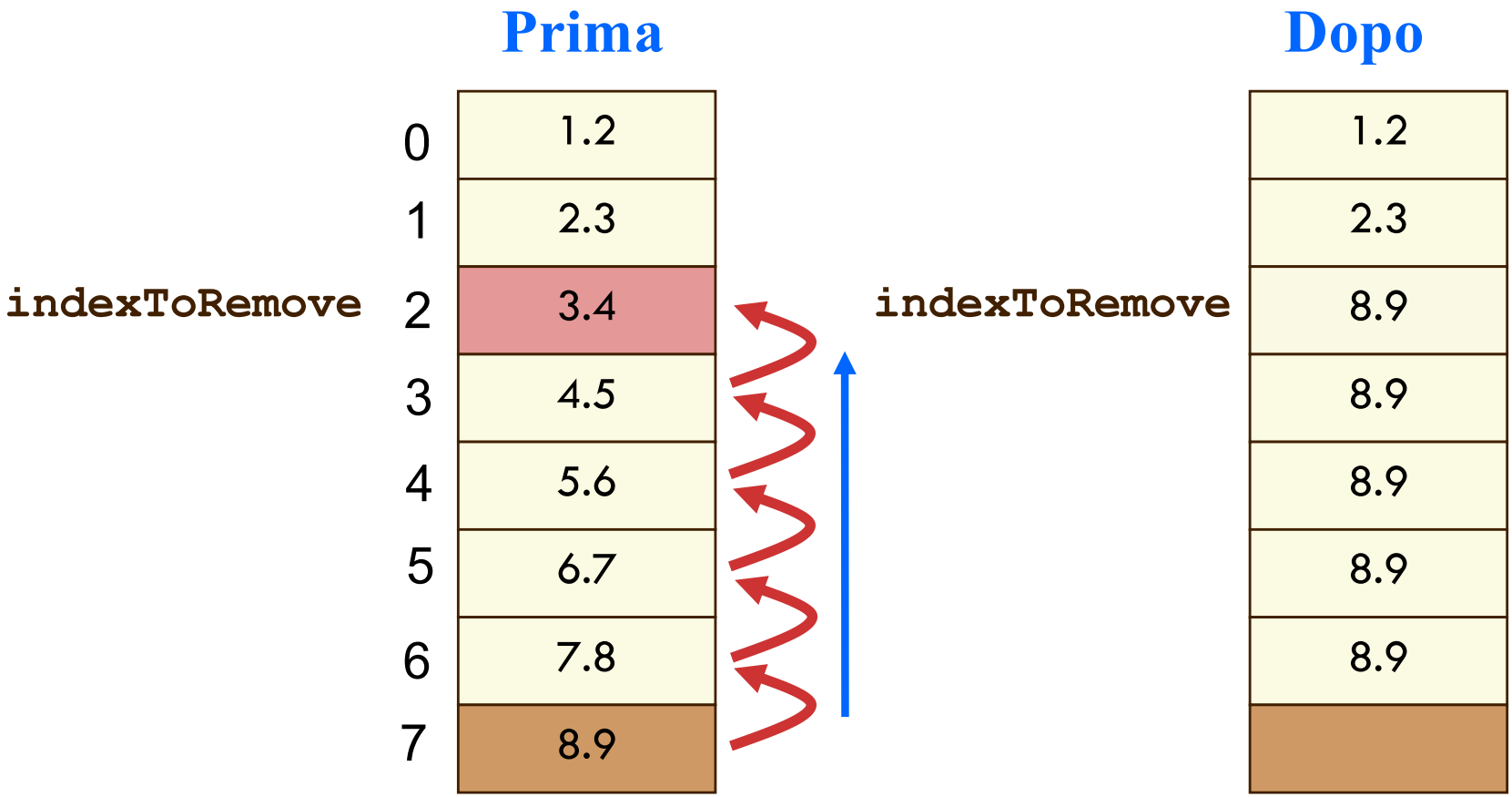
Eliminare un elemento: ordine degli elementi importante



L'elemento in posizione i+1 viene copiato in posizione i.
Nel ciclo si parte dalla posizione indexToRemove, e si va a crescere

Ridimensiono l'array oppure uso array riempito solo in parte

Cosa succederebbe se invertissi l'ordine



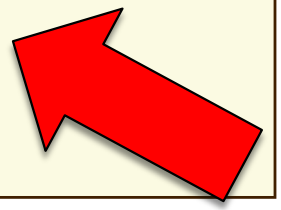
Ordine di trasferimento sbagliato
parto dall'ultima posizione e
descresco fino a indexToRemove!

