



Array ed ArrayList



Collezione di oggetti

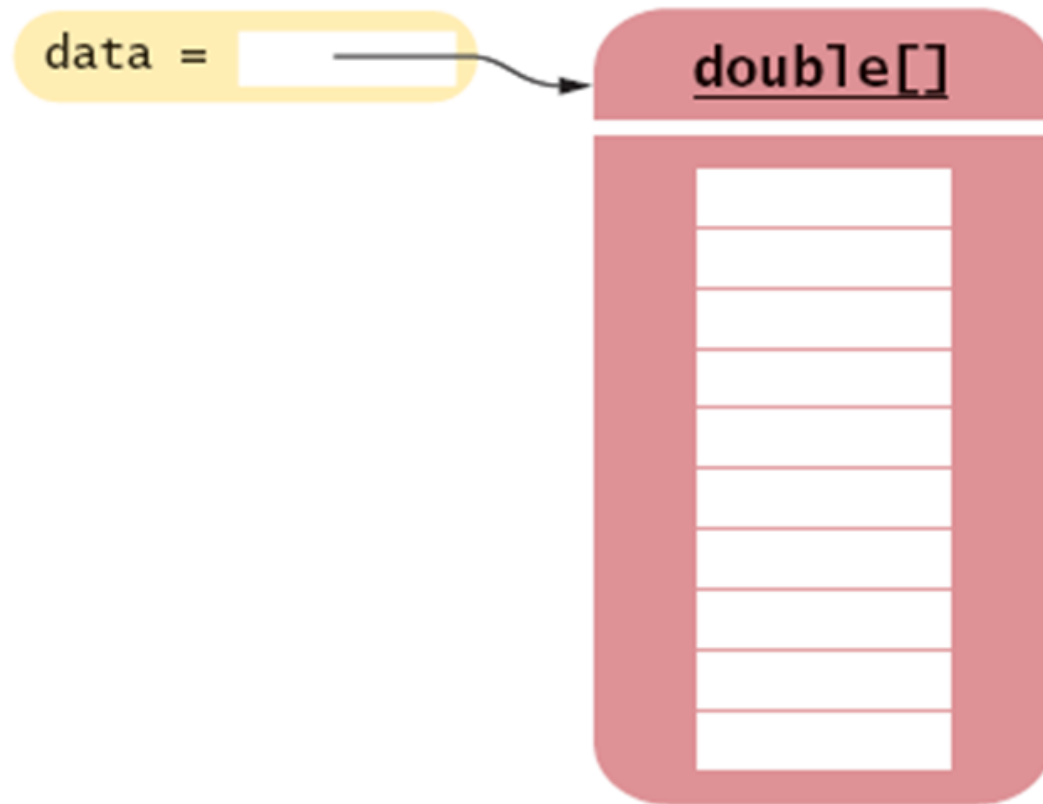
- Molto spesso è necessario manipolare grandi quantità di dati fra loro correlati (collezioni di dati **omogenei**)
- In generale, una collezione di oggetti è a sua volta un oggetto
- Java fornisce per le collezioni di dati
 - **array**
 - **ArrayList** (pacchetto **java.util**)

Collezione di oggetti: **array**

- Sequenza di lunghezza **prefissata** di valori dello **stesso tipo** (classe o tipo primitivo)
- Ogni posizione è individuata da un indice
- La prima posizione ha indice 0
- E' un **oggetto**
 - Deve essere creato con **new**, es. `new double[5]`
 - I valori sono inizializzati a 0 (per **int** o **double**), **false** (per **boolean**) o **null** (per oggetti)
 - Accesso attraverso variabili di riferimento

Riferimento ad array

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



Dichiarare un array

- Tipo array definito con: tipo dei dati seguito da parentesi quadre

- `int[] unSaccoDiNumeri;`
- `String[] vincitori;`
- `BankAccount[] contiCorrenti;`

- Esempio:

```
public static void main(String[] args)
```

- `args` è un array di stringhe (gli argomenti della linea di comando)

```
java MyProgram -d file.txt
```

```
args[0] = "-d"
```

```
args[1]= "file.txt"
```

Creare un'istanza di un array

- Per creare un'istanza di un array si usa **new** seguito dal tipo e quindi dalla grandezza in parentesi quadre:

```
int[] unSaccoDiNumeri;  
unSaccoDiNumeri = new int[10000];  
//un array di 10000 int
```

