

# 18 - Vettori

Programmazione e analisi di dati  
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa  
<http://pages.di.unipi.it/milazzo>  
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica  
A.A. 2016/2017

# Strutture dati

Una **struttura dati** è un'entità usata per organizzare un insieme di dati

L'unica struttura dati che abbiamo visto fino ad ora sono gli **array**

- Un array organizza dati **omogenei** (dello stesso tipo) come in un elenco

La struttura dati array **non è dinamica**

- La dimensione è fissata al momento della creazione

In certe situazioni, questo complica i programmi

# Array e dinamicità (1)

**Esempio:** scrivere un programma che chiede all'utente di inserire una sequenza di numeri terminata da 0, e poi stampa tutti i numeri inseriti

**Problema:** dove memorizzare i valori inseriti?

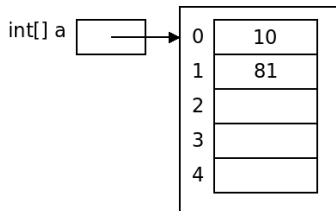
- quanti saranno?
- se si usa un array, di che dimensione crearlo?

**Soluzione (complicata):** Si crea un array di una certa dimensione, quando è pieno lo si sostituisce con uno più grande

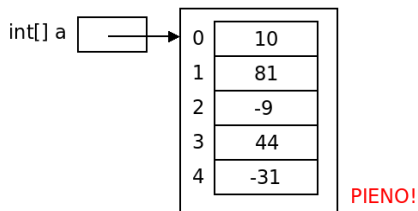
## Array e dinamicità (2)

Graficamente:

- Si crea un array e si inizia a inserire un valore dopo l'altro ....



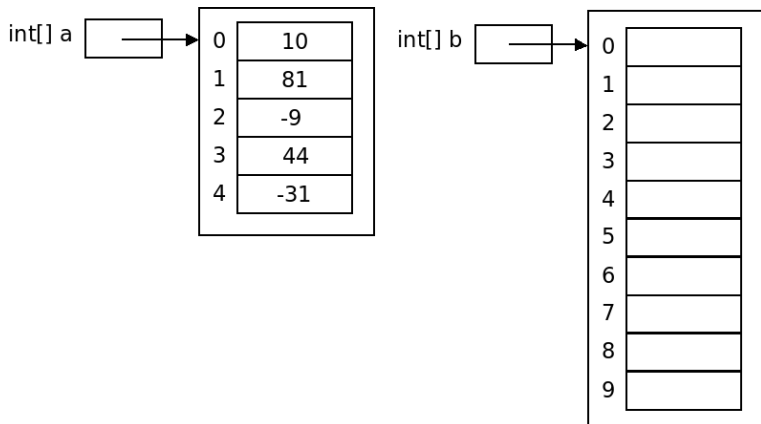
.....



## Array e dinamicità (3)

Graficamente:

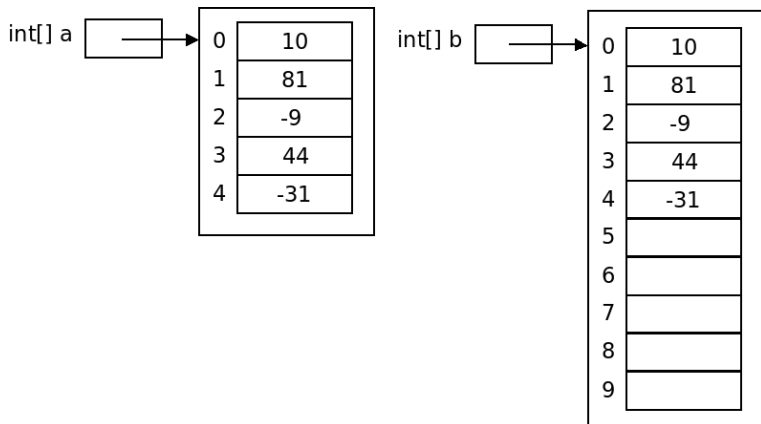
- .... quando l'array è pieno se ne crea uno più grande ....



