

## Array

Una matrice (**array**) è un *insieme ordinato e omogeneo* di dati: è una variabile che contiene uno o più valori in sequenza ordinata. Questi valori si chiamano "elementi" e devono avere lo stesso tipo di dato. I singoli elementi sono individuati tramite indici.

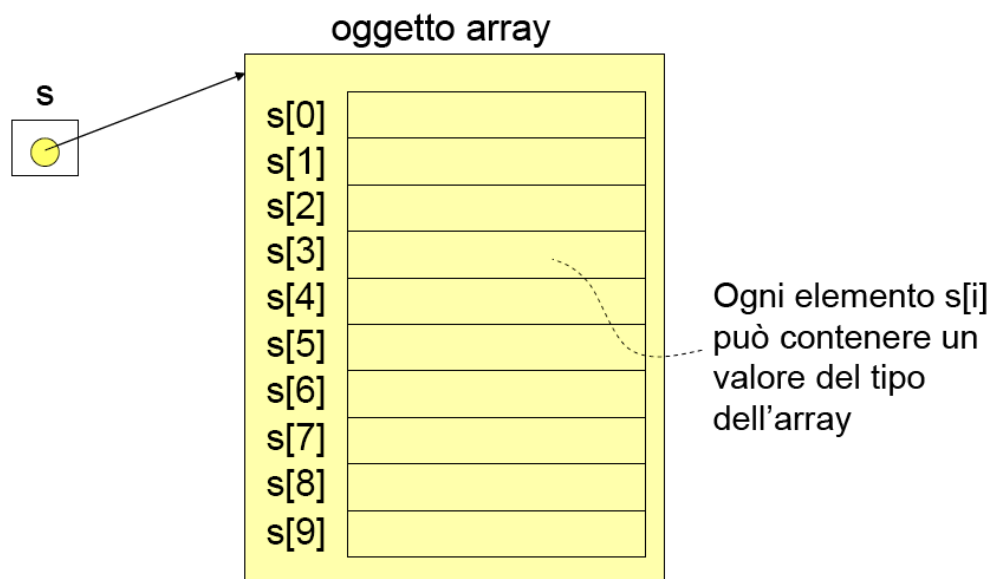
*Variabile di tipo strutturato*, dunque, in quanto l'indice introduce delle relazioni all'interno dell'insieme, stabilendo un nuovo ordine che non è necessariamente quello dei valori degli elementi componenti.

Il tipo dell'indice deve essere scalare, non reale perché, per ogni elemento, deve essere possibile definire "il precedente" ed "il seguente".

Il *nome* dell'insieme è l'*indirizzo* della locazione di memoria dove è memorizzato il valore del primo elemento; gli altri elementi della lista occuperanno locazioni di memoria consecutive.

In Java un **array** è un oggetto senza metodi, con attributo pubblico **length** che memorizza la dimensione dell'array

## Schema generico di un array



*Implementazione in linguaggio Java: array di lunghezza fissa*

La *sintassi* per dichiarare **matrice monodimensionale** una variabile è la seguente:

tipo nome []; oppure tipo [] nome;

Ad esempio, per dichiarare **array** la variabile di nome **s** si usa la seguente sintassi: `int s [];` o `int [] s ;` che definisce un insieme di elementi di tipo intero

Se l'array è già stato dichiarato, per crearne un nuovo oggetto si usa la sintassi:

nome = **new** tipo[*dimensione*];

Ad esempio, per creare un'istanza di **s** contenente 10 valori:

`s = new int [10] ;`

Per dichiarare l'**array** e contemporaneamente crearne un oggetto si usa la sintassi:

```
tipo nome [] = new tipo[dimensione];
                oppure
tipo [] nome = new tipo[dimensione];
```

Tale *dimensione* (il numero di elementi o *lunghezza*) non può essere modificata e può essere definita usando un valore costante o una variabile di valore già noto.

Ad esempio, per dichiarare **array** la variabile di nome **s** e crearne un'istanza contenente 10 valori:

```
int s [] = new int [10] ;
                o
int [] s = new int [10] ;
```

Lo stesso operatore ([]) si usa per accedere al singolo elemento a partire dal primo con indice 0 fino all'ultimo con indice **length** - 1. Ad esempio si può inizializzare il singolo elemento: **s[0] = valore;**

Il tentativo di accedere ad un elemento al di fuori del range (ad esempio s[10]), genera un'eccezione di **ArrayIndexOutOfBoundsException**.

