Array

- Array: sequenza di valori dello stesso tipo
- Costruzione di un array:

```
new double[10]
```

- Memorizzazione in una variabile di tipo `double[]`

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

- Quando l'array viene creato, tutti i suoi elementi vengono inizializzati in base al tipo di array:
  - Numeri: `0`
  - Boolean: *`false`*
  - Riferimenti ad oggetti: *`null`*

Array

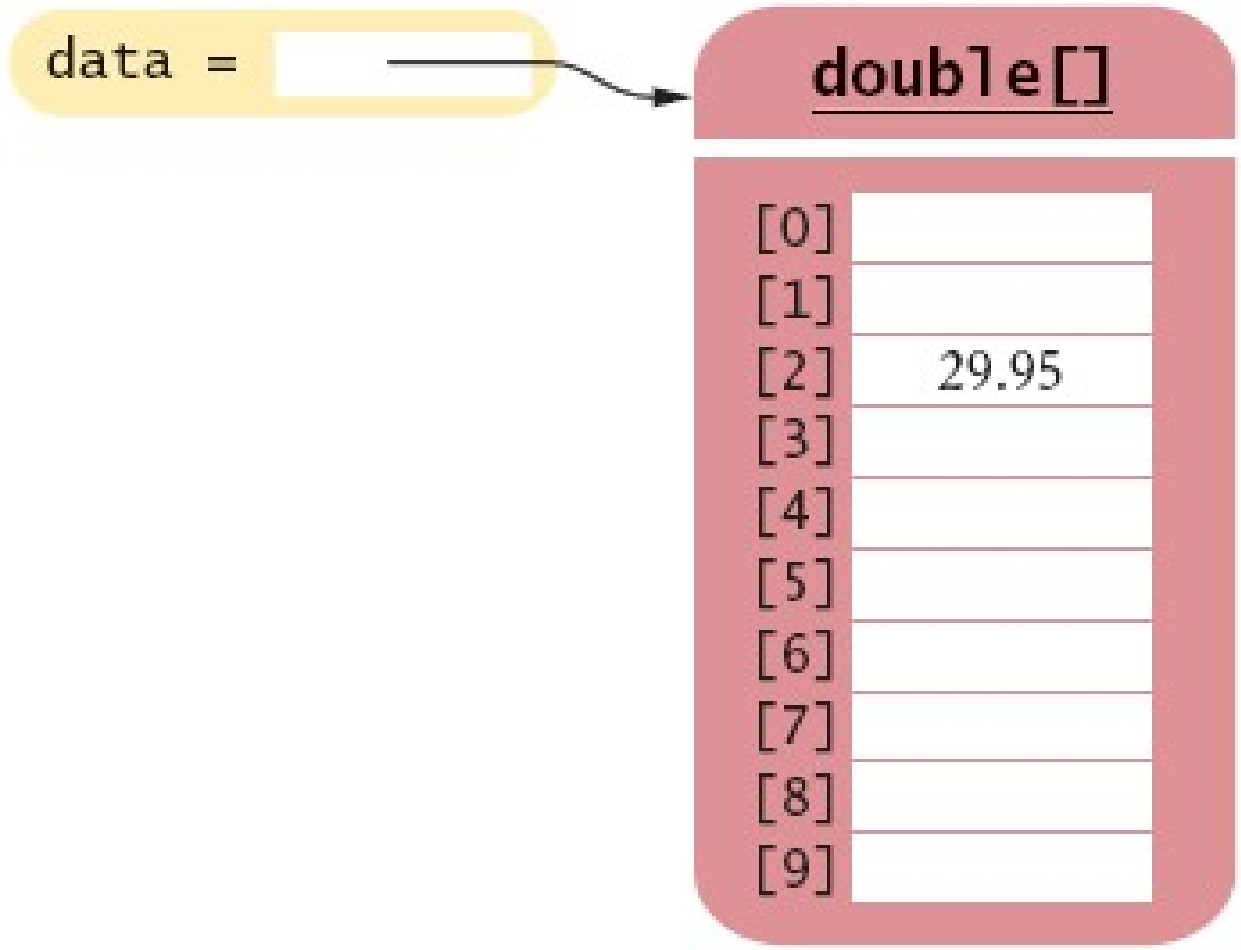


Figure 2 Storing a Value in an Array

Array

- Uso del valore memorizzato:

```
System.out.println("The value of this data item is "
      + data[4]);
```

- La lunghezza dell'array è contenuta in `data.length` (**non è un metodo, ma una costante pubblica**)
- Gli indici variano tra `0` e `length - 1`
- Accedere ad un elemento non esistente provoca un errore

```
double[] data = new double[10];
data[10] = 29.95; // ERROR
```

- Limitazione: gli array hanno lunghezza fissa

Sintassi 7.1 Costruzione di array

new *typeName*[*length*]

Esempio:

new double[10]

Scopo:

Costruire un array con un assegnato numero di elementi.

Sintassi 7.2 Accesso ad un elemento di un array

arrayReference[*index*]

Esempio:

data[2]

Scopo:

Accedere ad un elemento di un array.

Array List (liste sequenziali)

- La classe `ArrayList` consente di gestire una raccolta di oggetti
- Può crescere e calare di dimensione secondo necessità
- La classe `ArrayList` fornisce metodi per molte operazioni comuni, come l'inserimento e la rimozione di elementi
- La classe `ArrayList` è una classe generica: `ArrayList<T>` raccoglie oggetti di tipo `T`:

```
ArrayList<BankAccount> accounts = new
        ArrayList<BankAccount>();
accounts.add(new BankAccount(1001));
accounts.add(new BankAccount(1015));
accounts.add(new BankAccount(1022));
```

- Il metodo `size` restituisce il numero di elementi

Recuperare un elemento di un array list

- Si impiega il metodo `get`
- L'indice inizia da 0
- ```
  BankAccount anAccount = accounts.get(2); // gets the
  third element of the array list
  ```
- Si ha un errore se l'indice è al di fuori del range corretto
- Un tipico errore:

```
int i = accounts.size();
anAccount = accounts.get(i); // Error
//legal index values are 0. . .i-1
```
```

Aggiungere elementi

- Il metodo `set` sovrascrive un valore già esistente

```
BankAccount anAccount = new BankAccount(1729);
accounts.set(2, anAccount);
```

- Il metodo `add` aggiunge un nuovo elemento al vettore prima della posizione indicata come parametro