Dichiarazione ed inizializzazione

Table 1 Declaring Arrays

<pre>int[] numbers = new int[10];</pre>	An array of ten integers. All elements are initialized with zero.
<pre>final int NUMBERS_LENGTH = 10; int[] numbers = new int[NUMBERS_LENGTH];</pre>	It is a good idea to use a named constant instead of a “magic number”.
<pre>int valuesLength = in.nextInt(); double[] values = new double[valuesLength];</pre>	The length need not be a constant.
<pre>int[] squares = { 0, 1, 4, 9, 16 };</pre>	An array of five integers, with initial values.
<pre>String[] names = new String[3];</pre>	An array of three string references, all initially null.
<pre>String[] friends = { "Emily", "Bob", "Cindy" };</pre>	Another array of three strings.
<pre>double[] values = new int[10]</pre>	Error: You cannot initialize a double[] variable with an array of type int[].

Accesso agli elementi di un array

- Le parentesi quadrate [] consentono l'accesso agli elementi:

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

data =

double[]	
[0]	
[1]	
[2]	29.95
[3]	
[4]	
[5]	
[6]	
[7]	
[8]	
[9]	

Usare gli array

- Ogni elemento è una variabile:

```
int[] unSaccoDiNumeri;  
unSaccoDiNumeri = new int[10000];  
for (int i = 0; i < unSaccoDiNumeri.length; i++) {  
    unSaccoDiNumeri[i] = i;  
}  
System.out.println(unSaccoDiNumeri[0]);
```

- Range degli indici di **a**: 0, 1,, **a.length-1**
(**length** variabile di istanza che contiene numero elementi array,
non modificabile -> quindi costante)
- Se si usa un indice fuori dal range, viene sollevata a run-time
l'eccezione:
ArrayIndexOutOfBoundsException (**java.lang**)

Esempio:

- Stampiamo gli argomenti della linea di comando

```
public class PrintArgs {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            System.out.println(args[i]);  
        }  
    }  
}
```



Collezione di oggetti: **ArrayList**

- La classe **ArrayList** (pacchetto `java.util`) gestisce una sequenza di oggetti
- Può crescere e ridursi a piacimento
- La classe **ArrayList** implementa nei suoi metodi le operazioni più comuni su collezioni di elementi
 - inserimento
 - cancellazione
 - modifica
 - accesso dati

ArrayList

- La classe `ArrayList` è generica
 - contiene elementi di tipo `Object`
- (`Vettori parametrici`)

La classe `ArrayList<T>` contiene oggetti di tipo `T` (a partire da `Java 5.0`):

```
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 della collezione

File: Coin.java

```
public class Coin { //Una semplice classe Coin
    public Coin(double unValore, String unNome) {
        nome = unNome;
        valore = unValore;
    }
    public String daiNome() { return nome; }
    public double daiValore(){ return valore; }
    public boolean equals(Coin moneta){
        return nome.equals(moneta.daiNome());
    }
    private String nome;
    private double valore;
}
```

Aggiungere un elemento

- Per aggiungere l'elemento alla fine della collezione si usa il metodo **add(obj)** :

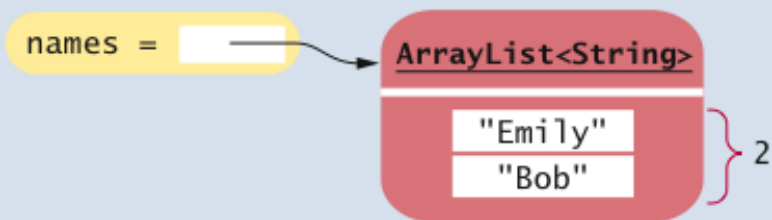
```
ArrayList<Coin> coins = new ArrayList<Coin>();  
coins.add(new Coin(0.1, "dime"));  
coins.add(new Coin(0.25, "quarter"));
```

- Dopo l'inserimento, la dimensione della collezione aumenta di uno

Aggiungere un elemento con add