## Array e dinamicità (4)

Graficamente:

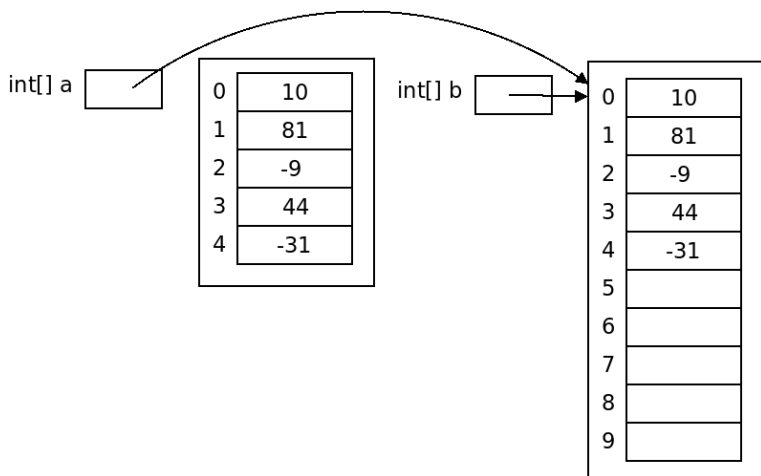
- .... si copiano i valori nel nuovo array ....



## Array e dinamicità (5)

Graficamente:

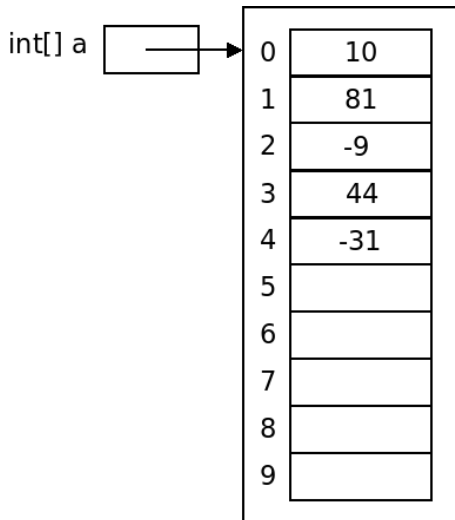
- .... si sostituisce il nuovo array al vecchio (riassegnando a)....



## Array e dinamicità (6)

Graficamente:

- .... e il gioco è fatto!





```
import java.util.Scanner;
public class StampaNumeri {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        // array inizialmente di dimensione 5
        int[] a = new int[5];
        // indice/contatore dei numeri letti
        int numeriLetti = 0;
        int n;
        do {
            // legge un valore
            n = input.nextInt();
            if (n!=0) {
                // se l'array non e' pieno inserisce il valore
                if (numeriLetti<a.length) {
                    a[numeriLetti] = n;
                    numeriLetti++;
                } else { // altrimenti crea un nuovo array piu' grande
                    int[] b = new int[numeriLetti+5];
                    // copia il contenuto del vecchio nel nuovo
                    for (int i=0; i<numeriLetti; i++) b[i] = a[i];
                    a = b; // sostituisce il nuovo al vecchio
                    // inserisce il valore letto
                    a[numeriLetti] = n;
                    numeriLetti++;
                }
            }
        } while (n!=0);
        // stampa tutto
        for (int i=0; i<numeriLetti; i++) System.out.println(a[i]);
    }
}
```

# Strutture dati dinamiche

Per rendere più facile la vita del programmatore, Java prevede alcune **strutture dati dinamiche**

- Ossia: strutture dati la cui dimensione può variare nel tempo

Tali strutture dati sono implementate come **classi** della **Libreria Standard di Java**

- Le operazioni su tali strutture dati si realizzano tramite invocazioni di opportuni **metodi**

La **documentazione** della Libreria Standard spiega il funzionamento di tutti i metodi di queste classi

- vedere il link alle API specifications nella pagina web del corso

Tutte queste classi fanno parte del package **java.util** (dovremo importarle)