```
accounts.add(i, a)
```

(l'elemento `a` finisce cioè nella posizione `i` e l'elemento precedentemente in posizione `i` e quelli successivi avanzano di una posizione)

- Se nel metodo `add` non si indica la posizione, l'elemento viene aggiunto in coda

Continua

Aggiungere elementi (continua)

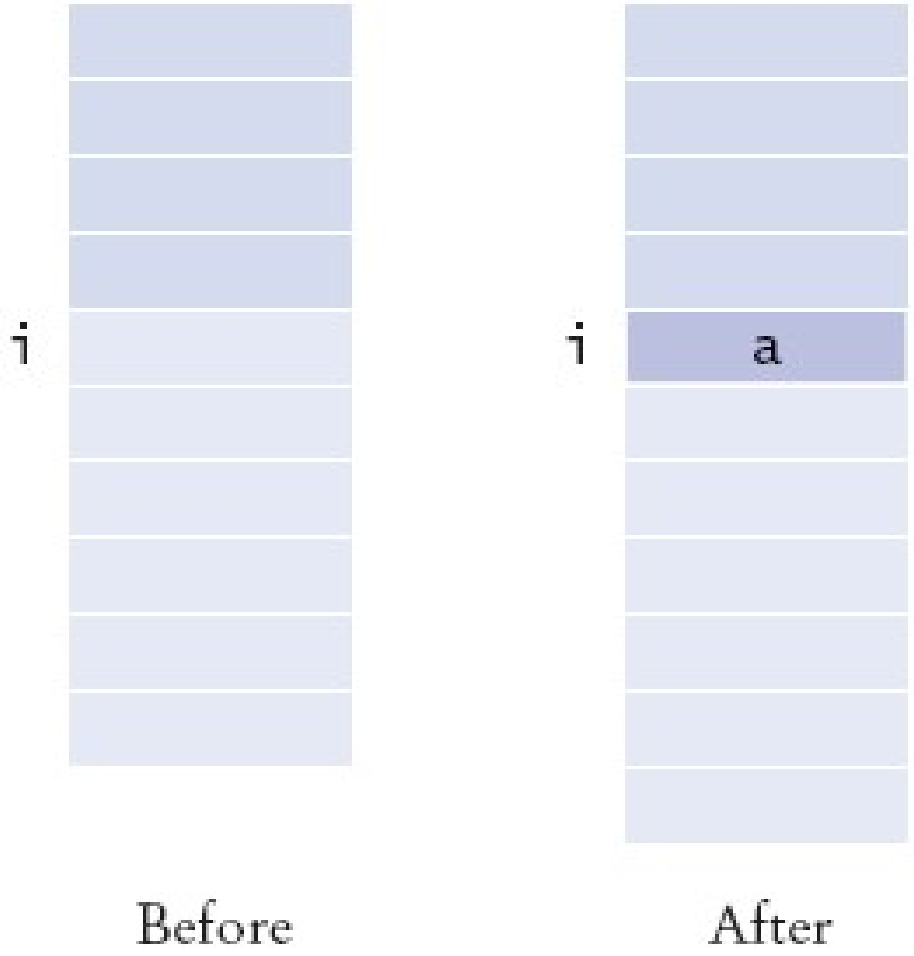


Figure 3 Adding an Element in the Middle of an Array List

Rimuovere elementi

Il metodo `remove` rimuove un elemento dalla posizione indicata

`accounts.remove(i)`

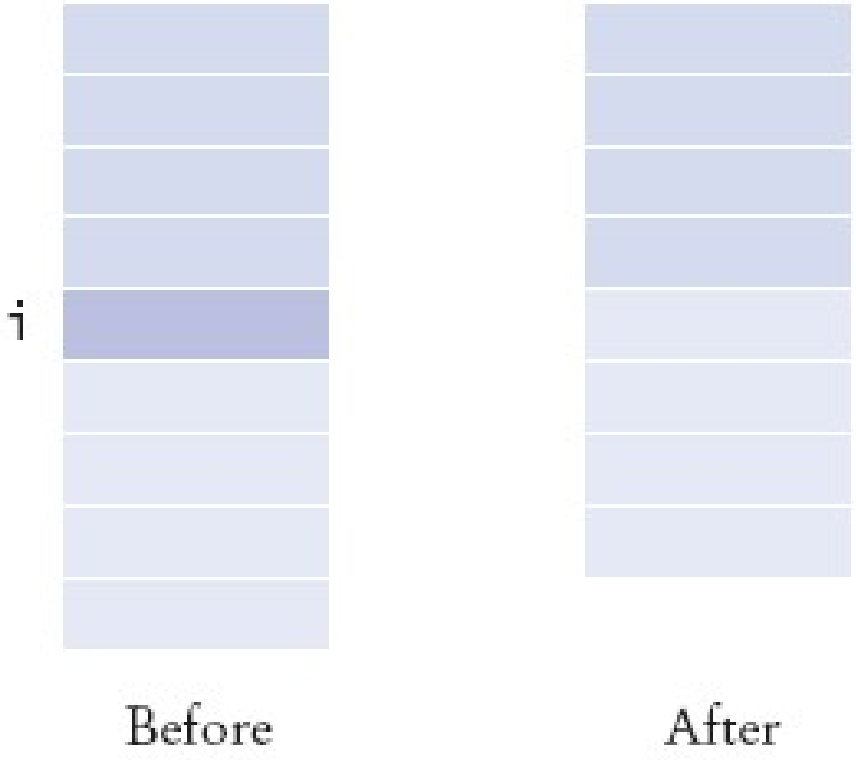


Figure 4 Removing an Element from the Middle of an Array List

Involucri

- Non si possono inserire valori di tipo primitivo negli array list
- Per trattare valori di tipo primitivo come oggetti si usano le classi involucro:

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);
double x = data.get(0);
```