Quando si usa l'operatore **new** gli elementi sono inizializzati automaticamente:

- **0** se elementi il tipo numerico (0 interi e 0.0 double o float)
- **false** se elementi il tipo booleano
- **null** se oggetti ad esempio String
- **\0** se elementi il tipo carattere

In alternativa, per dichiarare, creare ed inizializzare contemporaneamente un **array** si usa la sintassi:

```
tipo nome [] = { valore, ... };
```

che istanzia un array di dimensione pari al numero di elementi inseriti.

Ad esempio, per dichiarare **array** la variabile di nome **s**, crearne un'istanza contenente numeri interi di valore prestabilito, si usa la sintassi :

```
int s [] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Si può recuperare la dimensione del vettore grazie all'attributo opportuno: **s.length**

```
class CopiaRifArray {
    public static void main(String args[]) {
        int a1[] = {1,2,3,4,5}; // dichiarazione e inizializzazione
        int a2[]; // dichiarazione

        a2 = a1; // ora a1 e a2 si riferiscono allo stesso oggetto

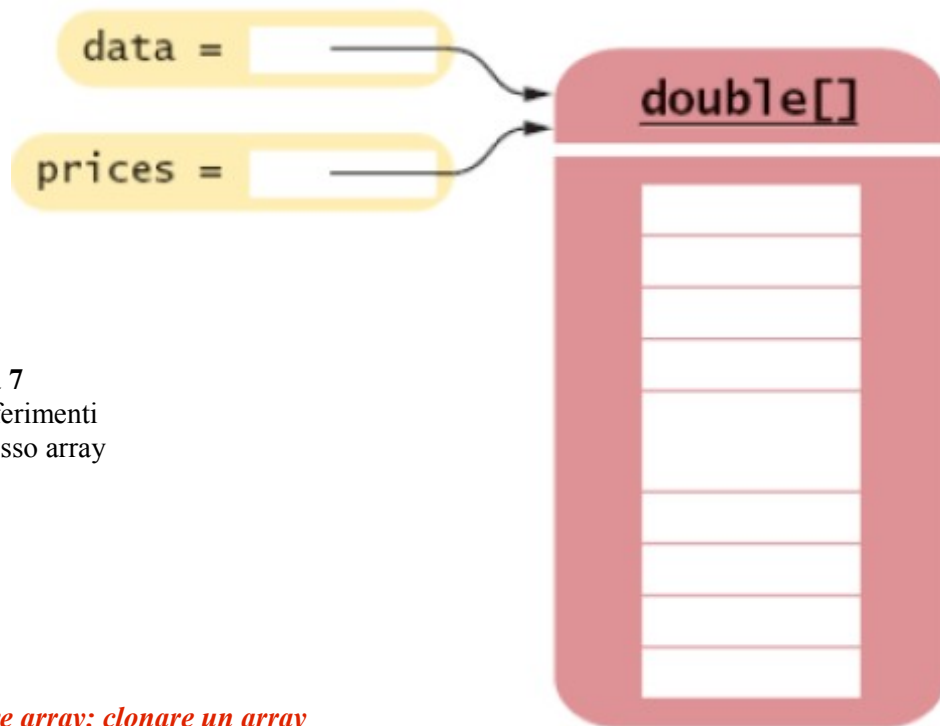
        for (int i=0; i<a2.length; i++)
            a2[i]++;
        System.out.println (" ");
        for (int i = 0; i < a1.length; i++)
            System.out.println("a1["+i+"] = " + a1[i]);

        // notare l'incremento del contenuto anche di a1[] infatti l'oggetto è lo stesso
    }
}
```

### Copiare array<sup>1</sup>: copiare il riferimento a un array

□ Una variabile di tipo array memorizza un riferimento all'array. Copiando la variabile si ottiene un secondo riferimento al medesimo array.