```
names.add("Emily");  
names.add("Bob");  
names.add("Cindy");
```

1 Before add



2 After add

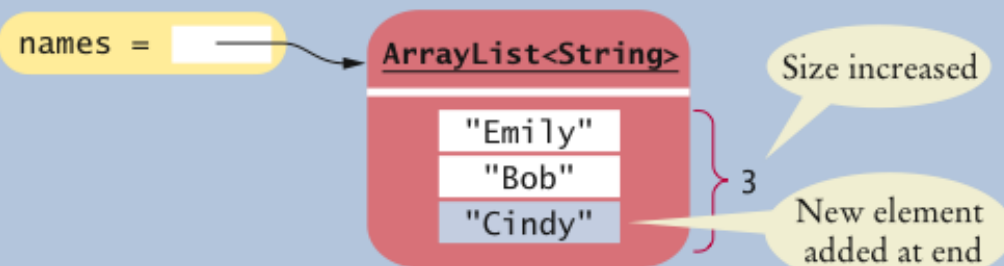


Figure 5 Adding an Element with add

Aggiungere un elemento

- Per aggiungere l'elemento in una certa posizione, facendo slittare in avanti gli altri, si usa il metodo `add(i, obj)`:

```
ArrayList<Coin> coins = new  
ArrayList<Coin>();  
coins.add(new Coin(0.1, "dime"));  
coins.add(new Coin(0.25, "quarter"));  
  
Coin aNickel = new Coin(0.05, "nickel");  
coins.add(1, aNickel);  
//quarter ora è il terzo oggetto della lista
```


Accedere agli elementi

- Bisogna usare il metodo `get(indice)`
`coins.get(2) ;`

- Nel caso generale `ArrayList` gestisce oggetti di tipo `Object`

- Possiamo passare qualsiasi oggetto al metodo `add`

```
ArrayList coins = new ArrayList();  
coins.add(new Rectangle(5, 10, 20, 30));
```

- Se definito con tipo `<T>` possiamo passare solo oggetti di tipo “compatibile” con `T`

Accedere agli elementi

- Se si usa `ArrayList` di tipo `Object` per utilizzare i metodi dell'oggetto inserito occorre fare il cast, altrimenti si possono solo usare i metodi di `Object`

```
Rectangle aCoin = (Rectangle) coins.get(i);  
aCoin.translate(x,y);
```

- Il cast ha successo solo se si usa il tipo corretto per l'oggetto considerato

```
Coin aCoin = (Coin) coins.get(i);  
//ERRORE  
//un Rectangle non può essere convertito in un  
Coin!
```

Accedere agli elementi

- Se si usa **ArrayList** di tipo **T**, il cast non è necessario

```
ArrayList<BankAccount> accounts =  
    new ArrayList<BankAccount>();  
BankAccount anAccount = accounts.get(i);  
anAccount.getBalance();
```

- Preferibile usare **ArrayList** parametrici
- Dalla versione 7 di Java è possibile scrivere
`ArrayList<String> names= new ArrayList<>();`

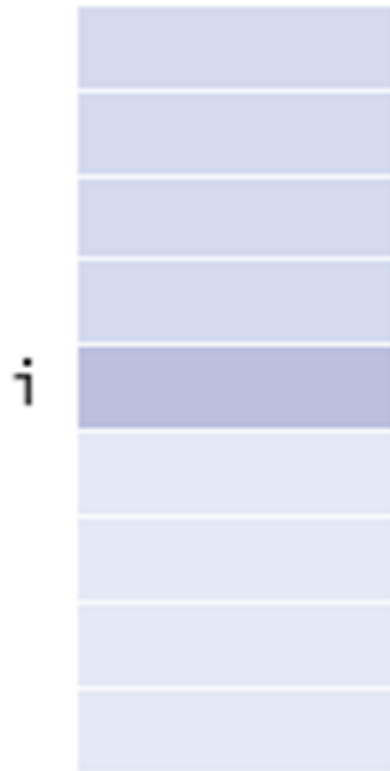
Rimuovere un elemento

- Per rimuovere un elemento da una collezione si usa il metodo **remove(indice)**
 - Restituisce l'oggetto rimosso
 - Gli elementi che seguono slittano di una posizione all'indietro

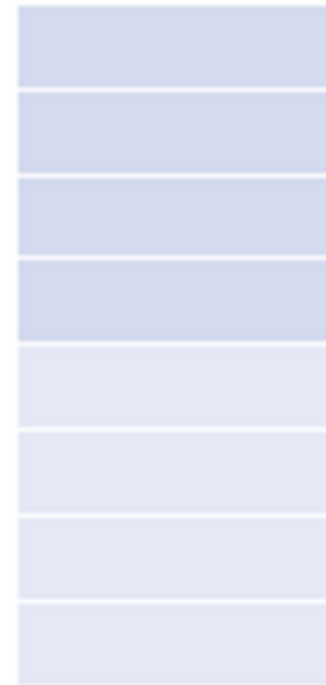
```
ArrayList<Coin> coins = new ArrayList<Coin>( );
coins.add(new Coin(0.1, "dime"));
coins.add(new Coin(0.25, "quarter"));
Coin aNickel = new Coin(0.05, "nickel");
coins.add(1, aNickel);
coins.remove(0);