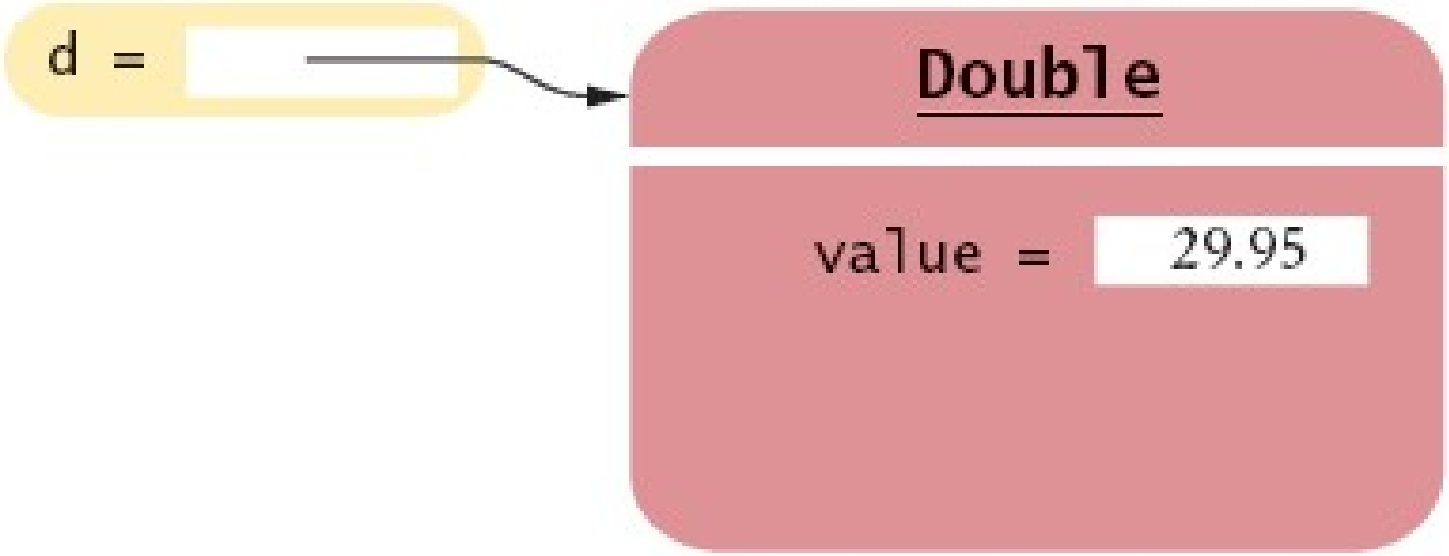


Figure 5 An Object of a Wrapper Class

Involucri

Ci sono classi involucro per tutti gli 8 tipi primitivi:

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Auto-impacchettamento

- Auto-impacchettamento: da Java 5.0, la conversione tra tipi primitivi e le corrispondenti classi involucro è automatica.

```
Double d = 29.95; // auto-boxing; same as Double d =
           new Double(29.95);
double x = d; // auto-unboxing; same as double x =
           d.doubleValue();
```

- L'auto-impacchettamento funziona anche nelle espressioni aritmetiche

```
Double e = d + 1;
```

- Significato:
  - *auto-spacchetta d in un* `double`
  - *aggiungi 1*
  - *auto-impacchetta il risultato in un nuovo oggetto* `Double`
  - *memorizza un riferimento al nuovo oggetto* `Double` *in e*

Il ciclo `for` generalizzato

- Attraversa tutti gli elementi di una collezione:

```
double[] data = . . .;
double sum = 0;
for (double e : data) // You should read this loop as
          "for each e in data"
{
   sum = sum + e;
}
```

- Alternativa tradizionale:

```
double[] data = . . .;
double sum = 0;
for (int i = 0; i < data.length; i++)
{
   double e = data[i];
   sum = sum + e;
}
```

Il ciclo `for` generalizzato

- Funziona pure per oggetti `ArrayList`:

```
ArrayList<BankAccount> accounts = . . . ;
double sum = 0;
for (BankAccount a : accounts)
{
   sum = sum + a.getBalance();
}
```

- Equivalente al seguente ciclo `for` tradizionale:

```
double sum = 0;
for (int i = 0; i < accounts.size(); i++)
{
   BankAccount a = accounts.get(i);
   sum = sum + a.getBalance();
}
```

Sintassi 7.3 Il ciclo "for each"

```
for (Type variable : collection)
   statement
```

Esempio:

```
for (double e : data)
   sum = sum + e;
```

Scopo:

Eseguire un ciclo per ogni elemento nella raccolta. In ogni iterazione la variabile riceve in assegnazione l'elemento successivo della raccolta. Successivamente viene eseguita l'istruzione.

Si scriva un ciclo "for each" che visualizza tutti gli elementi nell'array `data`.

Risposta:

```
for (double x : data) System.out.println(x);
```

Perché il ciclo "for each" non è un'alternativa ragionevole per il seguente ciclo `for` tradizionale?

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

Risposta: Il ciclo scrive un valore in `data[i]` dipendente dall'indice i. Il ciclo "for each" non possiede la variabile indice `i`.

Contare valori di un array con determinate caratteristiche

Controllare tutti gli elementi e contare quanti soddisfano a particolari caratteristiche fino a raggiungere la fine dell'array list.

```
public class Bank
{
   public int count(double atLeast)
   {
      int matches = 0;
      for (BankAccount a : accounts)
      {
         if (a.getBalance() >= atLeast)
            matches++;
           // Found a match
      }
      return matches;
   }
   . . .
   private ArrayList<BankAccount> accounts;
}
```

Trovare un valore

Controllare tutti gli elementi fino a trovare quello cercato.

```
public class Bank
{
   public BankAccount find(int accountNumber)
   {
      for (BankAccount a : accounts)
      {
         if (a.getAccountNumber() == accountNumber)
            // Found a match
            return a;
      }
      return null; // No match in the entire array list
   }
   . . .
}
```

Trovare il massimo o il minimo

- Inizializzare una variabile con l'elemento iniziale
- Confrontare questa variabile con gli elementi rimanenti
- Aggiornarla se si trova un valore maggiore o rispettivamente minore
- Esempio:

```
BankAccount largestYet = accounts.get(0);
for (int i = 1; i < accounts.size(); i++)
{
   BankAccount a = accounts.get(i);
   if (a.getBalance() > largestYet.getBalance())
      largestYet = a;
}
return largestYet;
```

Continua

Trovare il massimo o il minimo (continua)

- Se l'array è vuoto, restituire `null`:

```
if (accounts.size() == 0) return null;
BankAccount largestYet = accounts.get(0);
...
```

ch07/bank/Bank.java

```
01: import java.util.ArrayList;
02:
03: /**
04:    This bank contains a collection of bank accounts.
05: */
06: public class Bank
07: {
08:    /**
09:       Constructs a bank with no bank accounts.
10:    */
11:    public Bank()
12:    {
13:       accounts = new ArrayList<BankAccount>();
14:    }
15:
16:    /**
17:       Adds an account to this bank.
18:       @param a the account to add
19:    */
20:    public void addAccount(BankAccount a)
21:    {
22:       accounts.add(a);
23:    }
```

Continua

ch07/bank/Bank.java (cont.)

```
24:
25:    /**
26:       Gets the sum of the balances of all accounts in this bank.
27:       @return the sum of the balances
28:    */
29:    public double getTotalBalance()
30:    {
31:       double total = 0;
32:       for (BankAccount a : accounts)
33:       {
34:          total = total + a.getBalance();
35:       }
36:       return total;
37:    }
38:
39:    /**
40:       Counts the number of bank accounts whose balance is at
41:       least a given value.
42:       @param atLeast the balance required to count an account
43:       @return the number of accounts having least the given balance
44:    */
45:    public int count(double atLeast)
46:    {
```