```
double[] data = new double[10];  
// riempire l' array . . .  
double[] prices = data;
```

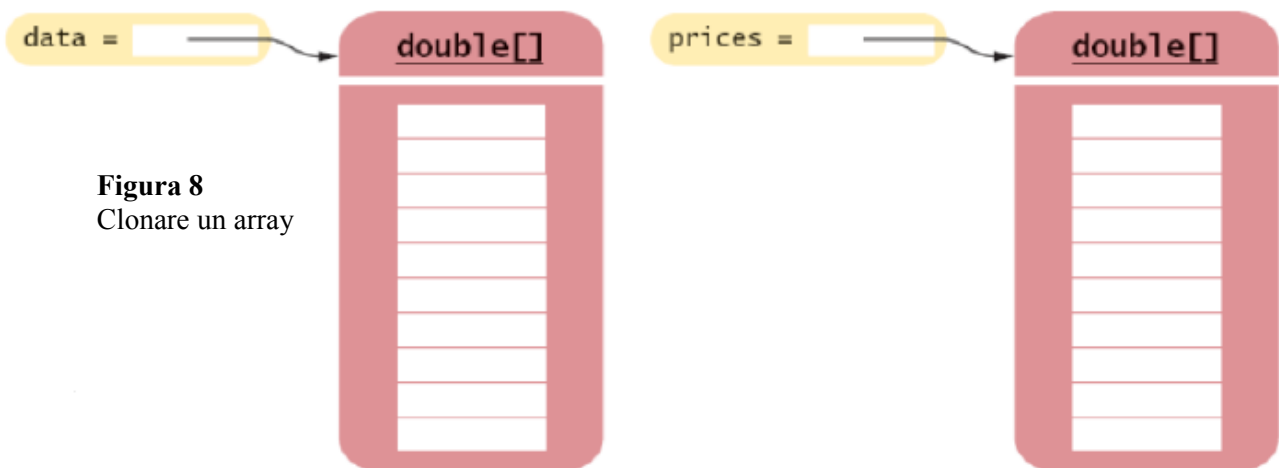


**Figura 7**  
Due riferimenti  
allo stesso array

### Copiare array: clonare un array

□ Per copiare gli elementi di un array (**duplicare**) usate il **metodo clone**.