//il vettore ora ha due elementi:
// quarter e nickel
```

Eliminare l'elemento alla i -esima posizione:
invocare metodo **remove(i)**



Prima



Dopo

Modificare un elemento

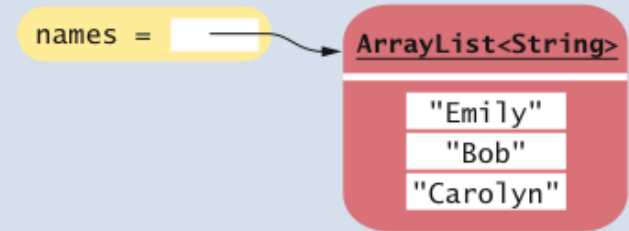
- Si usa il metodo `set(indice, obj)`
 - Restituisce l'oggetto rimpiazzato

```
ArrayList coins = new ArrayList();  
coins.add(new Coin(0.1, "dime"));  
coins.add(new Coin(0.25, "quarter"));  
Coin aNickel = new Coin(0.05, "nickel");  
coins.set(0, aNickel);  
//la posizione 0 viene sovrascritta
```

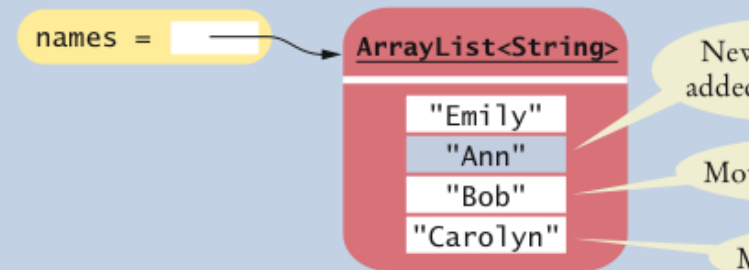
Aggiungere e Rimuovere Elementi

```
names.add("Emily");  
names.add("Bob");  
names.set(2, "Carolyn"); ①  
names.add(1, "Ann"); ②  
names.remove(1); ③
```

① Before add



② After `names.add(1, "Ann")`



③ After `names.remove(1)`

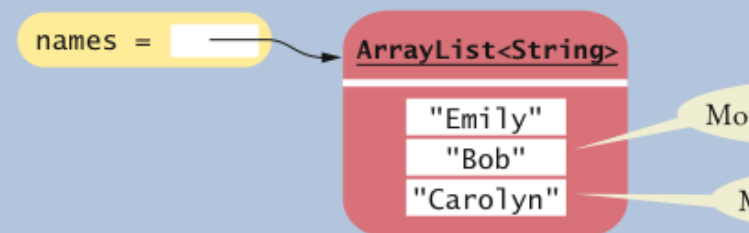


Figure 6 Adding and Removing Elements in the Middle of a List

File: BankAccount.java

```
01: /**
02:     A bank account has a balance that can be changed by
03:     deposits and withdrawals.
04: */
05: public class BankAccount
06: {
07:     /**
08:         Constructs a bank account with a zero balance
09:         @param anAccountNumber the account number for this account
10:     */
11:     public BankAccount(int anAccountNumber)
12:     {
13:         accountNumber = anAccountNumber;
14:         balance = 0;
15:     }
16:
17:     /**
18:         Constructs a bank account with a given balance
19:         @param anAccountNumber the account number for this account
20:         @param initialBalance the initial balance
21:     */
```


File: BankAccount.java

```
22:     public BankAccount(int anAccountNumber, double initialBalance)
23:     {
24:         accountNumber = anAccountNumber;
25:         balance = initialBalance;
26:     }
27:
28:     /**
29:      Gets the account number of this bank account.
30:      @return the account number
31:     */
32:     public int getAccountNumber()
33:     {
34:         return accountNumber;
35:     }
36:
37:     /**
38:      Deposits money into the bank account.
39:      @param amount the amount to deposit
40:     */
41:     public void deposit(double amount)
42:     {
43:         double newBalance = balance + amount;
44:         balance = newBalance;
45:     }
```

File: BankAccount.java

