

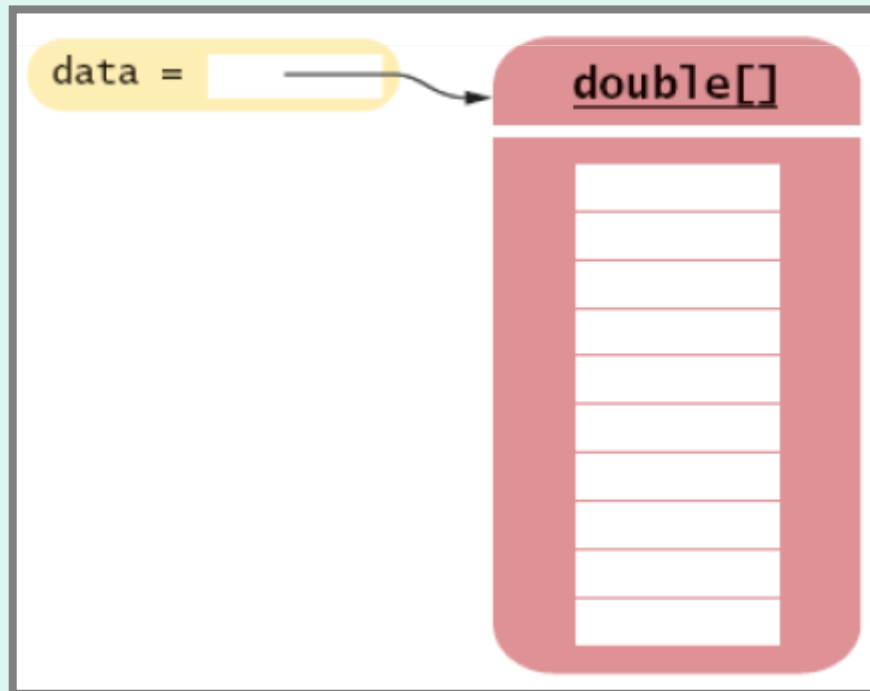
Arrays

Array Lists

Arrays

- **Array: una sequenza di valori dello stesso tipo**

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



Continua...

Arrays

- Al momento della allocazione, tutti i valori sono inizializzati ai valori di default del tipo base
 - numeri: **0**
 - booleani: **false**
 - riferimenti: **null**

Arrays

- **Accesso agli elementi mediante l'operatore []**

```
data[2] = 29.95;
```

- **Lunghezza dell'array: `data.length` (N.B. non è un metodo)**
- **Indici compresi tra 0 e `data.length - 1`**

```
data[10] = 29.95;// ERRORE: ArrayIndexOutOfBoundsException
```

Domanda

- Quali elementi contiene l'array al termine della seguente inizializzazione?

```
double[] data = new double[10];  
for (int i = 0; i < data.length; i++) data[i] = i * i;
```

Risposta

- 0 1 4 9 16 25 36 49 64 81 **ma non 100**

Domanda

- Cosa stampano i seguenti comandi. Ovvero, se causano un errore, quale errore causano? Specificate se si tratta di un errore di compilazione o di un errore run-time.

```
1.  double[] a = new double[10];  
    System.out.println(a[0]);  
2.  double[] b = new double[10];  
    System.out.println(b[10]);  
3.  double[] c;  
    System.out.println(c[0]);
```

Risposta

- 2. 0
 3. errore run-time error: indice fuori range
 4. Errore compile-time: c non è inizializzato

Array bidimensionali

- All'atto della costruzione specifichiamo il numero di righe e di colonne:

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

- Per accedere un elemento, utilizziamo i due indici: **a[i][j]**

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

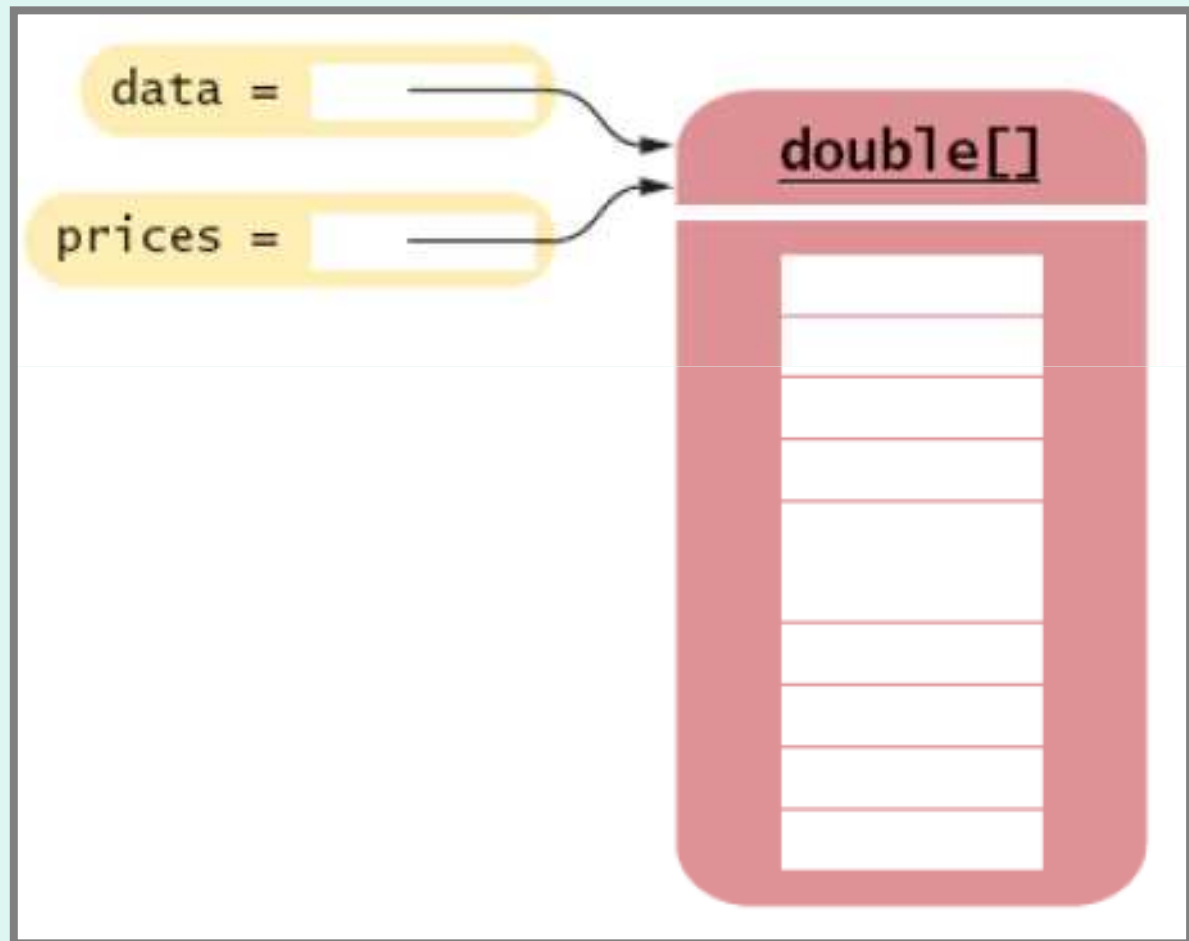
Copia di array

- Gli array sono oggetti, quindi l'assegnamento tra array è un assegnamento di riferimenti

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

Continua...