Ridimensiono l'array
oppure uso array riempito
solo in parte

Codice corretto

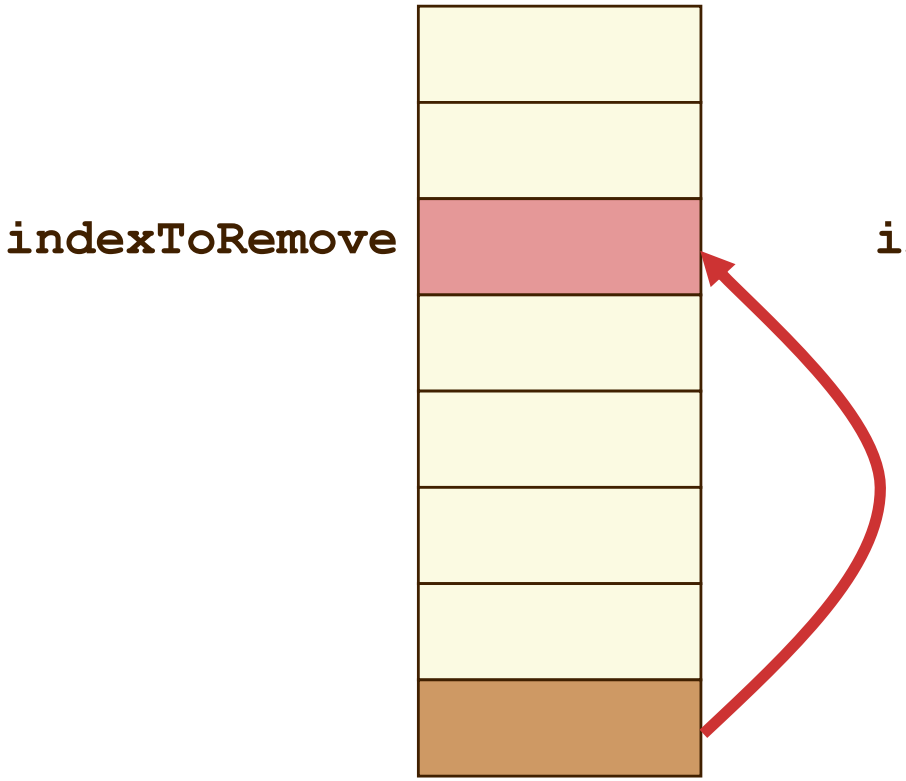
```
double[] values = {1, 2.3, 4.5, 5.6};

int indexToRemove = 1;

for (int i=indexToRemove; i<values.length-1; i++)
   values[i] = values[i + 1];

values = resize(values, values.length - 1);
```

```
double[] values = {1, 2.3, 4.5, 5.6};
int vSize = 4; //array pieno: dim logica = dim fisica

int indexToRemove = 1;
for (int i=indexToRemove; i<vSize-1; i++)
   values[i] = values[i + 1];

vSize--;
```