```
46:
47:     /**
48:         Withdraws money from the bank account.
49:         @param amount the amount to withdraw
50:     */
51:     public void withdraw(double amount)
52:     {
53:         double newBalance = balance - amount;
54:         balance = newBalance;
55:     }
56:
57:     /**
58:         Gets the current balance of the bank account.
59:         @return the current balance
60:     */
61:     public double getBalance()
62:     {
63:         return balance;
64:     }
65:
66:     private int accountNumber;
67:     private double balance;
68: }
```

File: ArrayListTester.java

```
01: import java.util.ArrayList;
02:
03: /**
04:     This program tests the ArrayList class.
05: */
06: public class ArrayListTester
07: {
08:     public static void main(String[] args)
09:     {
10:         ArrayList<BankAccount> accounts
11:             = new ArrayList<BankAccount>();
12:         accounts.add(new BankAccount(1001));
13:         accounts.add(new BankAccount(1015));
14:         accounts.add(new BankAccount(1729));
15:         accounts.add(1, new BankAccount(1008));
16:         accounts.remove(0);
17:
18:         System.out.println("Size: " + accounts.size());
19:
20:         BankAccount first = accounts.get(0);
```

File: ArrayListTester.java

```
21:         System.out.println("First account number: "  
22:             + first.getAccountNumber());  
23:  
24:         BankAccount last = accounts.get(accounts.size() - 1);  
25:         System.out.println("Last account number: "  
26:             + last.getAccountNumber());  
27:  
28:     }  
29: }
```

Output

size=3

first account number=1008

last account number=1729

Nuova classe Purse

```
import java.util.ArrayList;

public class Purse{
    public Purse() {
        coins = new ArrayList<Coin>();
    }
    public void add(Coin aCoin) {
        coins.add(aCoin);
    }
    public double getTotal() {
        double total = 0;
        for (int i = 0; i < coins.size(); i++) {
            Coin aCoin = coins.get(i);
            total = total + aCoin.getValue();
        }
        return total;
    }
    private ArrayList<Coin> coins;
}
```

Range degli indici per **ArrayList**

- Gli indici ammissibili per i metodi che fanno riferimento ad oggetti memorizzati (**get**, **remove**, **set**,..) sono:
 $0, 1, \dots, \text{size}() - 1$
- Gli indici ammissibili per i metodi che inseriscono nuove posizioni (**add**) sono:
 $0, 1, \dots, \text{size}()$
- Se si specifica un indice fuori da questi domini viene generata a run-time l'eccezione:
IndexOutOfBoundsException (java.lang)

Lunghezza e taglia

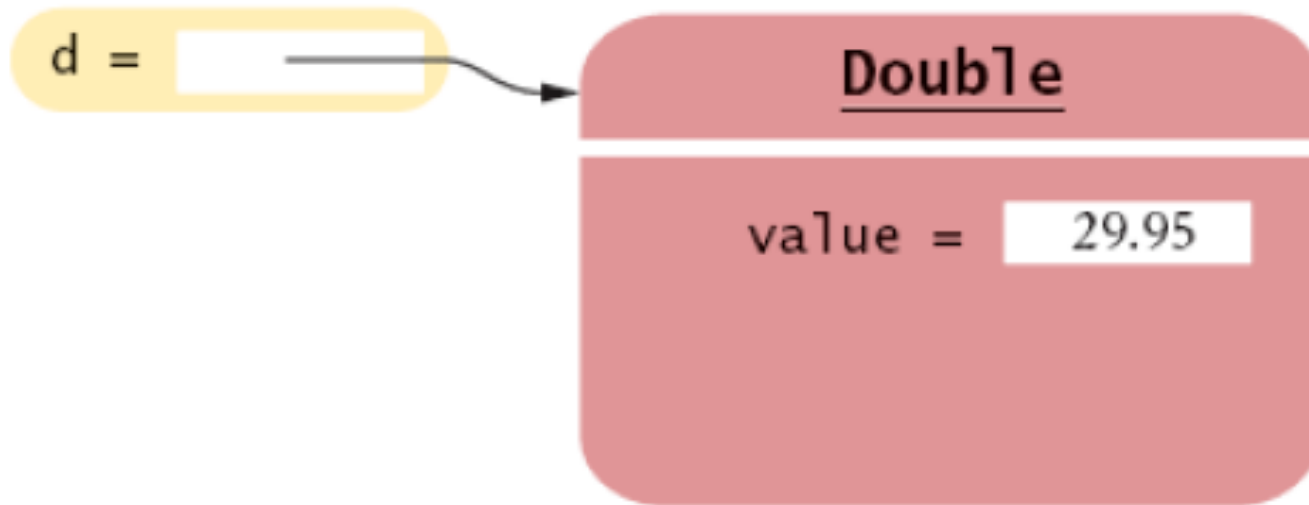
- La sintassi Java per determinare il numero di elementi in un array, un array list ed una stringa non è consistente:
- array **a** **a.length** (var. istanza final)
- array list **a** **a.size()** (metodo)
- stringa **a** **a.length()** (metodo)

Memorizzare dati primitivi in vettori

- **ArrayList** memorizza oggetti
- Per i dati primitivi si utilizzano classi wrapper (involucro)

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

Wrappers



Auto-boxing

- Auto-boxing: A partire da Java 5.0, la conversione tra i tipi primitivi e le corrispondenti classi wrapper è automatica.