Copia di array



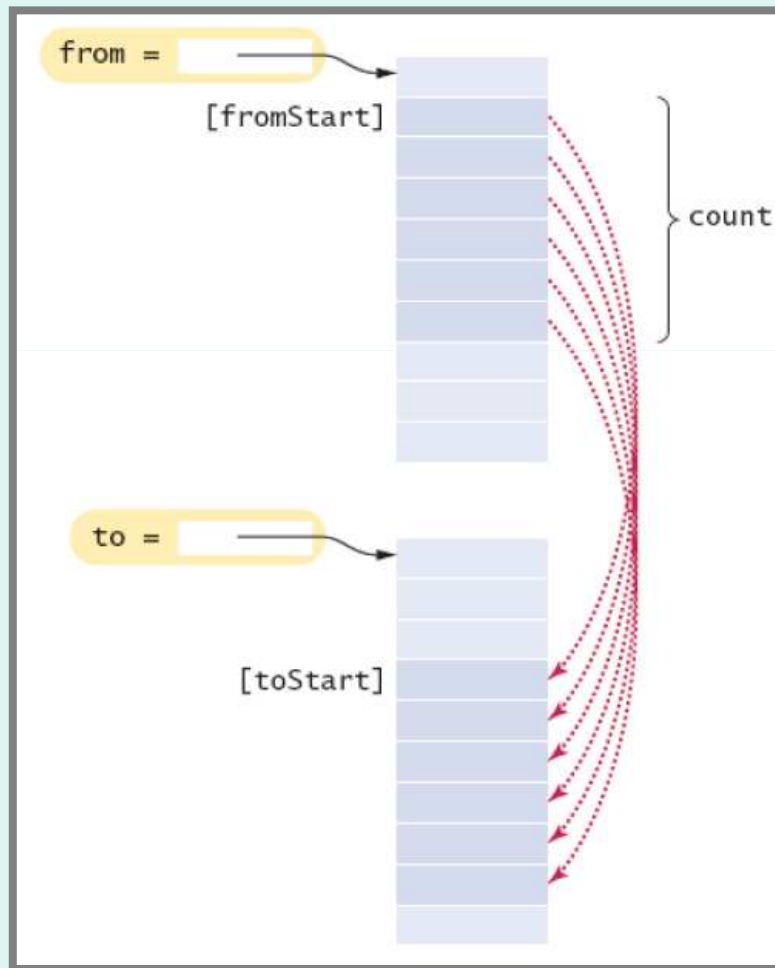
Copia di array – copia di elementi

- Volendo duplicare gli elementi, il linguaggio fornisce metodi efficienti:

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

Continua...

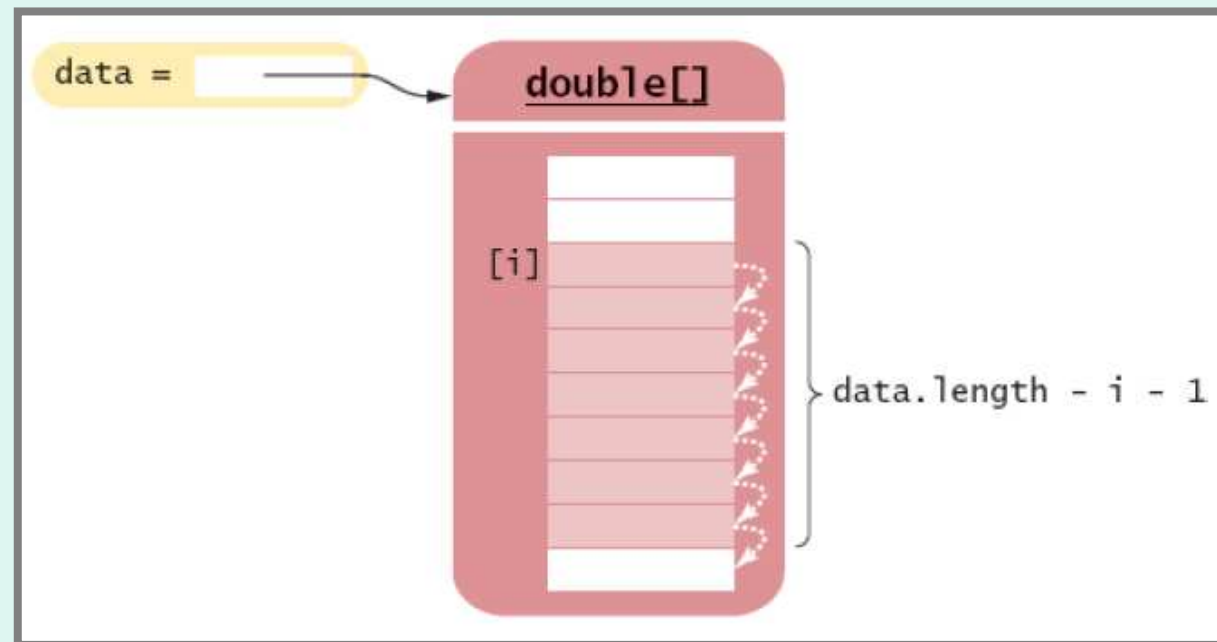
Copia di array – copia di elementi



Copia di elementi

- Array sorgente e destinazione possono coincidere

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



ArrayLists

- La classe **ArrayList** gestisce una sequenza di oggetti
- A differenza degli array può variare in dimensione
- Fornisce metodi corrispondenti a molte operazioni comuni
 - inserzione, accesso e rimozione di elementi, ...