# La classe Vector (1)

Una versione “dinamica” degli array (detti anche **vettori**) è fornita dalle classi

- **ArrayList**
- **Vector**

Tali classi sono molto simili tra loro (la differenza principale si nota solo nei programmi che prevedono parallelismo – non lo vedremo)

- consideriamo la classe **Vector**

## La classe Vector (2)

Per creare un vettore bisogna creare un oggetto della classe Vector

- La classe Vector è **generica**: si può specificare il tipo degli elementi tra parentesi angolari `< ... >`
- Non è necessario specificare la dimensione: all'inizio il vettore è vuoto.

```
Vector<String> v = new Vector<String>(); // vettore di stringhe
```

Gli elementi possono essere scritti e letti con i metodi **set** e **get**:

```
v.set(0, "Ciao"); // scrive un elemento in posizione 0
v.set(1, "Hello"); // scrive un elemento in posizione 1
v.get(1); // legge l'elemento in posizione 1
```

Si possono aggiungere elementi **in fondo** al vettore tramite il metodo **add**:

```
v.add("Bye"); // aggiunge un elemento in ultima posizione
```

## La classe Vector (3)

Si può inoltre ottenere la **dimensione** del vettore (numero di elementi contenuti) tramite il metodo **size**:

```
if (v.size() < 10) {...} // verifica dimensione del vettore
```

Inoltre vi sono decine di altri metodi per le operazioni più svariate (**leggete la documentazione**):

- Verificare se il vettore è vuoto (`v.isEmpty()`)
- Aggiungere un elemento in mezzo al vettore (`v.add(5, "Hi")`)
- Rimuovere un elemento in mezzo al vettore (`v.remove(5)`)
- Vedere se un elemento è presente nel vettore (`v.contains("Hi")`)
- Ottenere l'indice di un elemento (`v.indexOf("Hi")`)
- Aggiungere un gruppo di elementi al vettore (`v.addAll(v2)`)
- ....

Infine, i cicli `for-each` funzionano anche su vettori:

```
for (String s: v)  
    System.out.println(s);
```

## La classe Vector (4)

La classe Vector si aspetta come tipo per i suoi elementi (tra parentesi angolari) un **tipo classe**

Come fare per usare un vettore con elementi di tipo primitivo (int, double, ....) ?

- Bisogna usare le **classi involucro**
  - ▶ Integer per int
  - ▶ Double per double
  - ▶ Long per long
  - ▶ Character per char
  - ▶ Boolean per boolean
- Java si occuperà di trasformare automaticamente i valori dei tipi primitivi nei rispettivi oggetti (**autoboxing**) e viceversa (**autounboxing**)

```
Vector<Integer> vettore = new Vector<Integer>();  
vettore.add(5); // 5 viene trasformato nel corrispondente oggetto  
int x = vettore.get(0) // l'oggetto viene trasformato in intero
```

## La classe Vector (5)

Riprendiamo l'esempio della lettura e stampa dei numeri

```
import java.util.Vector;
import java.util.Scanner;

public class StampaNumeri2 {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        // vettore inizialmente vuoto
        Vector<Integer> v = new Vector<Integer>();

        int n;
        do {

            n = input.nextInt(); // legge un valore
            if (n!=0) v.add(n); // lo inserisce nel vettore

        } while (n!=0);

        // stampa tutto
        for (int x : v) System.out.println(x);
    }
}
```

## La classe Vector (6)

Modifichiamo l'esempio:

- Stampa solo i numeri in posizioni pari del vettore

```
for (int i = 0; i<v.size(); i+=2) {  
    System.out.println(v.get(i));  
}
```

- Azzera i numeri in posizioni pari del vettore

```
for (int i = 0; i<v.size(); i+=2) {  
    v.set(i,0);  
}
```



## La classe Vector (7)

La classe Vector (come la maggior parte delle classi della Libreria Standard di Java) ridefinisce e utilizza in maniera corretta i **metodi di Object** quali `toString` e `equals`.