```
Double d = 29.95;
```

```
// auto-boxing;
```

```
// versioni precedenti Java 5.0:
```

```
// Double d = new Double(29.95);
```

```
double x = d;
```

```
// auto-unboxing;
```

```
// versioni precedenti Java 5.0: x = d.doubleValue();
```

Auto-boxing

- Conversioni per auto-boxing avvengono anche all'interno di espressioni

Double e = d + 1;

Significa:

- converti **d** in un **double** (unbox)
- aggiungi **1**
- Impacchetta il risultato in un nuovo **Double**
- Memorizza in **e** il riferimento all'oggetto appena creato

Il ciclo for generalizzato (Java 5.0)

- Scandisce tutti gli elementi di una collezione:

```
double[] data = . . . ;  
double sum = 0;  
for (double e: data) // va letto come "per ogni e in data"  
{  
    sum = sum + e;  
}
```

- Alternativa tradizionale:

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

Esempio

- For generalizzato:

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

- Alternativa

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

Self check

- Scrivere un ciclo "for each" che stampa tutti gli elementi nell'array data
- Perché il ciclo "for each" non è adatto a rappresentare il seguente ciclo?

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

Risposte

- `for (double x : data) System.out.println(x) ;`
- Il ciclo "for each" non ha l'indice i.

Ricerca Lineare

```
public class Purse
{
    public boolean find(Coin aCoin)
    {
        for (Coin c: coins)
        {
            if (c.equals(aCoin))
                return true; //trovato
        }
        return false; //non trovato
    }
    ...
}
```

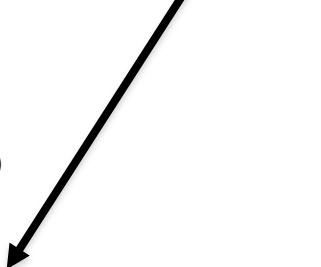

Contare elementi di un certo tipo

```
public class Purse
{
    public int count(Coin aCoin)
    {
        int matches = 0;
        for (Coin c: coins)
        {
            if (c.equals(aCoin))
                matches++;
                                //found a match
        }
        return matches;
    }
    ...
}
```

```
if (coins.size() == 0) return null;
```

Trovare il massimo

```
public class Purse
{
    public Coin getMaximum()
    {
        //inizializza il max al primo valore
        Coin max = coins.get(0);
        for (Coin c: coins)
        {
            if (c.daiValore() > max.daiValore())
                max = c;
        }
        return max;
    }
    ...
}
```

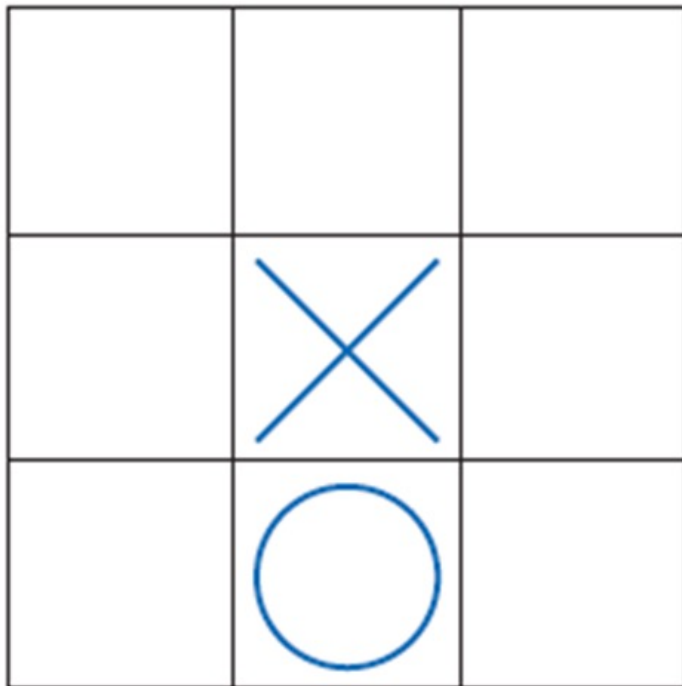


Trovare il minimo