Continua...

Array Lists

- La classe **ArrayList** è una classe parametrica (generica)
- **ArrayList<T>** contiene 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));
```

- **size()**: il metodo che calcola il numero di elementi nella struttura

Accesso agli elementi

- `get()`
- **Come per gli array**
 - gli indici iniziano da 0
 - errore se l'indice è fuori range
 - posizioni accessibili: $0 \dots \text{size}() - 1$.

```
BankAccount anAccount = accounts.get(2);  
    // il terzo elemento dalla arraylist
```

Continua...

Nuovi elementi

- **set()** sovrascrive un valore esistente

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

- **add()** aggiunge un nuovo valore, alla posizione i

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

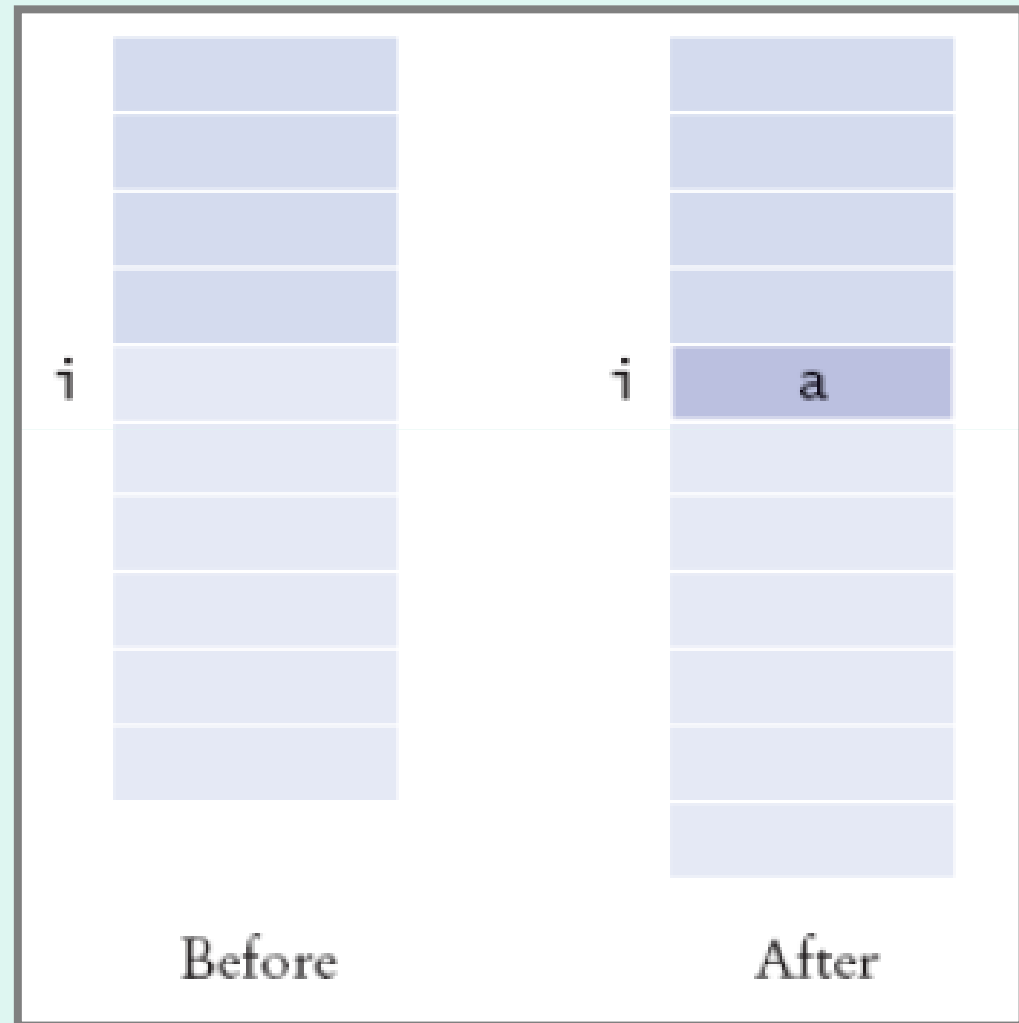
- **Oppure all'ultima posizione**