Continua

ch07/bank/Bank.java (cont.)

```
47:          int matches = 0;
48:          for (BankAccount a : accounts)
49:          {
50:             if (a.getBalance() >= atLeast) matches++; // Found a match
51:          }
52:          return matches;
53:       }
54:
55:       /**
56:          Finds a bank account with a given number.
57:          @param accountNumber the number to find
58:          @return the account with the given number, or null if there
59:          is no such account
60:       */
61:       public BankAccount find(int accountNumber)
62:       {
63:          for (BankAccount a : accounts)
64:          {
65:             if (a.getAccountNumber() == accountNumber) // Found a match
66:                return a;
67:          }
68:          return null; // No match in the entire array list Continua
69:       }
70:
```

ch07/bank/Bank.java (cont.)

```
71:    /**
72:       Gets the bank account with the largest balance.
73:       @return the account with the largest balance, or null if the
74:       bank has no accounts
75:    */
76:    public BankAccount getMaximum()
77:    {
78:       if (accounts.size() == 0) return null;
79:       BankAccount largestYet = accounts.get(0);
80:       for (int i = 1; i < accounts.size(); i++)
81:       {
82:          BankAccount a = accounts.get(i);
83:          if (a.getBalance() > largestYet.getBalance())
84:             largestYet = a;
85:       }
86:       return largestYet;
87:    }
88:
89:    private ArrayList<BankAccount> accounts;
90: }
```

ch07/bankBankTester.java

```
01: /**
02:    This program tests the Bank class.
03: */
04: public class BankTester
05: {
06:    public static void main(String[] args)
07:    {
08:       Bank firstBankOfJava = new Bank();
09:       firstBankOfJava.addAccount(new BankAccount(1001, 20000));
10:       firstBankOfJava.addAccount(new BankAccount(1015, 10000));
11:       firstBankOfJava.addAccount(new BankAccount(1729, 15000));
12:
13:       double threshold = 15000;
14:       int c = firstBankOfJava.count(threshold);
15:       System.out.println("Count: " + c);
16:       System.out.println("Expected: 2");
17:
18:       int accountNumber = 1015;
19:       BankAccount a = firstBankOfJava.find(accountNumber);
20:       if (a == null)
```

Continua

ch07/bankBankTester.java (cont.)

```
21:             System.out.println("No matching account");
22:          else
23:             System.out.println("Balance of matching account: " +
                        a.getBalance());
24:          System.out.println("Expected: 10000");
25:
26:          BankAccount max = firstBankOfJava.getMaximum();
27:          System.out.println("Account with largest balance: "
28:                + max.getAccountNumber());
29:          System.out.println("Expected: 1001");
30:       }
31: }
```

Output:

```
Count: 2
Expected: 2
Balance of matching account: 10000.0
Expected: 10000
Account with largest balance: 1001
Expected: 1001
```

Array 2-dimensionali

- Nel costruire un array 2-dimensionale bisogna specificare il numero di righe e di colonne

```
final int ROWS = 3;
final int COLUMNS = 3;
String[][] board = new String[ROWS][COLUMNS];
```

- Si accede agli elementi attraverso due indici `a[i][j]`

```
board[i][j] = "x";
```

Copiare gli array: copiare il riferimento all'array

Copiare una variabile array produce un secondo riferimento allo stesso array

```
Double[ ] data = new double[10];
// fill array . . .
Double[ ] prices = data;
```