```
public class Purse
{
    public Coin getMinimum()
    {
        if (coins.size() == 0) return null;
        Coin min = coins.get(0);
        for (Coin c: coins)
        {
            if (c.daiValore() < min.daiValore()) min = c;
        }
        return min;
    }
    ...
}
```

Array a due dimensioni

- Tabella con righe e colonne
- Esempio: la scacchiera del gioco Tris



```
String[][] board = new String[3][3];  
//array di 3 righe e 3 colonne  
board[i][j] = "x";  
// accedi all'elemento della riga  
// i e colonna j
```

Classe Tris

```
/**
    Una scacchiera 3x3 per il gioco Tris.
 */
public class Tris{
    /**
        Costruisce una scacchiera vuota.
    */
    public Tris(){
        board = new String[ROWS][COLUMNS];
        // riempi di spazi
        for (int i = 0; i < ROWS; i++)
            for (int j = 0; j < COLUMNS; j++)
                board[i][j] = " ";
    }
}
```

Classe Tris

```
/**
    Crea una rappresentazione della scacchiera
    in una stringa, come ad esempio
    |x o|
    | x |
    | o|
    @return la stringa rappresentativa
*/
public String toString()
{
    String r = "";
    for (int i = 0; i < ROWS; i++)
    {
        r = r + "|";
        for (int j = 0; j < COLUMNS; j++)
            r = r + board[i][j];
        r = r + "|\n";
    }
    return r;
}
```

Classe Tris

```
/**
    Imposta un settore della scacchiera.
    Il settore deve essere libero.
    @param i l'indice di riga
    @param j l'indice di colonna
    @param player il giocatore ('x' o 'o')
 */
public void set(int i, int j, String player)
{
    if (board[i][j].equals(" "))
        board[i][j] = player;
}
private String[ ][ ] board;
private static final int ROWS = 3;
private static final int COLUMNS = 3;
}
```

File TrisTester.java

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

```
/**
```

```
Questo programma collauda la classe Tris  
chiedendo all'utente di selezionare posizioni sulla  
scacchiera e visualizzando il risultato.
```

```
*/
```

```
public class TrisTester
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        String player = "x";
```

```
        Tris game = new Tris();
```

```
        Scanner in = new Scanner(System.in);
```


File TrisTester.java

```
boolean done = false;
while(!done) {
    System.out.println(game.toString());
    System.out.println("Inserisci riga per " +
                        player + "(-1 per uscire):");
    int riga = in.nextInt();
    if (riga < 0) done=true;
    else{
        System.out.println("Inserisci colonna per " +
                            player + ":");
        int colonna = in.nextInt();
        game.set(riga, colonna, player);
        if (player.equals("x")) player = "o";
        else player = "x";
    }
}
}
```

Output

| |

| |

| |

Row for x (-1 to exit): 1

Column for x: 2

| |

| x|

| |

Row for o (-1 to exit): 0

Column for o: 0

|o |

| x|

| |

Row for x (-1 to exit): -1

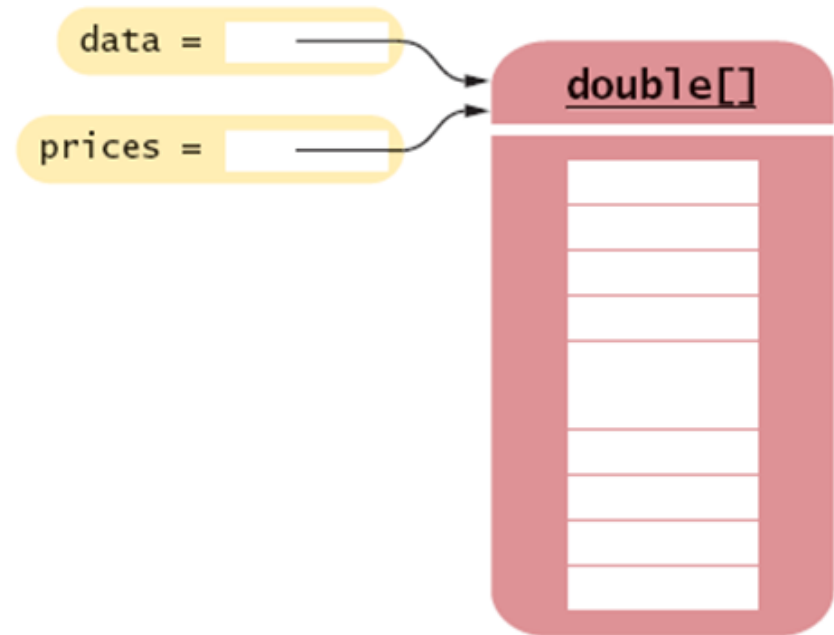
Copiare Array

- Una variabile array memorizza un riferimento all'array
- Copiando la variabile otteniamo un secondo riferimento allo stesso array

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