```
accounts.add(a)
```

Continua...

Nuovi elementi – add

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

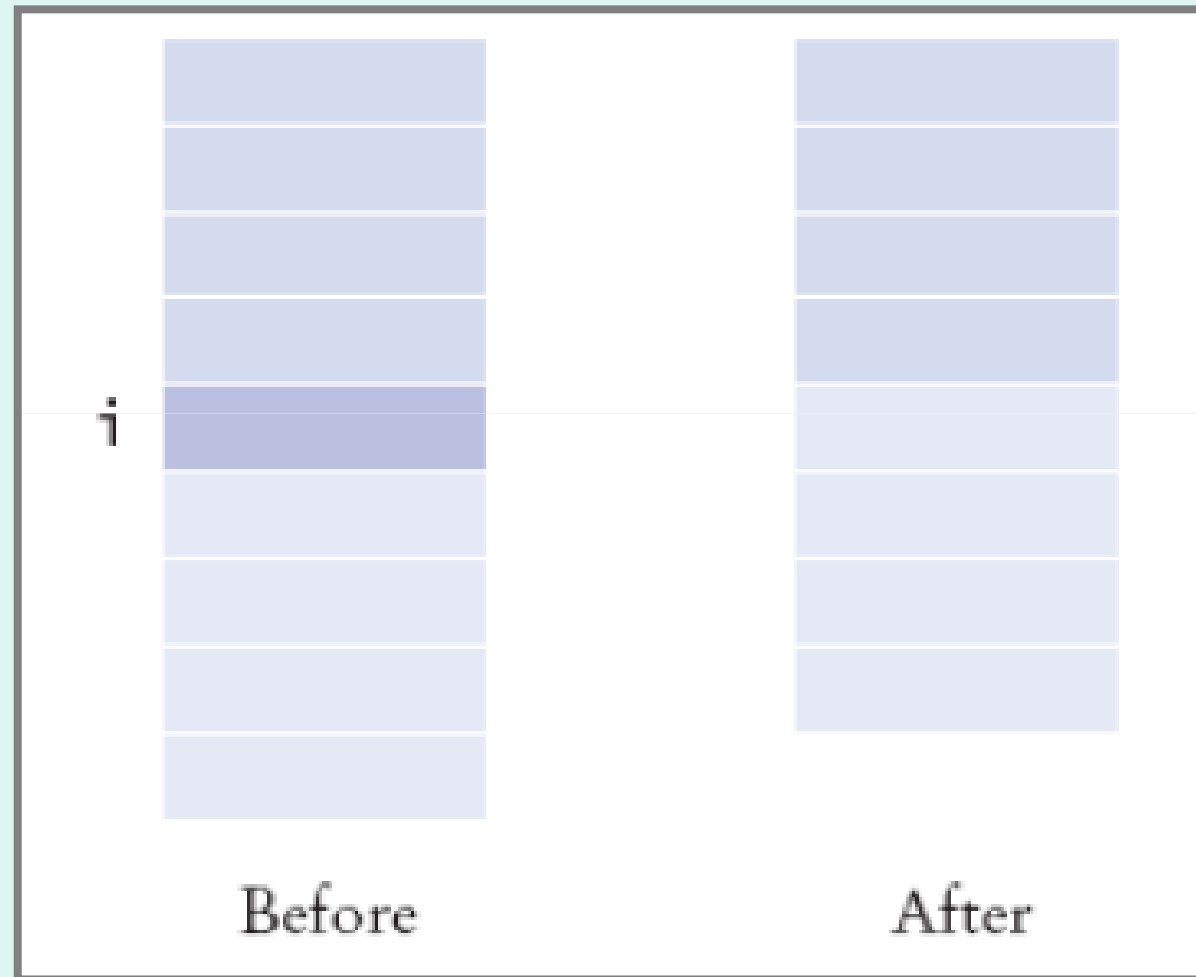


Rimozione di elementi

- **remove()** rimuove l'elemento all'indice i

```
accounts.remove(i)
```

Rimozione di elementi



Errori tipici

- **Accesso fuori dai bounds**

- indici legali 0..size()-1

```
int i = accounts.size();  
accounts.get(i);           // errore!  
accounts.set(i, anAccount); // errore!  
accounts.add(i+1, anAccount); // errore!
```

Domanda

3. Quale è il contenuto della struttura `names` dopo le seguenti istruzioni?

