Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica

Programmazione I - corso B
a.a. 2009-10

prof. Viviana Bono

Blocco 7 – Array

Array

- Un array è una sequenza di celle di memoria consecutive (cioè poste a indirizzi consecutivi); in senso più astratto, è un contenitore composto, costituito da un numero prefissato di contenitori più semplici etichettati da numeri interi consecutivi (l'indice);
- un array è tale che la lettura e modifica dell' i-esimo componente può avvenire con i calcolabile dinamicamente, e in tempo costante indipendente da i;
- in Java, come in Pascal e in C, gli elementi di un array devono essere tutti dello stesso tipo, in modo da evitare errori di tipo durante l'esecuzione (perché?);
- in Java, a differenza che in Pascal o in C, gli array sono "oggetti" nel senso della programmazione a oggetti, e inoltre possono risiedere solo nell'area logica di memoria detta heap (cioè mucchio); gli array sono il primo esempio di oggetto Java che vediamo in questo corso.
- in Java tutti gli oggetti risiedono nello heap; non vi possono essere oggetti nello stack;
- nello stack vi possono invece essere, come vedremo, dei *riferimenti a oggetti*, cioè degli indirizzi di oggetti.

Array

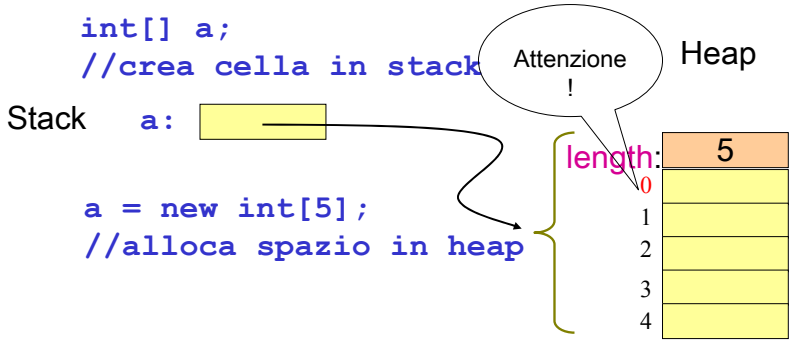
- in Java non esistono, in senso proprio, variabili di tipo array, e più in generale non esistono variabili di tipo oggetto;
- esistono solo variabili di tipo riferimento ad array o riferimento a oggetto; cioè variabili che possono contenere l'indirizzo di un array o di un oggetto, che "punta" a una cella della heap;
- per brevità si dirà ugualmente "una variabile di tipo array", intendendo una variabile di tipo riferimento ad array
- il tipo "riferimento ad array" non specifica il numero di elementi che l'array deve avere: una variabile di tipo `int[]` (cioè array di interi) può contenere l'indirizzo di un array di interi di lunghezza arbitraria
- la dichiarazione

```
int[] a;
```

di una variabile di tipo (riferimento ad) array NON crea l'array, ma soltanto una cella che PUÒ contenere un INDIRIZZO di array;

- l'oggetto-array, come qualunque oggetto, viene creato con una istruzione `new` (vedi esempi successivi, e libri di testo).

Ogni oggetto-array possiede un campo `length` contenente sempre il numero di elementi che compongono l'array; tale campo non può venire modificato.
Nell'esempio, il valore di `a.length` è 5



Inizializzare un array

- Nel caso di array di tipi elementari, all'atto della creazione vengono assegnati valori di default
  - *0* per i numeri (interi e reali)
  - *false* per i valori logici
  - *'\000'* per i caratteri
- Nel caso di array di tipi oggetto (es. String, Scanner,…, ma anche array stesso!), il valore di default è *null*

Creazione di un array con inizializzazione

Un array può essere contemporaneamente creato e inizializzato con una sequenza di espressioni, come nell'esempio seguente:

```
int[] arr = {3, 5, 15, 9};
```

è una scrittura concisa che equivale a:

```
int[] arr = new int[4];
arr[0]= 3; arr[1]= 5; arr[2]= 15; arr[3]= 9;
```

Tale forma di creazione con inizializzazione non è possibile come assegnazione a variabili di tipo array già dichiarate:

```
int[] arr;
arr = {3, 5, 15, 9};
```

ERRORE DI COMPILAZIONE!

Attenzione: si può invece scrivere:

```
arr = new int[] {3, 5, 15, 9};
```

Attenzione!

La lunghezza di un array può **non** essere nota staticamente; ad esempio:

...

```
int n = tastiera.nextInt();
double[] a = new double[n];
```

Ma attenzione:

un array, una volta creato, non può più variare la sua lunghezza.

Non si può fare:

```
double[] a = {3.14, 2.71, 0.25}
a[5] = 3.14;
```

(**errore** all'esecuzione)

oppure qualcosa come allunga(a) o accorcia(a).