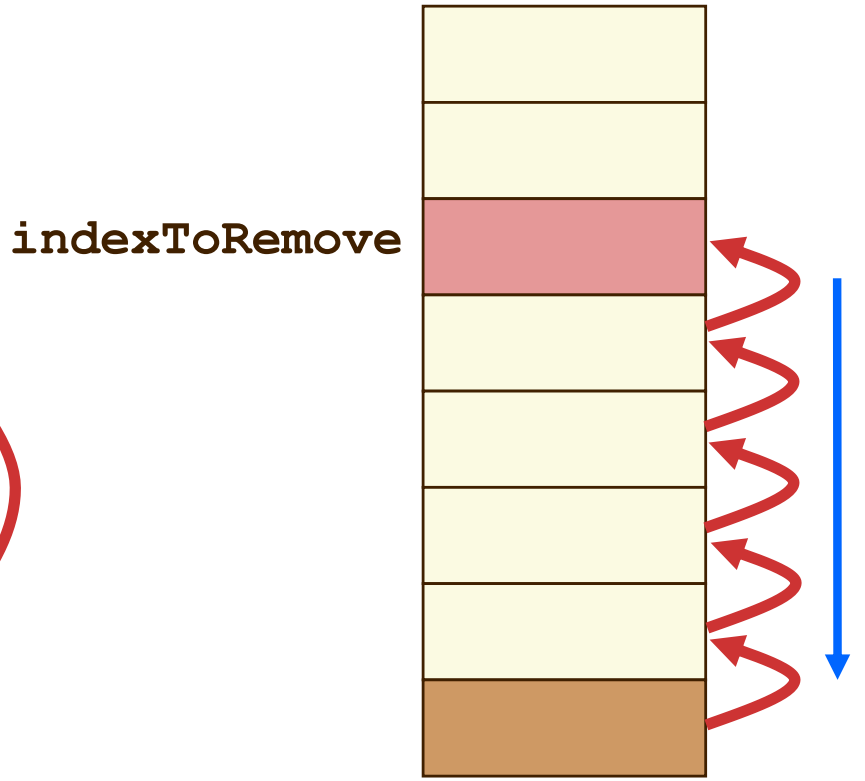


Eliminare un elemento: riassunto

Senza ordinamento



Con ordinamento



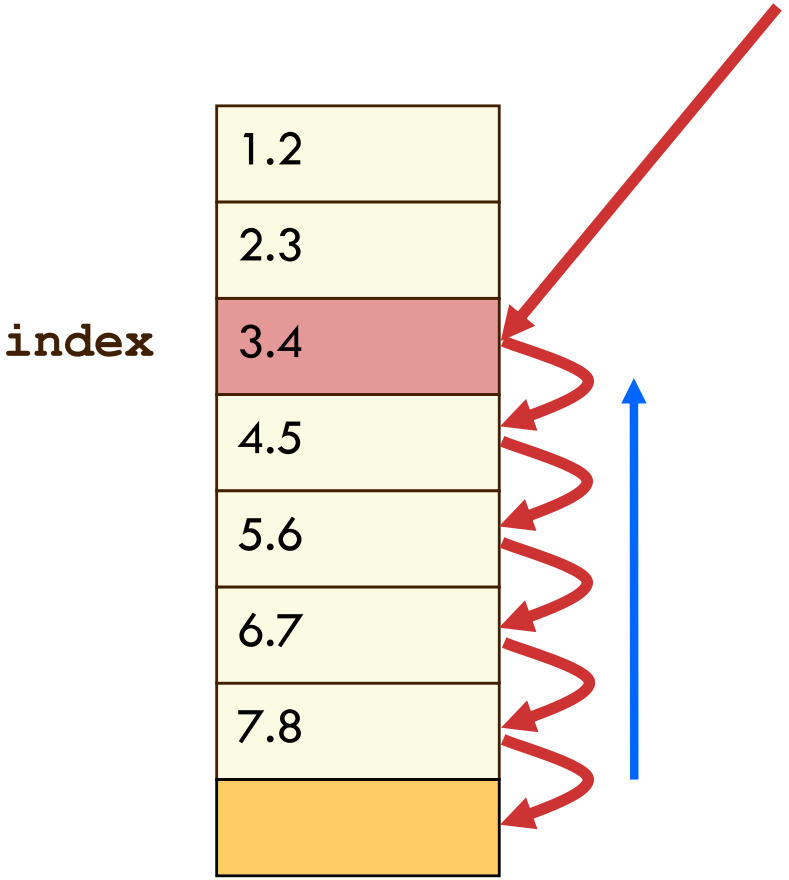
i trasferimenti vanno
eseguiti dall'alto in basso!