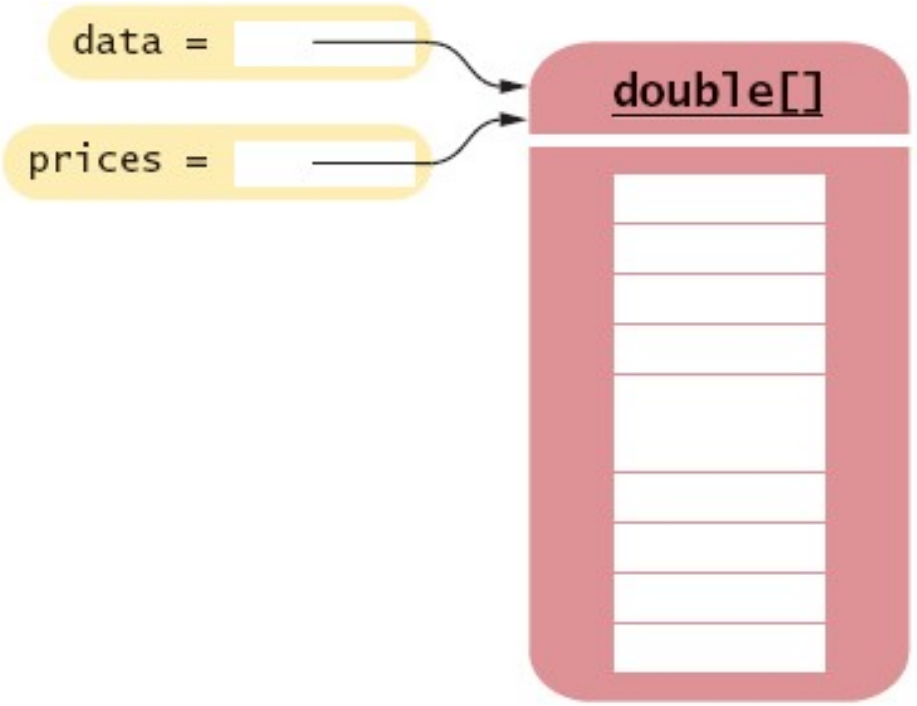


Figure 7 Two References to the Same Array

Copiare gli array: clonare un array

Si usi il metodo `clone` per realizzare una vera copia

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

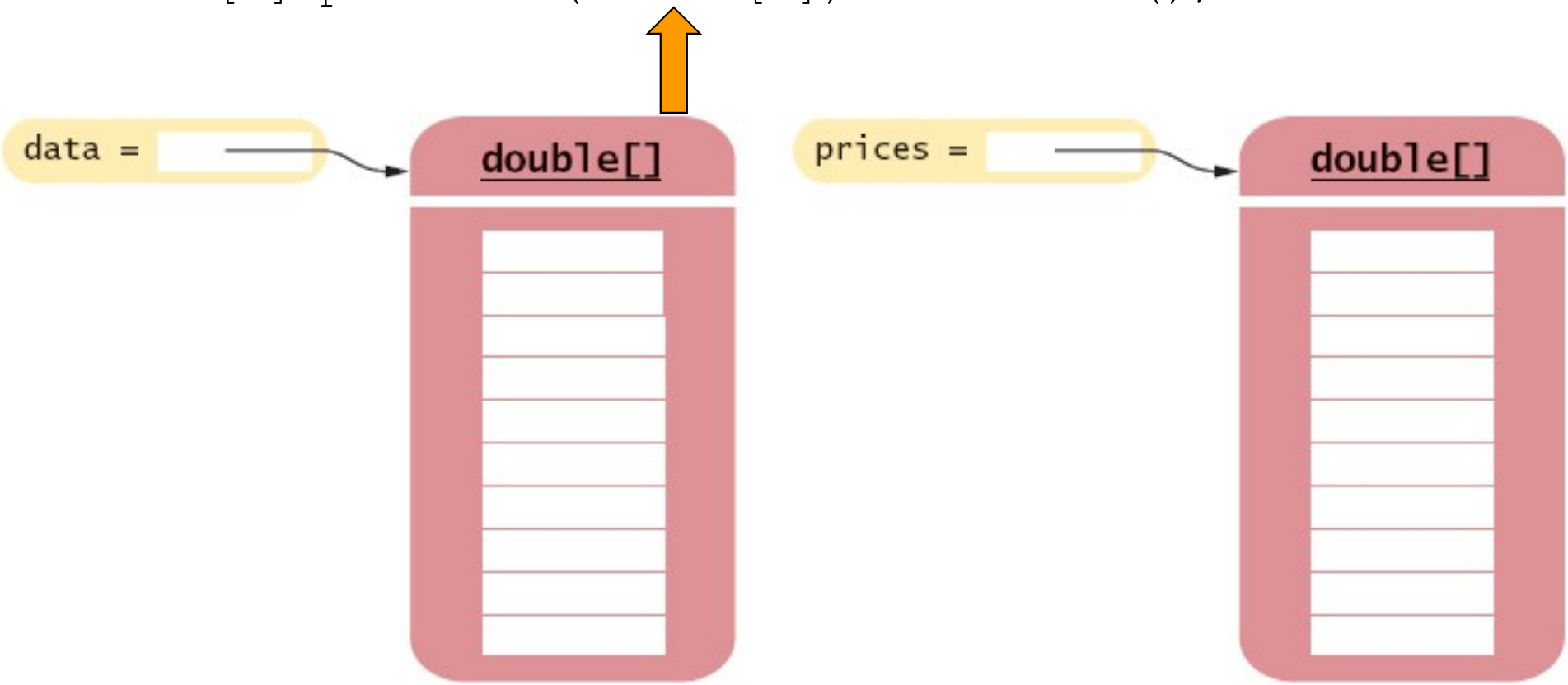


Figure 8 Cloning an Array

Copiare array: copiare gli elementi dell'array

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

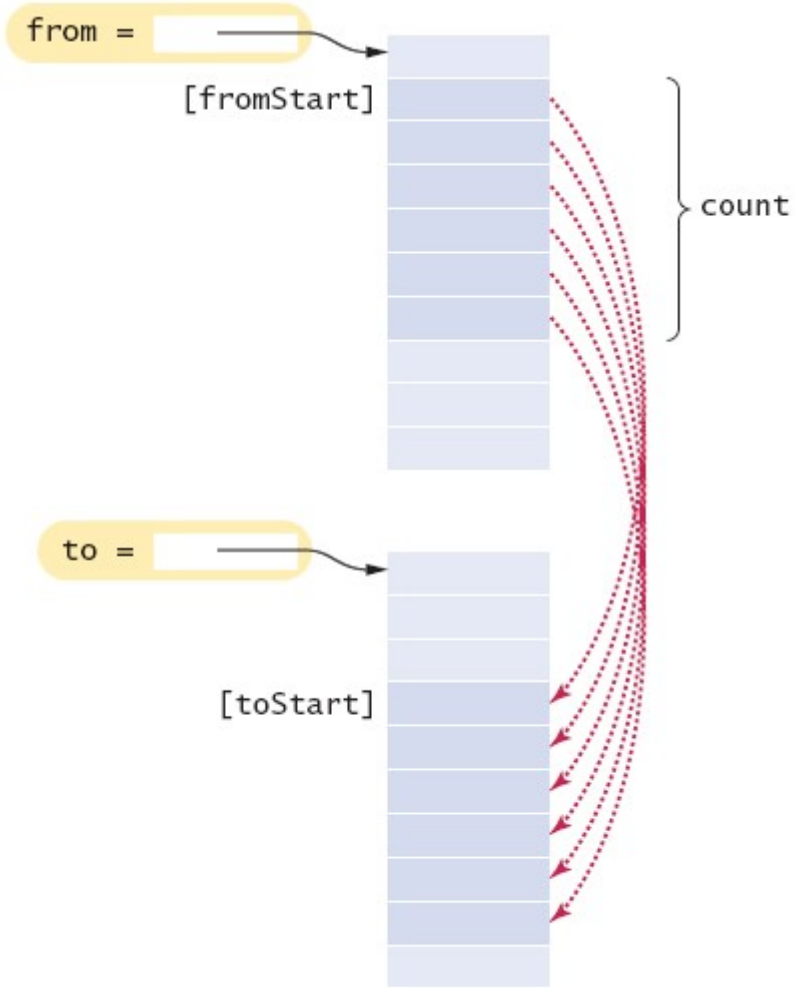


Figure 9 The System.arraycopy Method

Aggiungere un elemento ad un array

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

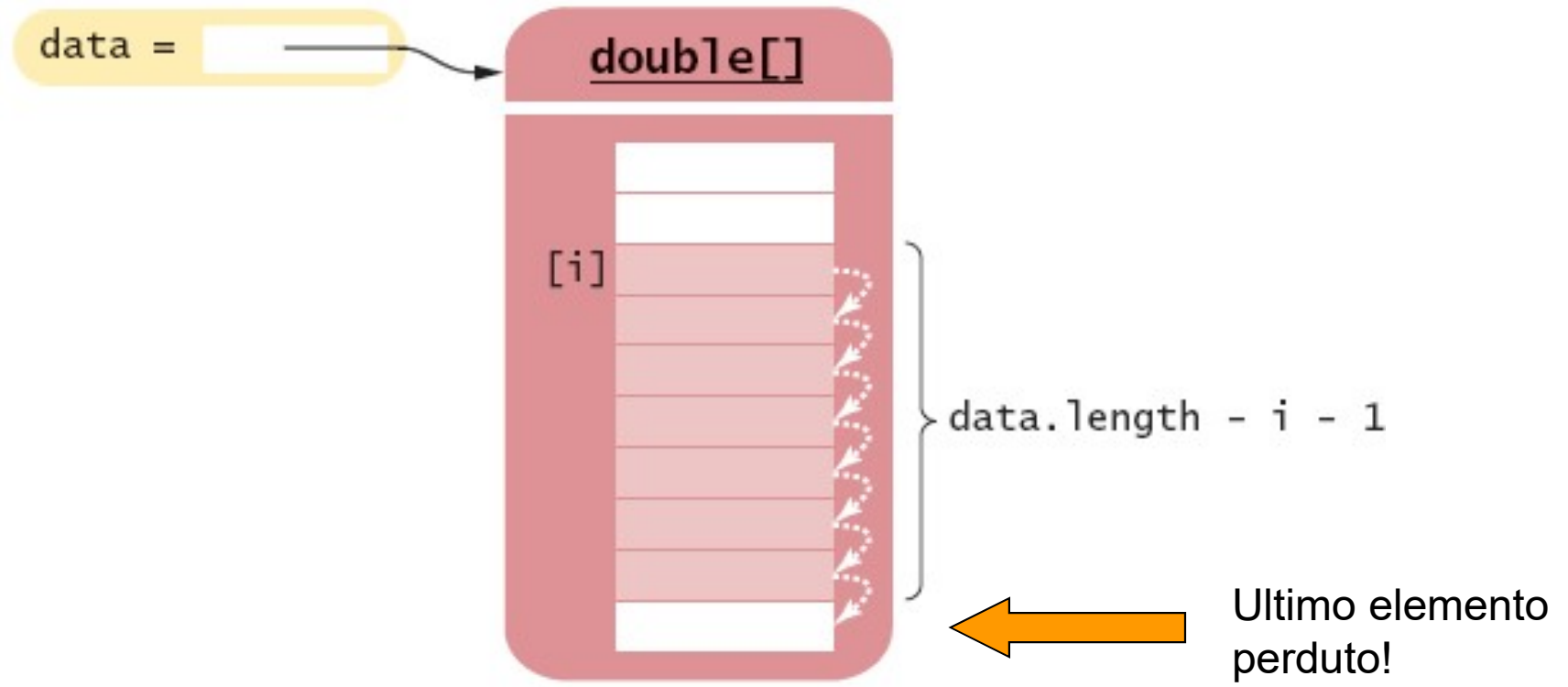


Figure 10 Inserting a New Element into an Array