```
double[] data = new double[10];  
double[] prices = (double[]) data.clone();
```



**Figura 8**  
Clonare un array

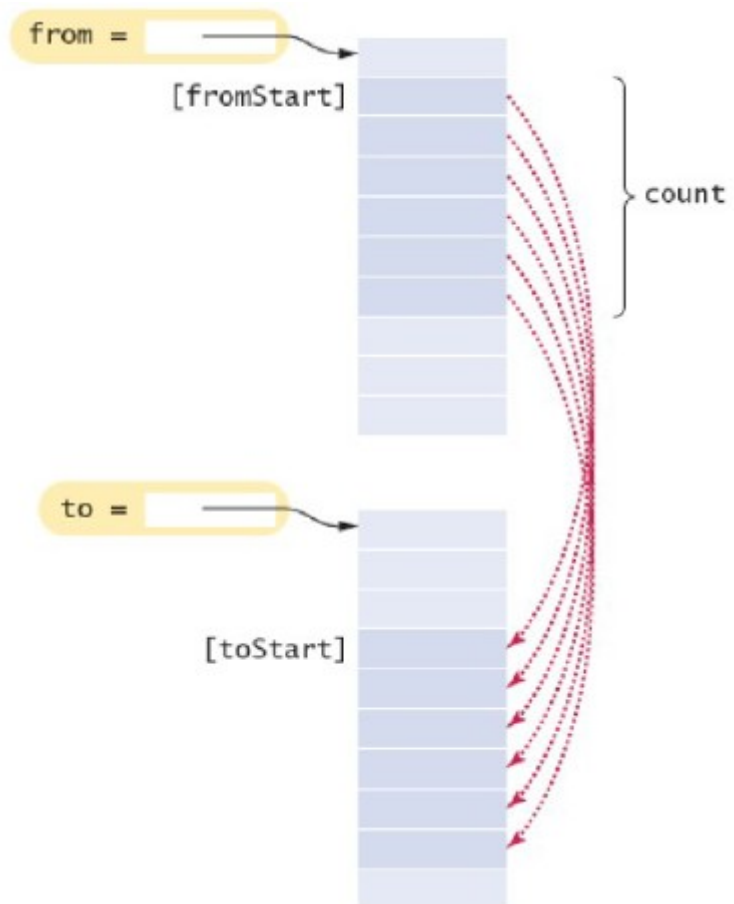
<sup>1</sup> C. Horstmann, “Fondamenti di Programmazione e Java 2” cap.8 , 3a ed., Apogeo

## Copiare array: copiare gli elementi di un array

- Usate il metodo `System.arraycopy` per **copiare** elementi da un array ad un altro.

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

**Figura 9**  
Il metodo  
`System.arraycopy()`  
per copiare



`arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`

## Elaborazioni su array

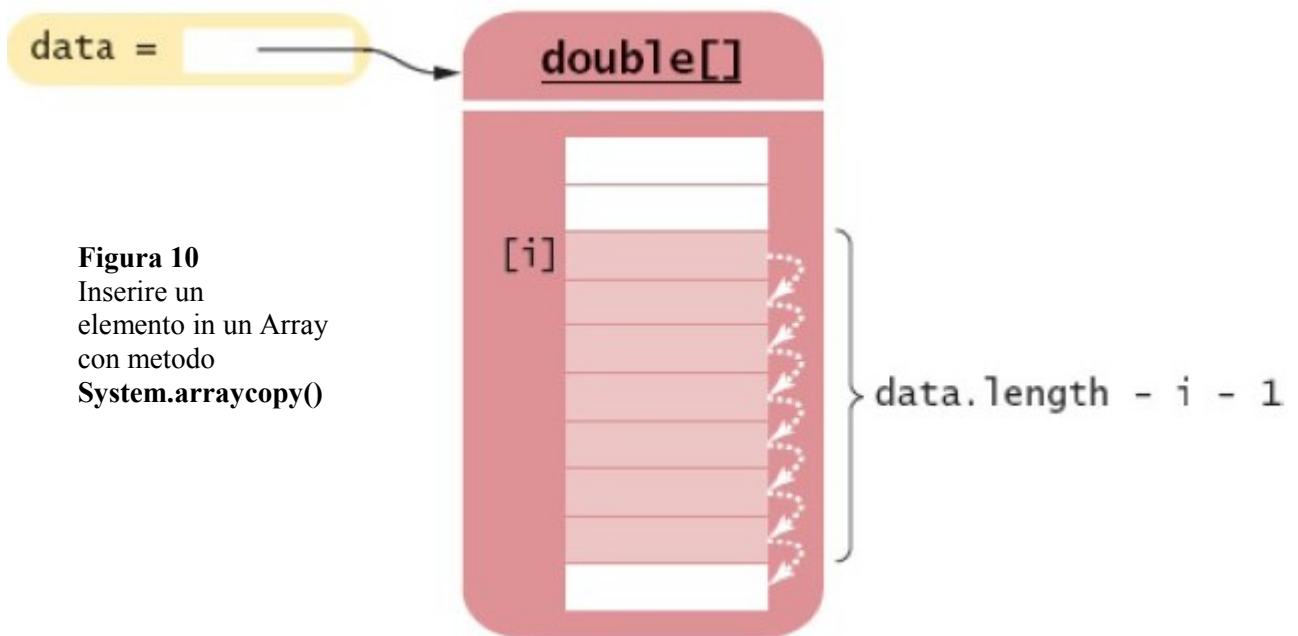
Si può ricorrere alla classe [Arrays](#) definita nel `package java.util` che rende disponibili metodi per:

- **riempimento** anche parziale
- **ricerca binaria**
- **ordinamento**

### *Inserire un elemento in un array*

- Usate il metodo `System.arraycopy` per **inserire** un elemento in un array