Inserire un elemento

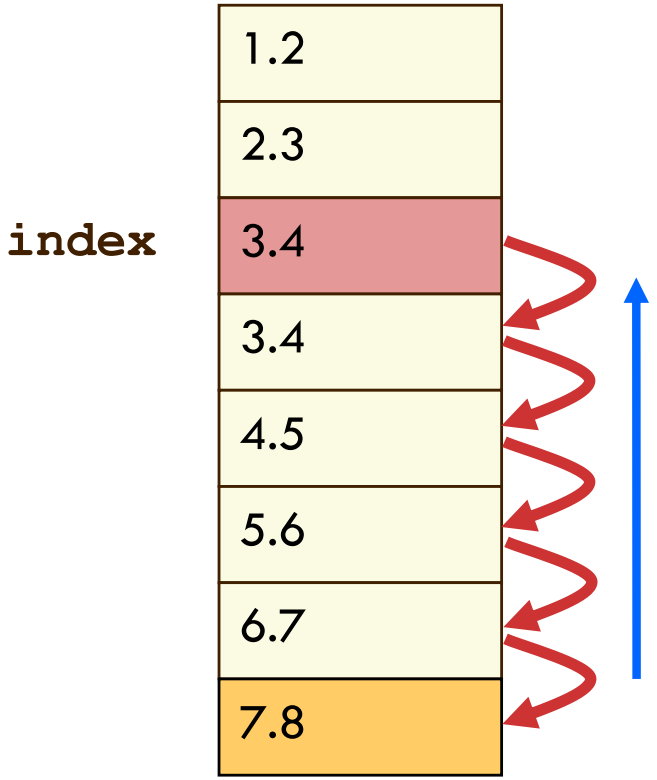
- Per inserire un elemento in un array nella posizione voluta, se questa non è la prima posizione libera, bisogna "fargli spazio"
  - tutti gli elementi il cui indice è maggiore dell'indice della posizione voluta devono essere spostati nella posizione con indice immediatamente superiore, ***a partire dall'ultimo elemento dell'array***

Inserire un elemento

Inserire in posizione 2 il valore 10.0

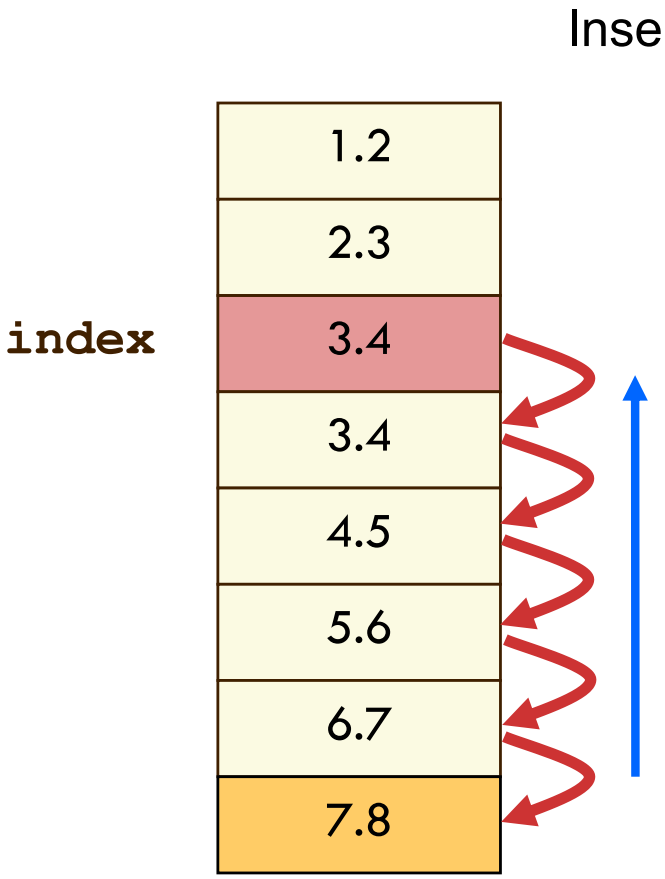


i trasferimenti vanno eseguiti dal basso in alto!