Rimuovere un elemento da un array

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

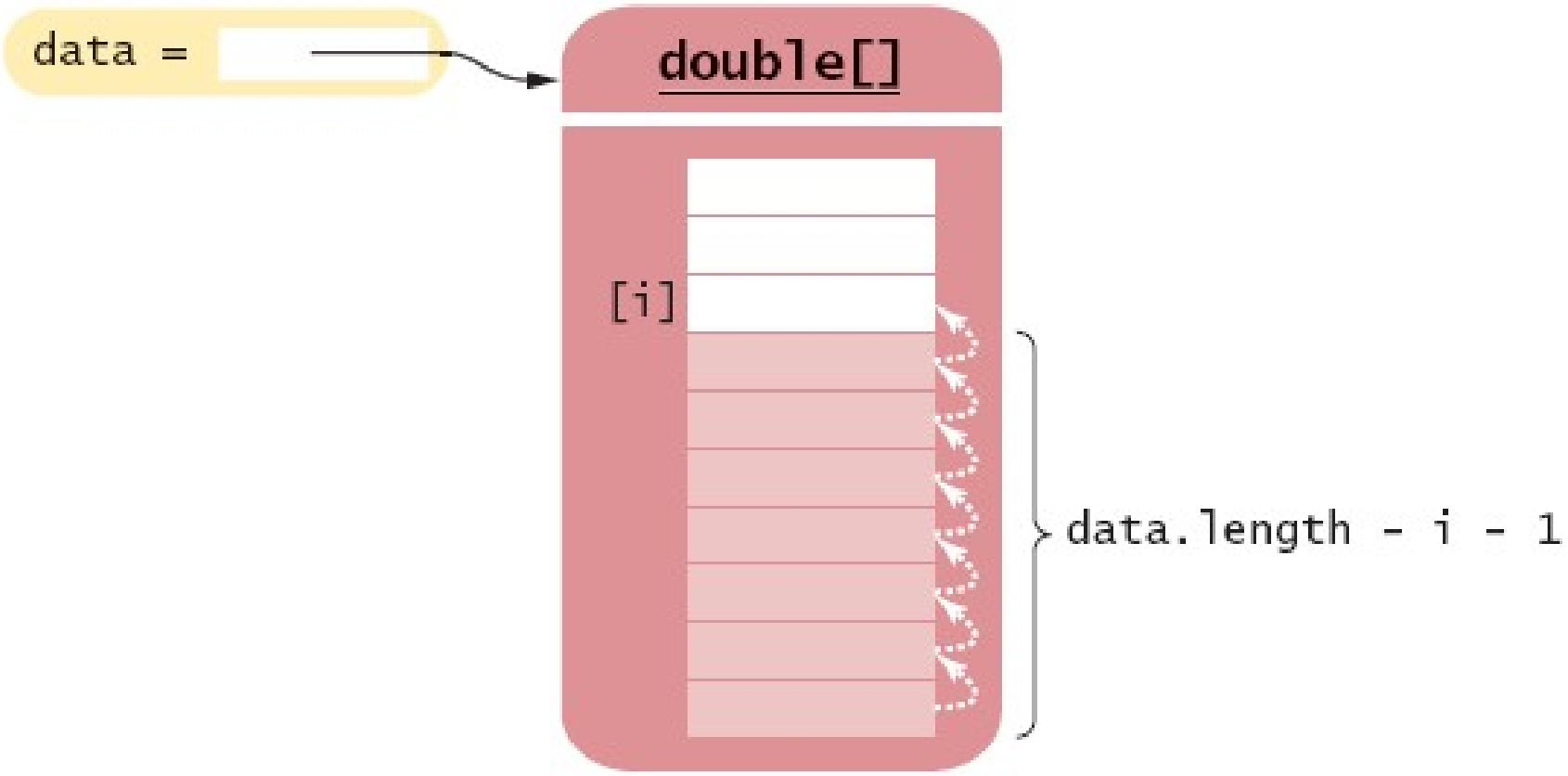


Figure 11 Removing an Element from an Array

Incrementare la dimensione dell'array

- Se l'array è pieno e si necessita di ulteriore spazio, si può farlo crescere:
- Si crea un nuovo array più grande:

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

- Si copiano tutti gli elementi nel nuovo array:

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

- Si memorizza il riferimento al nuovo array nella variabile array:

```
data = newData;
```

Incrementare la dimensione dell'array (continua)