```
System.arraycopy(data, i, data, i + 1, data.length - i - 1);  
data[i] = x;
```

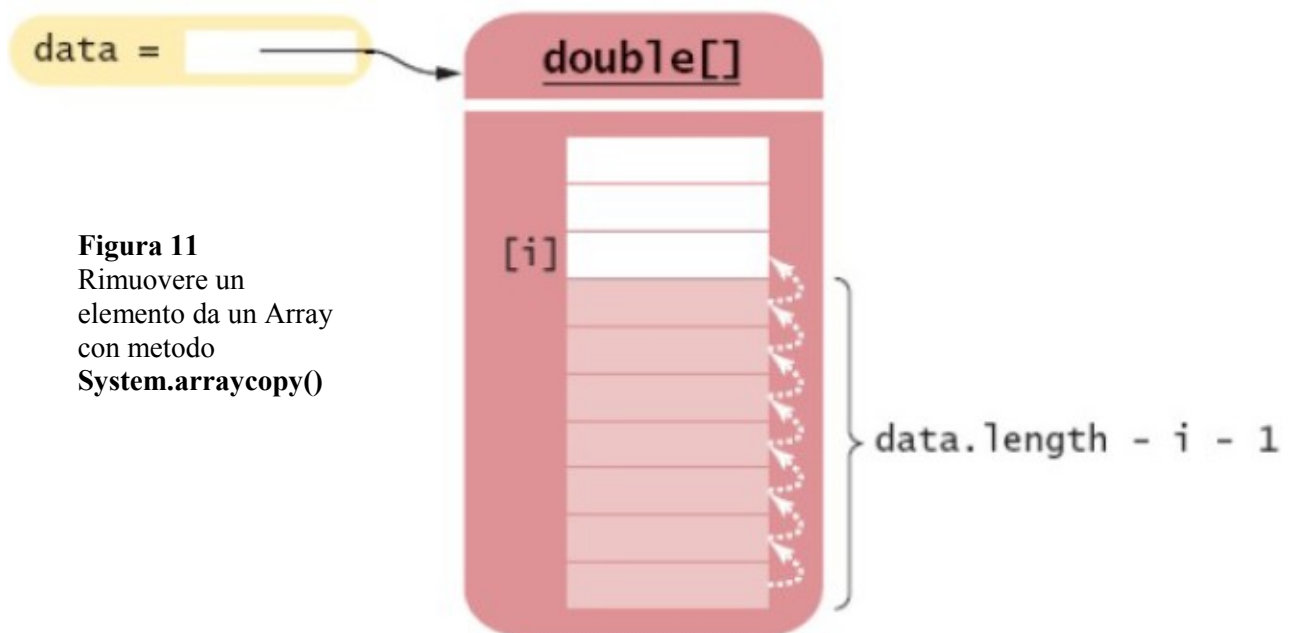


**Figura 10**  
Inserire un  
elemento in un Array  
con metodo  
`System.arraycopy()`

### *Rimuovere un elemento da un array*

- Usate il metodo `System.arraycopy` per **eliminare** un elemento da un array

```
System.arraycopy(data, i + 1, data, i, data.length - i - 1);
```



**Figura 11**  
Rimuovere un  