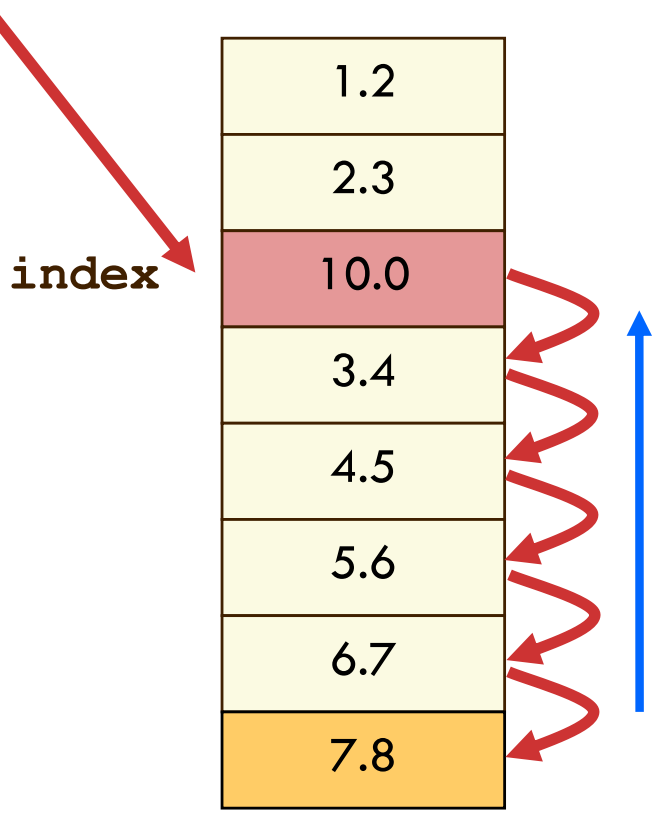
Ora posso sovrascrivere la posizione 2

Inserire un elemento



Ora posso sovrascrivere la posizione 2

Inserire in posizione 2 il valore 10.0

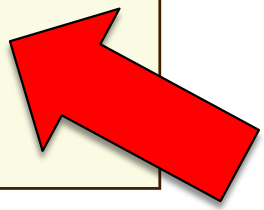


i trasferimenti vanno eseguiti dal basso in alto!

Inserire un elemento - codice

```
double[] values = {1, 2.3, 4.5, 5.6};
values = resize(values, values.length + 1);

int index = 2;
for (int i = values.length - 1; i > index; i--){
   values[i] = values[i-1];
}
values[index] = 5.4;
```

```
double[] values = {1, 2.3, 4.5, 5.6, 0.0, 0.0};
int vSize = 4;

int index = 2;
for (int i = vSize; i > index; i--){
   values[i] = values[i-1];
}
values[index] = 5.4;
vSize++;
```



Algoritmi di ricerca di elementi in un array

Ricerca di un valore in un array

- Idea algoritmica:
  - Per ogni elemento dell'array dal primo all'ultimo
    - Confronta l'elemento considerato con il target
    - Se uguali, termina la scansione
      - == per dati primitivi
      - Metodo equals(), eventualmente ridefinito, per oggetti
    - Altrimenti passa alla posizione successiva nell'array

Ricerca di un valore in un array

```
double[] v = new double[100];
int vSize = 0; double target; boolean found = false;

// leggo i dati, alla fine vSize contiene il num di elem
// e target l'elemento da cercare
. . .
for (int i = 0; i < vSize; i++){

   if (v[i] == target) {
      found = true;
      break;
   }
}

if(found){
   System.out.println("trovato");
}
else{
   System.out.println("non trovato");
}
```

Ricerca del valore minimo (massimo) in un array

- Idea algoritmica:
  - Assumi che il più piccolo sia il primo
  - Per ogni elemento dell'array dal secondo all'ultimo
    - Confronta l'elemento considerato con il minimo
    - Se minore, aggiorna il minimo
    - Passa alla posizione successiva nell'array