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

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

data = newData;
```

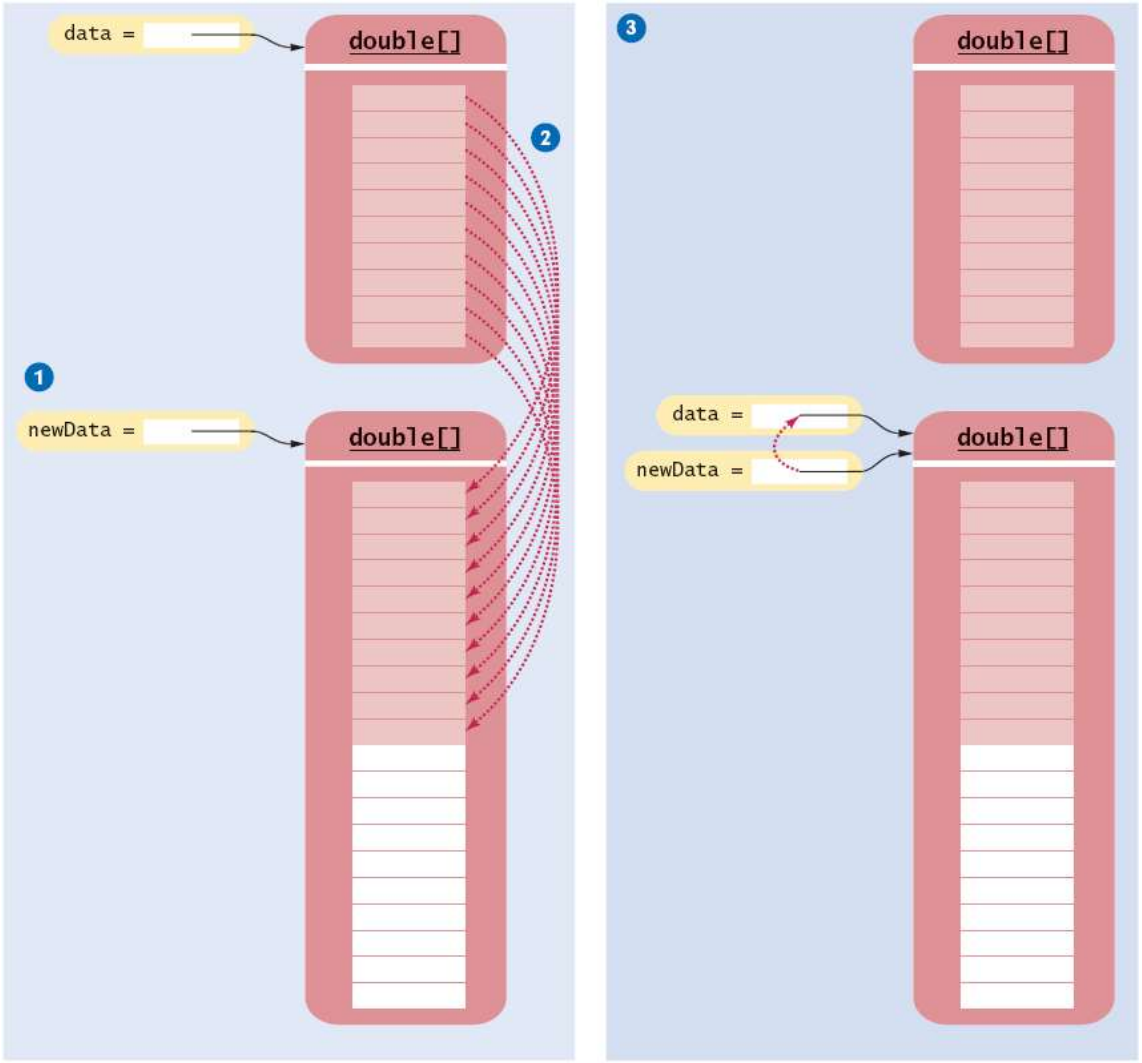
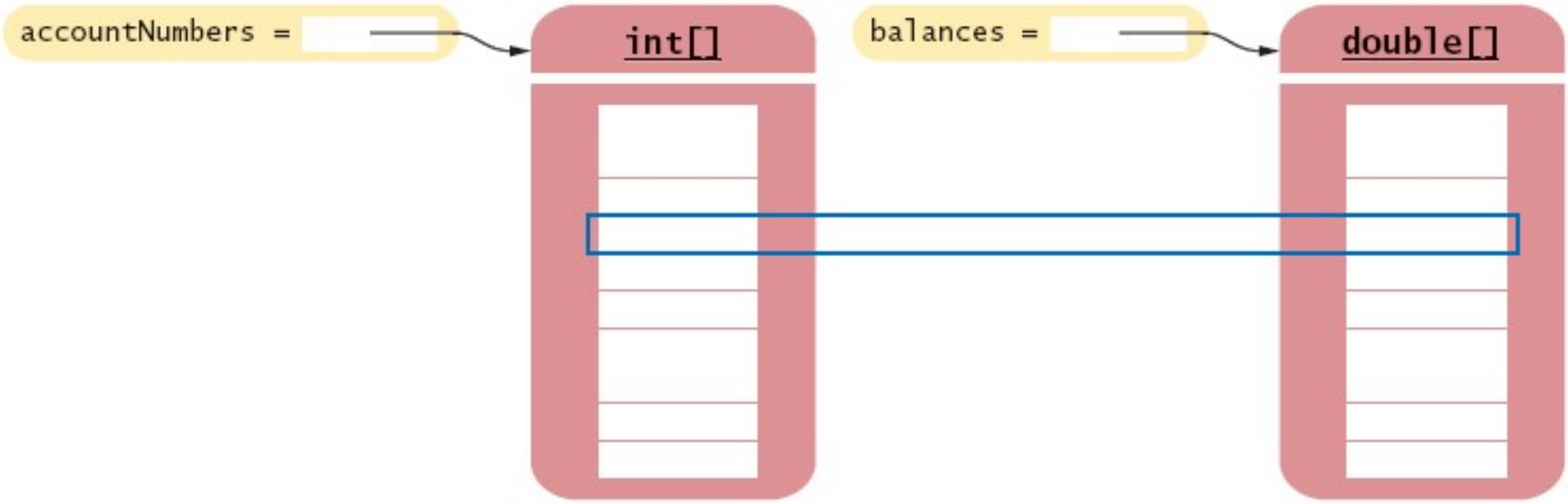


Figure 12 Growing an Array

Trasformare array paralleli in array di oggetti