elemento da un Array  
con metodo  
`System.arraycopy()`

### *Far crescere un array*

□ Il metodo `System.arraycopy` viene anche utilizzato per **far crescere di dimensione** un array che non ha più spazio, seguendo queste fasi operative:

1. Create un nuovo array, di dimensione maggiore

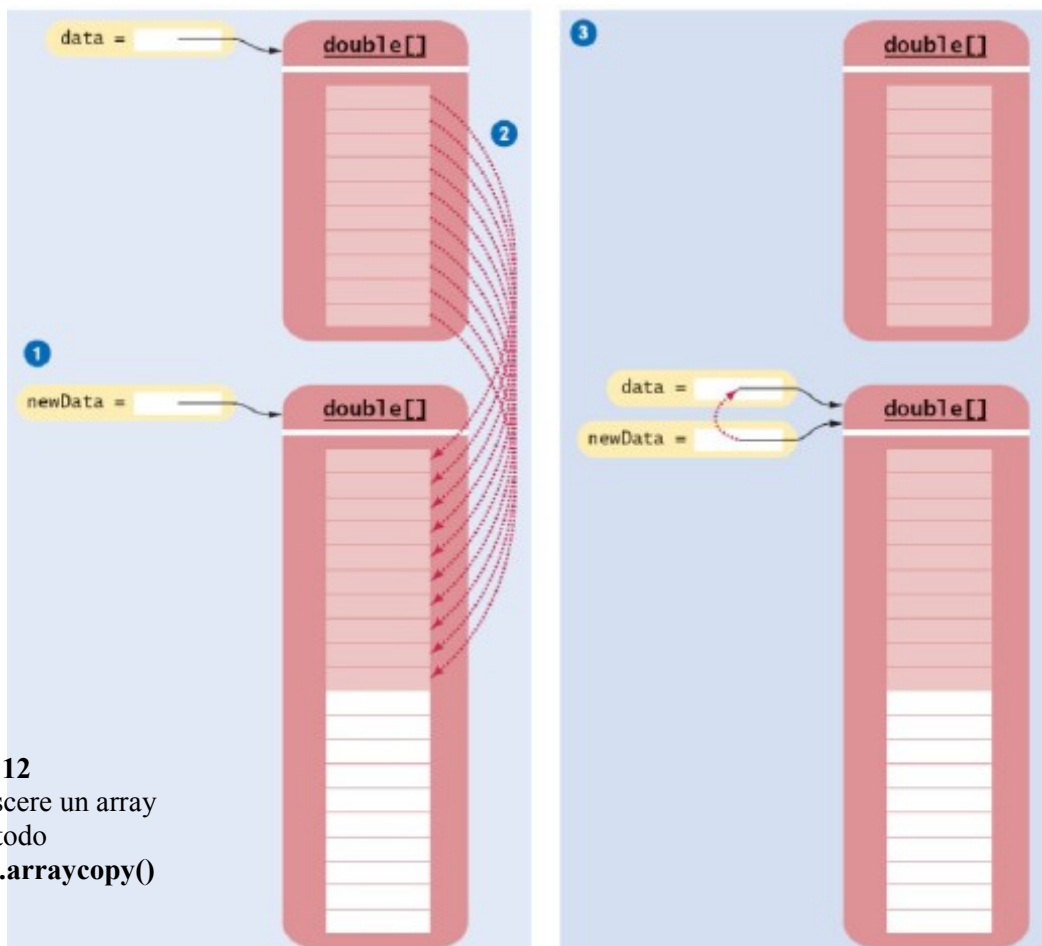
```
double[] newData = new double[2 * data.length];
```

2. Copiare tutti gli elementi nel nuovo array