```
ArrayList<String> names = new ArrayList<String>();  
names.add("A");  
names.add(0, "B");  
names.add("C");  
names.remove(1);
```

Risposta

3. contiene le stringhe **"B"** e **"C"** alle posizioni 0 e 1

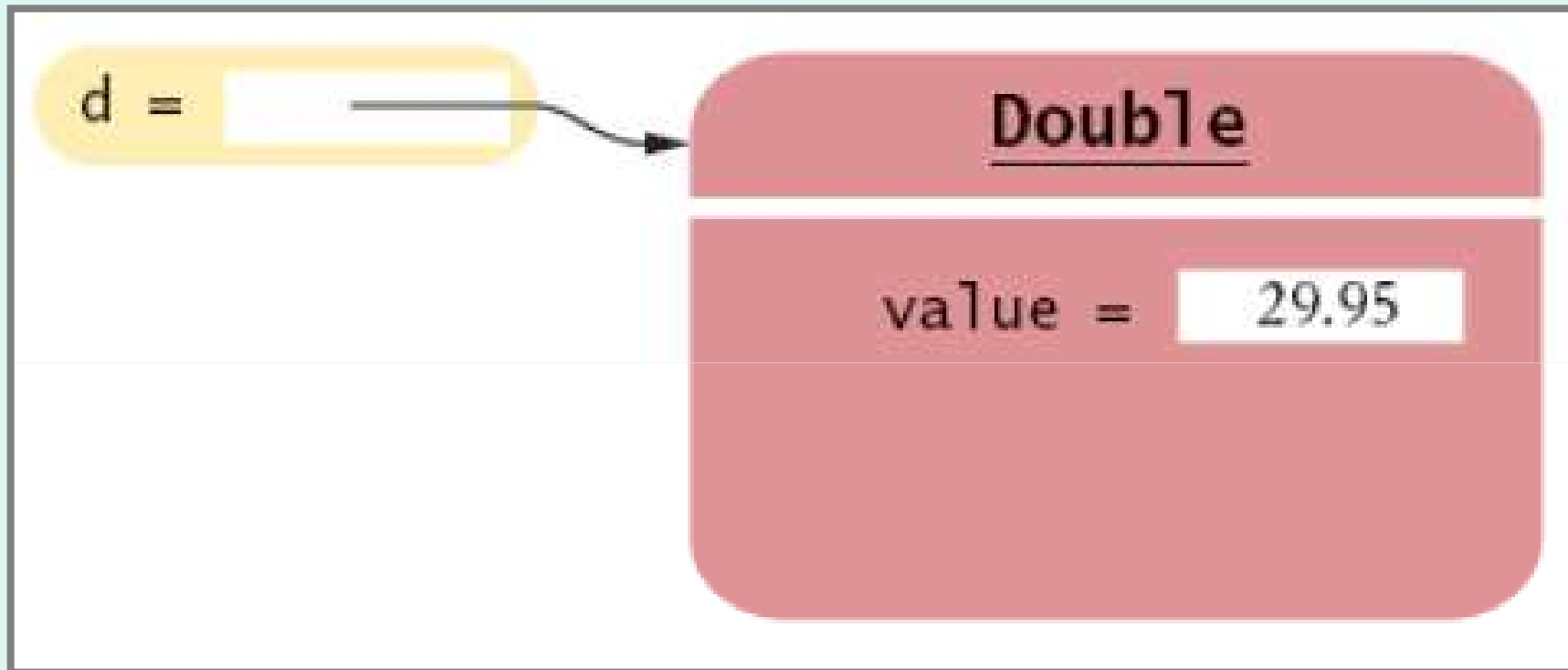
Wrappers

- A differenza degli array, le arraylists possono solo contenere elementi di tipo reference
- È necessario quindi utilizzare le cosiddette classi “wrapper” che permettono di convertire valori primitivi in corrispondenti valori di tipo riferimento:

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

Continua...

Wrappers



Wrappers

- Ci sono classi wrapper per ciascuno degli otto tipi primitivi

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

Auto-boxing

- Da Java 5.0, la conversione tra tipi primitivi e i corrispondenti tipi wrapper è automatica.