È invece possibile memorizzare nella variabile-riferimento l'indirizzo di un diverso array di diversa lunghezza, ad es.:

a = {5, 4.6, 2.7, 4, 9, 26, 41};

Percorrere un array

Gli elementi di un array di lunghezza $\mathcal{N}$ hanno indici da 0 a $\mathcal{N}-1$; il tipico ciclo per percorrere interamente un array ha pertanto la forma:

```
...
n = a.length;
for(int i = 0; i < n; i++) {
  // accedi ad a[i] e fa' le azioni opportune
}
```

NB Essendo i un intero, le espressioni

```
i < n
```

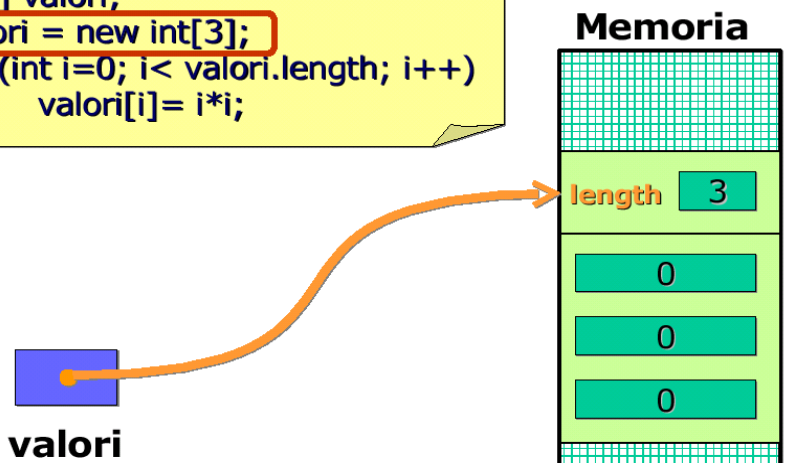
e

```
i <= n-1
```

sono equivalenti; tuttavia in Java, come in C, si preferisce la prima perché piú concisa.

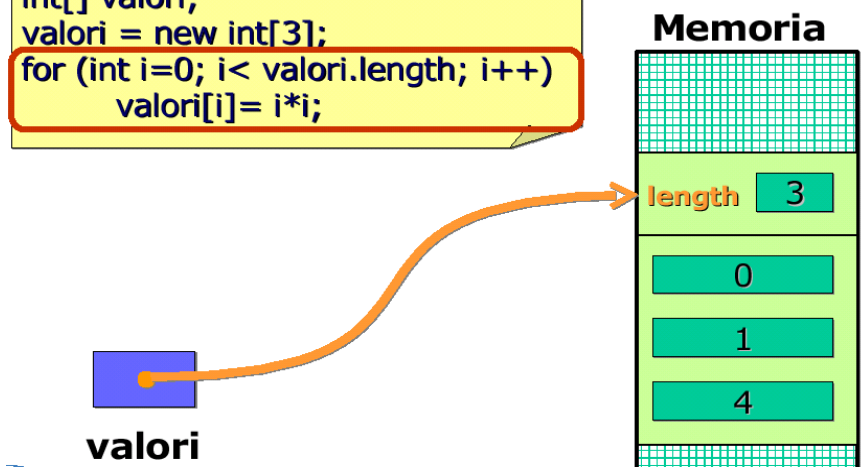
Esempio

```
int[] valori;
valori = new int[3];
for (int i=0; i< valori.length; i++)
        valori[i]= i*i;
```



Esempio (cont.)

```
int[] valori;
valori = new int[3];
for (int i=0; i< valori.length; i++)
        valori[i]= i*i;
```



Esempio: costruzione di un array con immissione dei suoi elementi da tastiera

```
public static void main(String args) {
  int[] unArray;
  out.print("lunghezza dell'array: ");
  Scanner input = new Scanner(System.in);
  int n = input.nextInt();
  unArray = new int[n];
  for(int i=0; i<n; i++) {
    out.println("immetti elemento n. " + i + ": ");
    unArray[i] = input.nextInt("elemento n. " + i + ": ");
  }
  ...
}
```

Nota

In questa versione elementare non viene effettuato nessun controllo sull'input.

Visualizzazione di un array su video

```
...
double[] a;
...
int n = a.length;
for(int i=0; i < n; i++)
  System.out.print(a[i] + " ");
System.out.println();
```

Nota: memorizzare la lunghezza del vettore in una variabile locale `n` è leggermente più efficiente che andarsela a reperire ogni volta nell'oggetto array, come nella versione seguente:

```
for(int i=0; i < a.length; i++)
  System.out.print(a[i] + " ");
```

Advanced topic: nuova forma di for in Java 1.5

```
double[] arr = {3.14, 2.71, 32, 0.75};

for(double x:arr)
 System.out.print(x + " ");
```

si legge: for each double x in arr ...

cioè: per ciascun double x in arr ...

NB Con questa nuova forma non si ha il controllo dell'indice; ad esempio non è possibile visualizzare un array mostrando anche l'indice di ogni elemento: a questo fine occorre usare la forma tradizionale, ad es.:

```
int n = a.length;
for(int i = 0; i < n; i++)
  System.out.println("n. " + i + ": " + a[i]);
```