```
System.arraycopy(data, 0, newData, 0, data.length);
```

2. Memorizzare nella variabile array il riferimento al nuovo array

```
data = newData;
```



**Figura 12**  
Far crescere un array  
con metodo  
`System.arraycopy()`

## Trasformare array paralleli in array di oggetti

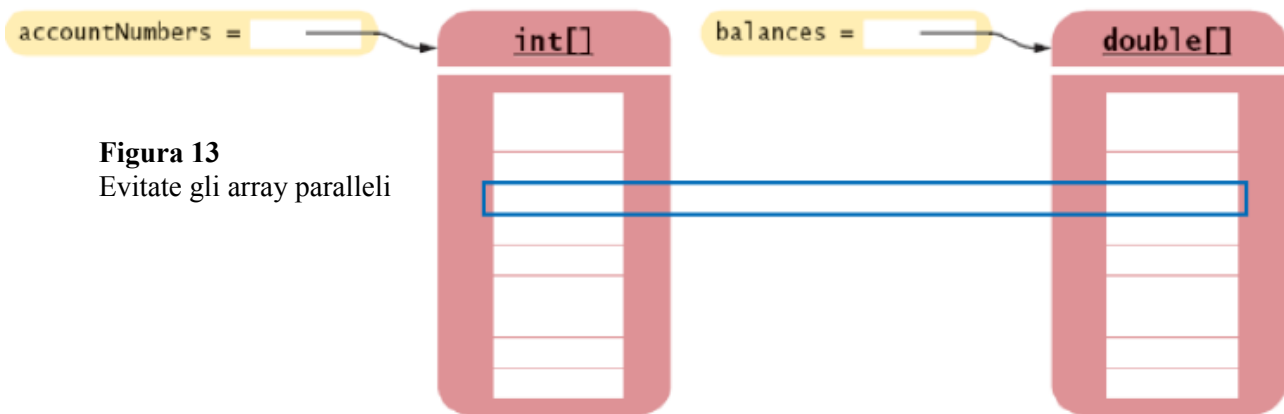
Una struttura dati è denominata “array paralleli” quando si usano *diversi array* per contenere i dati del problema e gli *elementi aventi lo stesso indice* nei diversi array sono tra loro *fortemente correlati* :

- devono sempre contenere lo *stesso numero di elementi*
- in questo caso, rappresentano *diverse proprietà dello stesso oggetto*
- molte elaborazioni hanno bisogno di *utilizzare tutti gli array*, che devono quindi essere passati come parametri.

Tutte le volte in cui il problema presenta una struttura dati del tipo “array paralleli”, si consiglia di *trasformarla in un array di oggetti* :

- occorre realizzare la classe con cui costruire gli oggetti
- risulta molto più facile scrivere il codice e, soprattutto, apportare modifiche

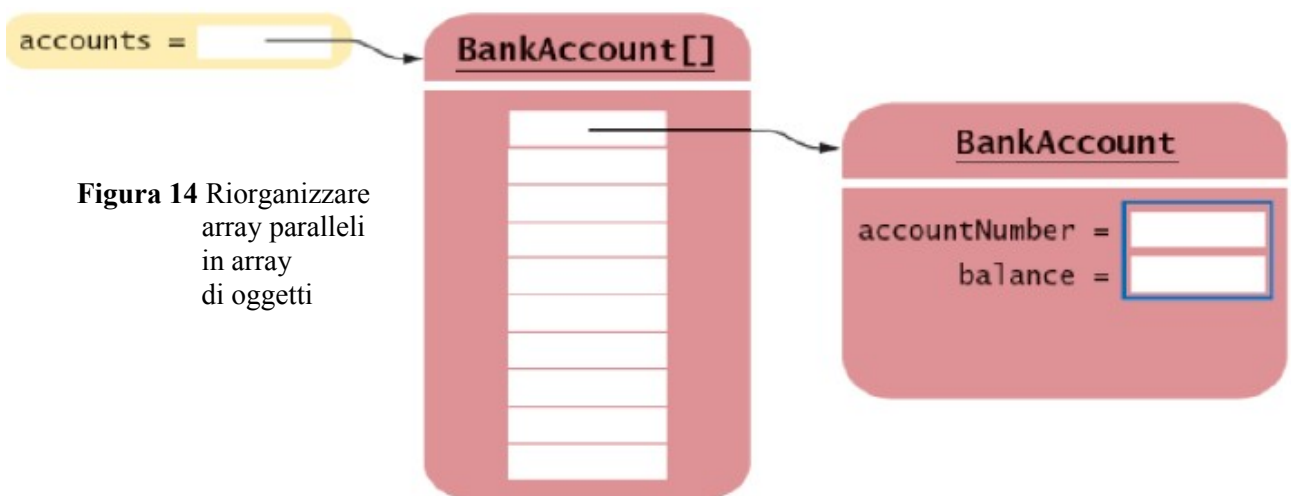
```
// non fare così  
int[] accountNumbers;  
double[] balances;
```



**Figura 13**  
Evitate gli array paralleli

□ Usare un **unico array di oggetti**

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



**Figura 14** Riorganizzare  
array paralleli  
in array  
di oggetti

### Array riempiti solo in parte

- La dimensione dell'array va impostata prima di sapere quanti sono gli elementi di cui si ha bisogno e non può più essere modificata.
- Si può creare un array che sia sicuramente più grande del numero massimo possibile di voci e poi riempirlo solo parzialmente.
- Usare una variabile complementare che dica quanti elementi dell'array sono realmente utilizzati.
- Assegnare sempre (con buono stile) a tale variabile complementare un nome ottenuto aggiungendo il suffisso Size al nome dell'array.