```
// Don't do this
int[] accountNumbers;
double[] balances;
```

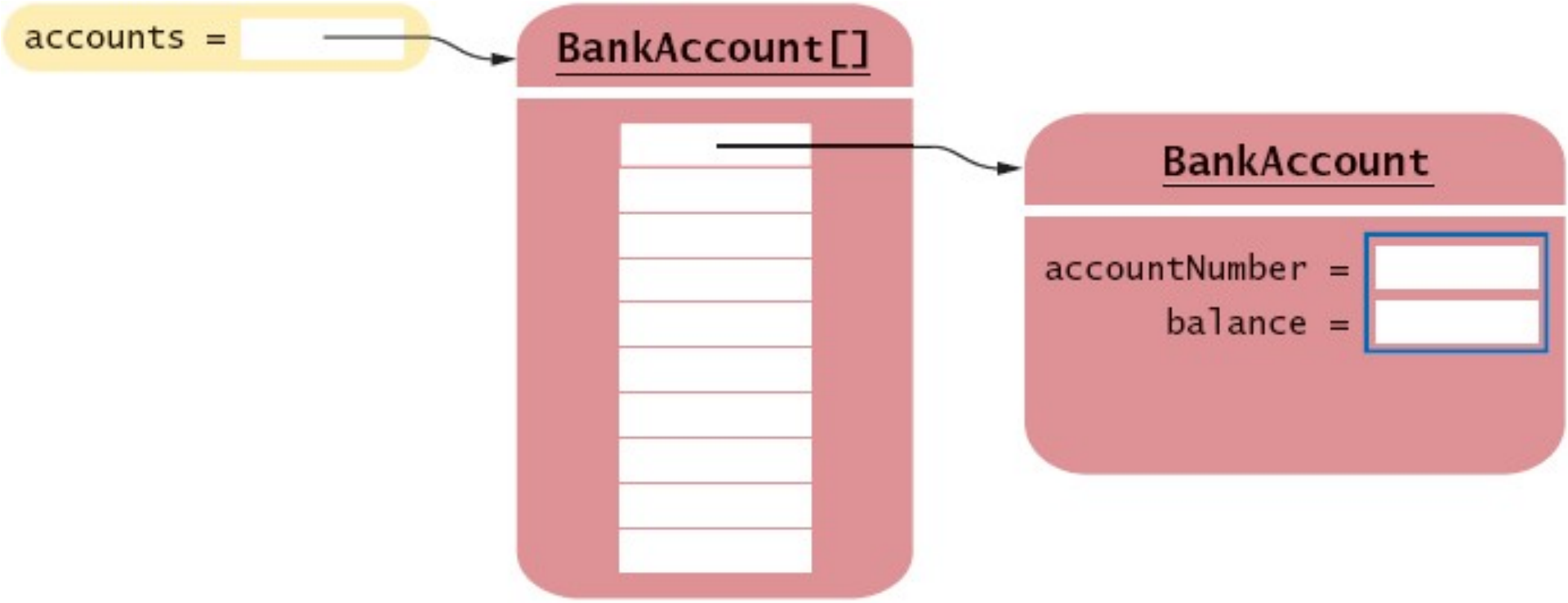


Avoid Parallel Arrays

Trasformare array paralleli in array di oggetti

Si evitino array paralleli cambiandoli in array di oggetti:

`BankAccount[] = accounts`



Reorganizing Parallel Arrays into an Array of Objects

Array parzialmente riempiti

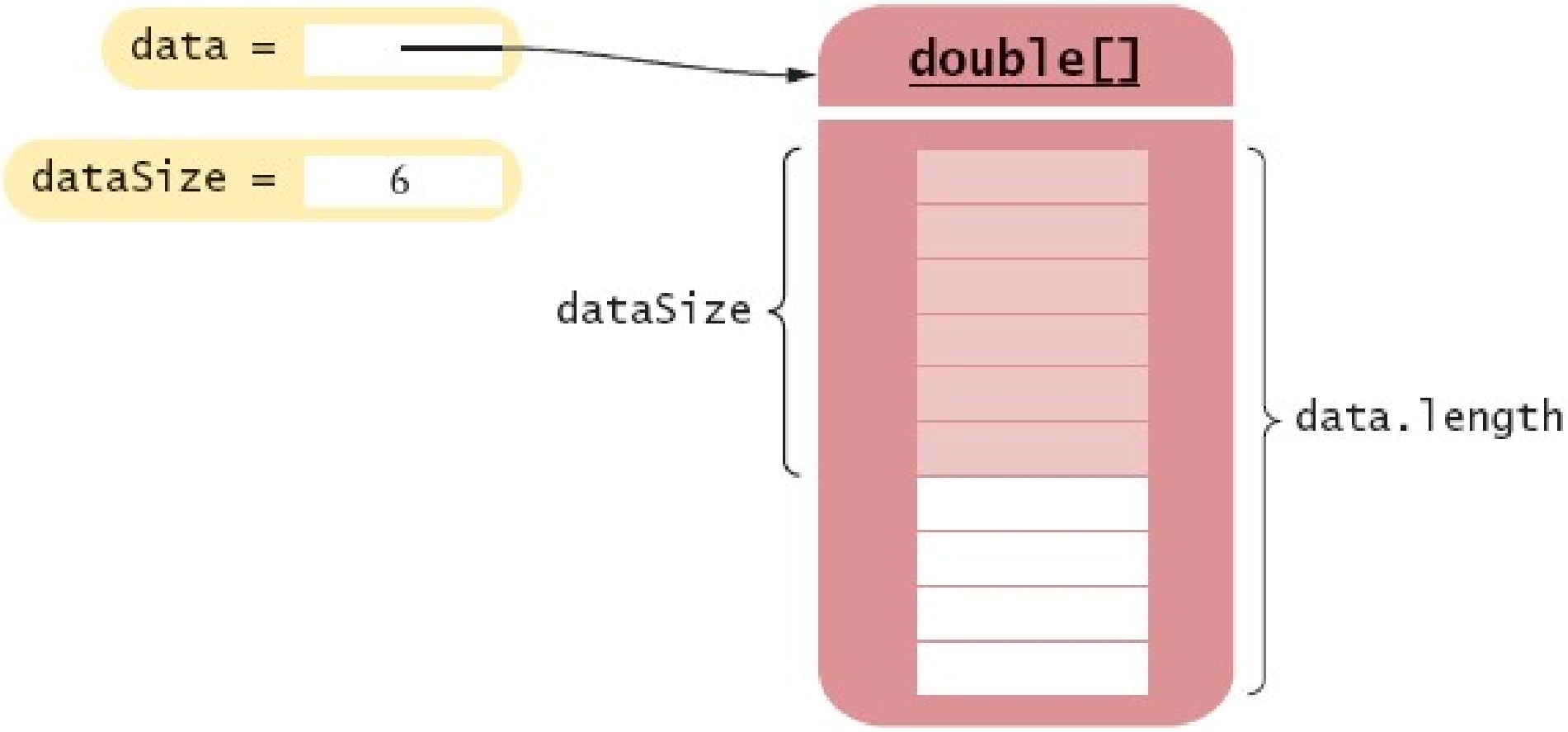
- Lunghezza dell'array = massimo numero di elementi nell'array
- Di norma un array è riempito soltanto parzialmente
- Serve dunque una variabile ausiliaria per tenere traccia della dimensione corrente
- Convenzione di denominazione: *nome array* + *Size*

```
final int DATA_LENGTH = 100;
double[] data = new double[DATA_LENGTH];
int dataSize = 0;
```

- Aggiornamento di dataSize quando viene aggiunto un elemento:

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

Array parzialmente riempiti (continua)



A Partially Filled Array