Ad esempio:

- ridefinisce **toString** in modo da poter **stampare facilmente** i vettori

```
System.out.println(v);
```

e utilizza il metodo `toString` degli elementi per stamparne il contenuto

- utilizza il metodo **equals** per **confrontare elementi** del vettore
  - ▶ ad esempio, nell'implementazione dei metodi `indexOf` e `contains`

# La classe Vector (8)

Esempio: uno zoo!

- Classe `Animale` che rappresenta un animale
- Classe `Zoo` che rappresenta uno zoo

## La classe Vector (9)

```
public class Animale {

    // specie e nome dell'animale (public per semplicita')
    public String specie;
    public String nome;

    // costruttore
    public Animale(String specie, String nome) {
        this.specie = specie;
        this.nome = nome;
    }

    // metodo toString
    public String toString() {
        return specie + " (" + nome + ")";
    }

    // metodo equals
    public boolean equals(Object o) {
        if (o instanceof Animale) {
            Animale a = (Animale) o;
            return (specie.equals(a.specie) && nome.equals(a.nome));
        }
        else return false;
    }
}
```

# La classe Vector (10)

```
import java.util.Vector;

public class Zoo {

    // uno zoo e' una collezione di animali (vettore)
    private Vector<Animale> animali;

    // costruttore (di uno zoo vuoto)
    public Zoo() {
        animali = new Vector<Animale>();
    }

    // inserisce un animale nello zoo (se non gia' presente)
    public void inserisci(Animale a) {
        if (!presente(a))
            animali.add(a);
    }

    // verifica se un certo animale e' presente
    public boolean presente(Animale a) {
        // contains usa il metodo equals di Animale
        return animali.contains(a);
    }
}
```

(segue)

## La classe Vector (11)

(segue Zoo)

```
// verifica se un animale di una certa specie e' presente
public boolean presentePerSpecie(String sp) {
    boolean trovato = false;
    for (Animale a : animali)
        if (a.specie.equals(sp)) trovato = true;
    return trovato;
}

// stampa l'elenco degli animali
public void visualizza() {
    // println usa il metodo toString di Vector e di Animale
    System.out.println(animali);
}
}
```

## La classe Vector (12)

E, infine, un main di prova

```
public class UsaZoo {  
  
    public static void main (String[] args) {  
  
        Zoo zoo = new Zoo();  
  
        Animale zebra1 = new Animale("Zebra","Gino");  
        Animale zebra2 = new Animale("Zebra","Fulvia");  
        Animale giraffa1 = new Animale("Giraffa","Alda");  
  
        zoo.inserisci(zebra1);  
        zoo.inserisci(zebra2);  
        zoo.inserisci(giraffa1);  
  
        // zebra3 e' uguale (equals) a zebra2  
        Animale zebra3 = new Animale("Zebra","Fulvia");  
  
        System.out.println(zoo.presente(zebra3));  
        System.out.println(zoo.presentePerSpecie("Giraffa"));  
        System.out.println(zoo.presentePerSpecie("Leone"));  
        zoo.visualizza();  
  
    }  
}
```

## La classe Vector (13)

Risultato dell'esecuzione:

```
true  
true  
false  
[Zebra (Gino), Zebra (Fulvia), Giraffa (Alda)]
```

# Altre strutture dati dinamiche

La Libreria di Java fornisce molte altre strutture dati dinamiche

Esempi (da vedere nella documentazione per approfondimento):

- **HashSet**: Descrive **insiemi** di elementi senza duplicati e senza un ordine predefinito
- **HashMap**: Descrive **dizionari**, ossia associazioni di chiavi-valori
  - ▶ Una **chiave** è un elemento che compare una sola volta nella struttura dati
  - ▶ Ad una chiave è associato un solo **valore**
  - ▶ Un **valore** può essere associato a più chiavi
- **Stack**: Descrive **pile** di valori
  - ▶ I valori si possono solo inserire (push) e rimuovere (pop)
  - ▶ L'ultimo valore inserito è il primo che viene rimosso (Last-In-First-Out – LIFO)