```
final int DATA_LENGTH = 100;
```

```
double[] data = new double[DATA_LENGTH];  
int dataSize = 0;
```

- `data.length` è la **capacità** dell'array `data`, mentre `dataSize` è la **dimensione reale** dell'array (Figura 15). Continuando ad aggiungere elementi all'array, bisogna incrementare di pari passo la variabile dimensione.

```
data[dataSize] = x;  
dataSize++;
```

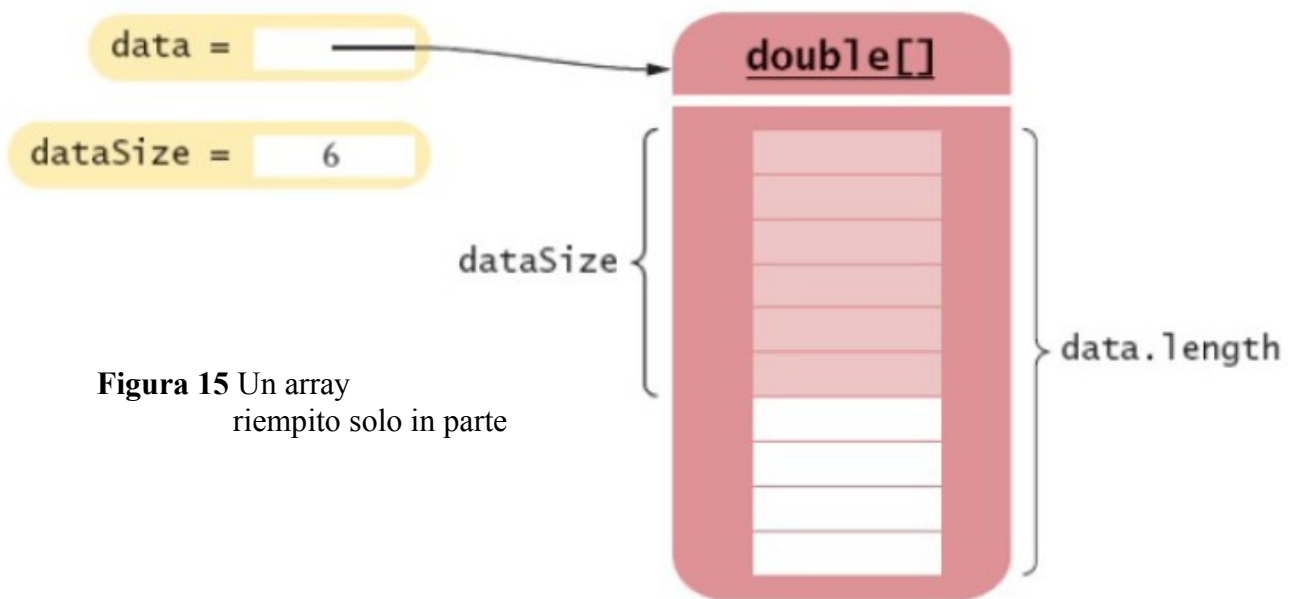


Figura 15 Un array riempito solo in parte

### Uno dei primi worm di Internet

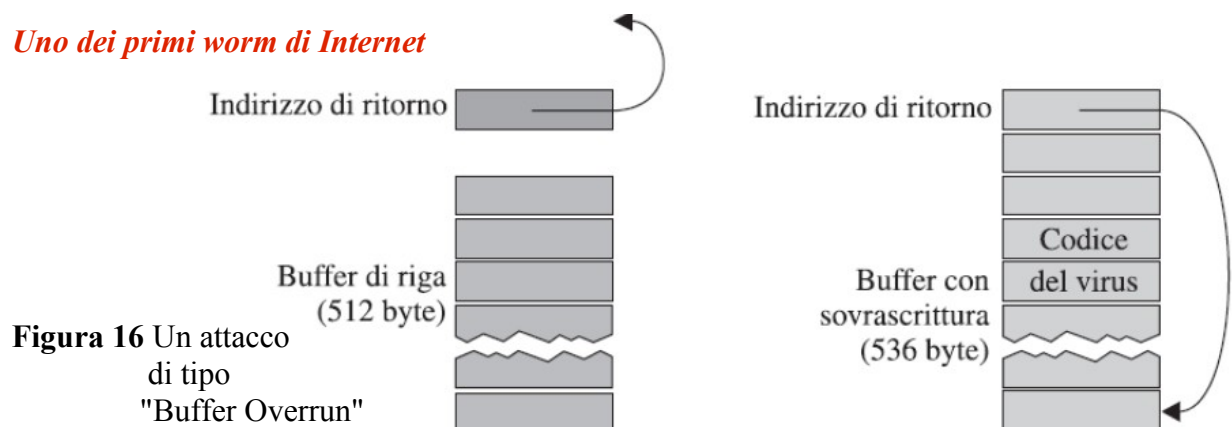


Figura 16 Un attacco di tipo "Buffer Overrun"



## Uso di metodi della classe Arrays

// riempimento anche parziale

```
import java.util.*;
class Array2 {
    public static void main(String [] args) {
        int v[] = new int[5]; // dichiarazione e creazione
        System.out.println("\nPrima dell'inizializzazione");
        for (int i= 0; i< v.length; i++)
            System.out.print(v[i]+" ");

        Arrays.fill(v,123); // anche Arrays.fill(v, 1,3,123) parziale riempimento

        System.out.println("\nDopo l'inizializzazione");
        for (int i= 0; i< v.length; i++)
            System.out.print(v[i]+" ");
    }
}
```

// ricerca binaria

```
import java.util.*;
public class Binary{

    public static void main(String [] args) {
        int v[] = {2,3,4,5,6}; // dichiarazione e inizializzazione
        System.out.println("\nPrima della ricerca");
        for (int i= 0; i< v.length; i++)
            System.out.print(v[i]+" ");

        int elemento = Arrays.binarySearch(v,3);

        System.out.println("\nIl numero 3 e' stato trovato nella posizione "+ (elemento+1) + "^");
    }
}
```

// ordinamento

```
import java.util.*;
public class Sort{
    public static void main(String [] args) {
        int v[] = {2,1,4,3,5}; // dichiarazione e inizializzazione
        System.out.println("\nPrima dell'ordinamento\n");
        for (int i= 0; i< v.length; i++)
            System.out.print(v[i]+" ");

        Arrays.sort(v);

        System.out.println("\n\nDopo l'ordinamento\n");
        for (int i= 0; i< v.length; i++)
            System.out.print(v[i]+" ");
        System.out.println("\n");
    }
}
```