Ricerca del valore minimo (massimo) in un array

```
double[] v = new double[100];
int vSize = 0; double min;

// leggo i dati, alla fine vSize contiene il num di elem
. . .
if (vSize != 0){
   min = v[0]; // ipotesi da verificare…

   for (int i = 1; i < vSize; i++){
         if (v[i] < min) {
            min = v[i]; // nuova ipotesi...
         }
   }
   System.out.println("Il minimo e' : "+min);
}
else{
   System.out.println("Nessun dato e' stato introdotto");
}
```

Casi limite

- Bisogna sempre verificare il comportamento di questi algoritmi nei casi degeneri, tra i quali, ad esempio:
  - Cosa succede cercando il minimo/massimo in un array di lunghezza zero/uno?
  - Cosa succede cercando un elemento in un array di lunghezza zero/uno?

Varianti

- I vari metodi di ricerca che abbiamo visto si possono estendere alla ricerca di elementi che soddisfino una generica condizione, anziché la condizione

"essere uguale a **`target`**"

Varianti - esempi

- l'array contiene elementi negativi? (Sì/No)
- l'array contiene elementi dispari?
- Elementi che siano quadrati perfetti?
- Lo schema dell'algoritmo è sempre uguale… cambia soltanto la verifica

```
if (v[i] == target)
```

che diventa, ad esempio,

```
if (v[i] < 0)
```

oppure

```
if (v[i] % 2 != 0)
```

Varianti per verificare la presenza di uno specifico valore in un array

- L'algoritmo può essere realizzato in diverse varianti e inserito in un metodo che riceve il valore da cercare e l'array in cui cercare (in generale, riempito solo in parte), restituendo, in alternativa:
  - Un valore **boolean**: successo/insuccesso della ricerca
  - Un valore **int** che rappresenta l'indice della posizione in cui è stato trovato il valore cercato
    - Solitamente in caso di insuccesso restituisce **-1**
      oppure (più raramente, diciamo mai…) lancia un'eccezione
    - In caso di presenza di duplicati, restituisce una delle posizioni in cui si trova l'elemento cercato (di solito la posizione di indice minore)
  - Un array di **int** che contiene gli indici di tutte le posizioni in cui si trova l'elemento cercato
    - Array di lunghezza zero se l'elemento cercato non è presente
    - Non può essere "riempito solo in parte", deve essere "tutto pieno"

Varianti con ricerca della posizione del minimo

```
double[] v = new double[100];
int vSize = 0; int minPos;

// leggo i dati, alla fine vSize contiene il num di elem
. . .

if (vSize != 0){

   minPos = 0; // ipotesi da verificare…

   for (int i = 1; i < vSize; i++){
         if (v[i] < v[minPos]) {
            minPos = i; // nuova ipotesi...
         }
   }
   System.out.println("Il minimo e' : "+v[minPos]);
}
else{
   System.out.println("Nessun dato e' stato introdotto");
}
```

Trovare un elemento (restituendone LE posizioni)

```
public static int[] findMultiPos(double target,
                                 double[] v, int endIndex)
{  if (v==null || endIndex > v.length)
      return new int[0];

   int[] pos = new int[1]; // se serve, crescerà
   int posSize = 0; // riempito nella parte iniziale

   for (int i = 0; i < endIndex; i++)
   {  if (v[i] == target)
      {  if (posSize == pos.length) // deve crescere
         {   pos = resize(pos,2*posSize);
         }
         pos[posSize++] = i; // trovato ma NON mi fermo
      } // attenzione al ++ qui sopra
   }
   if (posSize == pos.length)
      return pos; // è già tutto pieno

   //spazio in eccesso, "taglio" e ritorno un array pieno
   return resize(pos, posSize);
}
```

Equivale a:
pos[posSize]=i;
posSize++;

Take home message

- Array: struttura dati che implementa l'ADT sequenza
  - Dimensione fisica
  - Dimensione logica: array riempiti in parte

- Algoritmi su array
  - Generazione numeri casuali
  - Convertire array in stringhe
  - Inserire/eliminare un elemento da un array (ordinato o non)
  - Ricerca di un elemento in un array