```
Double d = 29.95; // auto-boxing; equivale a
                  // Double d = new Double(29.95);

double x = d;      // auto-unboxing; equivale a
                  // double x = d.doubleValue();
```

Continua...

Auto-boxing

- Auto-boxing opera anche all'interno delle espressioni aritmetiche

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

Valutazione:

- auto-unbox **d** in un **double**
- aggiungi **1**
- auto-box il risultato in un nuovo **Double**
- Memorizza il riferimento nel wrapper **e**

Auto-boxing

Attenzione!

- **==** è definito diversamente sui tipi primitivi e sul loro corrispettivo boxed
- **==**
 - su interi significa uguaglianza di valori
 - su integer significa identità di riferimenti
- **Evitiamo l'uso di == sulle classi wrapper**

Domanda

- Supponiamo che `data` sia un `ArrayList<Double>` di dimensione > 0 . Come facciamo ad incrementare l'elemento all'indice 0?

Risposta

- `data.set(0, data.get(0) + 1);`

Array e *for* loops

- La soluzione tradizionale

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

“for each”: un for generalizzato

- Itera su tutti gli elementi di una collezione
 - ad esempio sugli elementi di una array

```
double[] data = . . .;  
double sum = 0;  
for (double e : data) // “per ciascun e in data”  
{  
    sum = sum + e;  
}
```

Continua...

“for each”

- **Si applica nello stesso modo alle ArrayLists:**

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

“for each”

- Equivalente al seguente loop tradizionale:

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

Sintassi

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

- **Esegue il corpo del ciclo su ciascun elemento della collezione**
- **La variabile è assegnata ad ogni ciclo all'elemento successivo**

“for each” – limitazioni

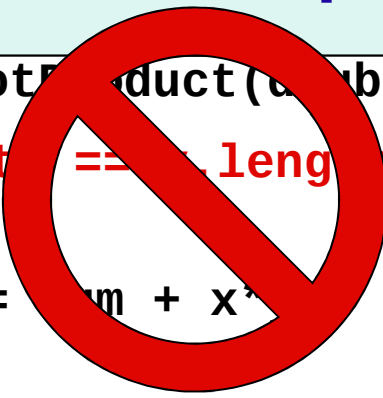
- **La sintassi è deliberatamente semplice**
- **Si applica solo ai casi più semplici di gestione di collezioni.**
- **Spesso è necessario utilizzare la sintassi tradizionale**

Continua...

“for each” – limitazioni

- Non utilizzabile per scorrere due strutture all'interno dello stesso loop

```
public static double dotProduct(double[] u, double[] v)
{
    // assumiamo u.length == v.length;
    double sum = 0,
    for (x:u, y:v) sum = sum + x*y;
}
```



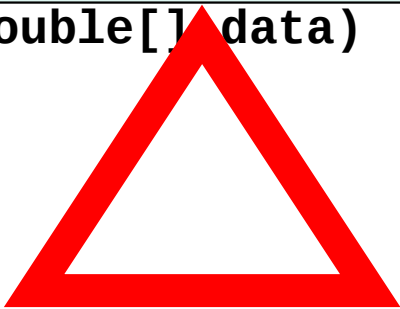
```
public static double dotProduct(double[] u, double[] v)
{
    // assumiamo u.length == v.length;
    double sum = 0,
    for (int i=0; i<u.length; i++) sum = sum + u[i]*v[i];
    return sum;
}
```

..

“for each” – limitazioni

- Non sempre utilizzabile per inizializzazioni

```
public static double dotProduct(double[] data)
{
    int i = 0;
    for (x:data) { x = i*i; i++; }
}
```



```
public static double dotProduct(double[] data)
{
    for (int i = 0; i<data.length; i++)
        data[i] = i*i;
}
```

Continua...

“for each” – limitazioni

- Iterazione su array bidimensionali
- tipicamente utilizziamo due cicli innestati: anche qui, il “foreach” non aiuta

```
for (int i = 0; i < ROWS; i++)  
    for (int j = 0; j < COLUMNS; j++)  
        board[i][j] = " ";
```

Esercizio

- Definiamo una classe per rappresentare un polinomio a coefficienti reali

$$C_0 + C_1X + C_2X^2 + \dots C_nX^n$$

Continua...