```
// riempi array . . .
```

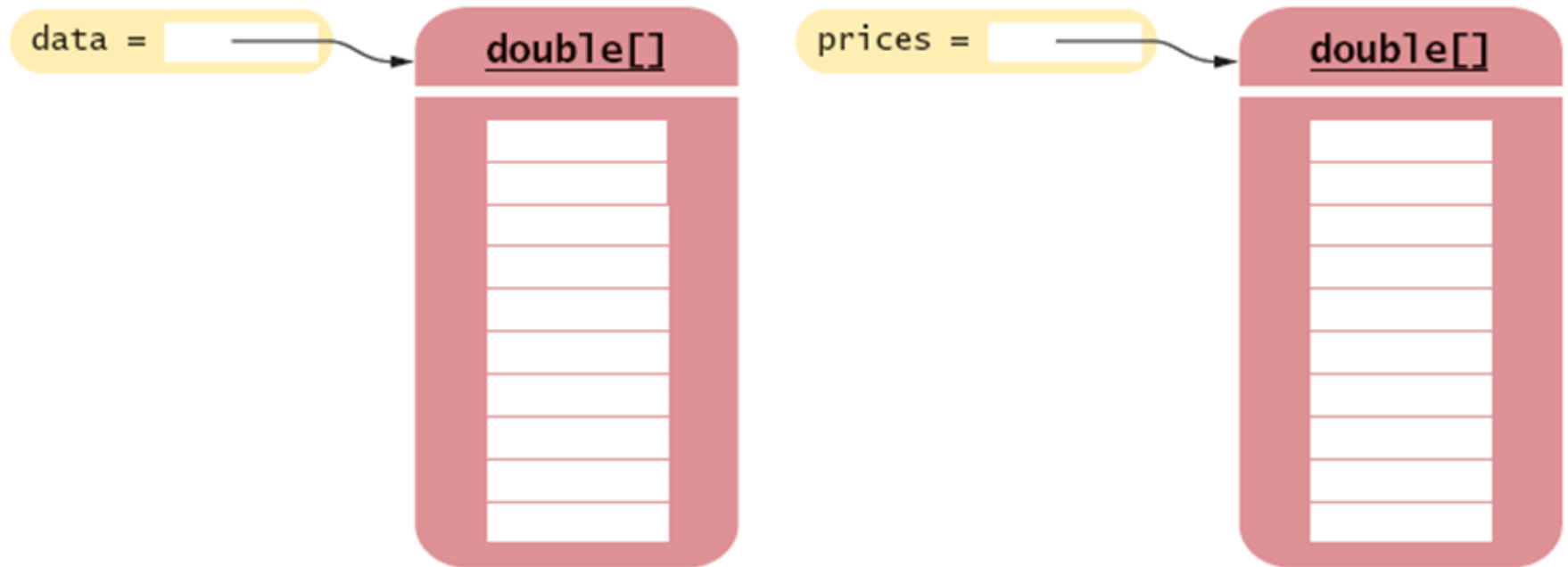
```
double[] prices = data;
```



Copiare Array

- Per fare una vera copia occorre invocare il metodo **clone**

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

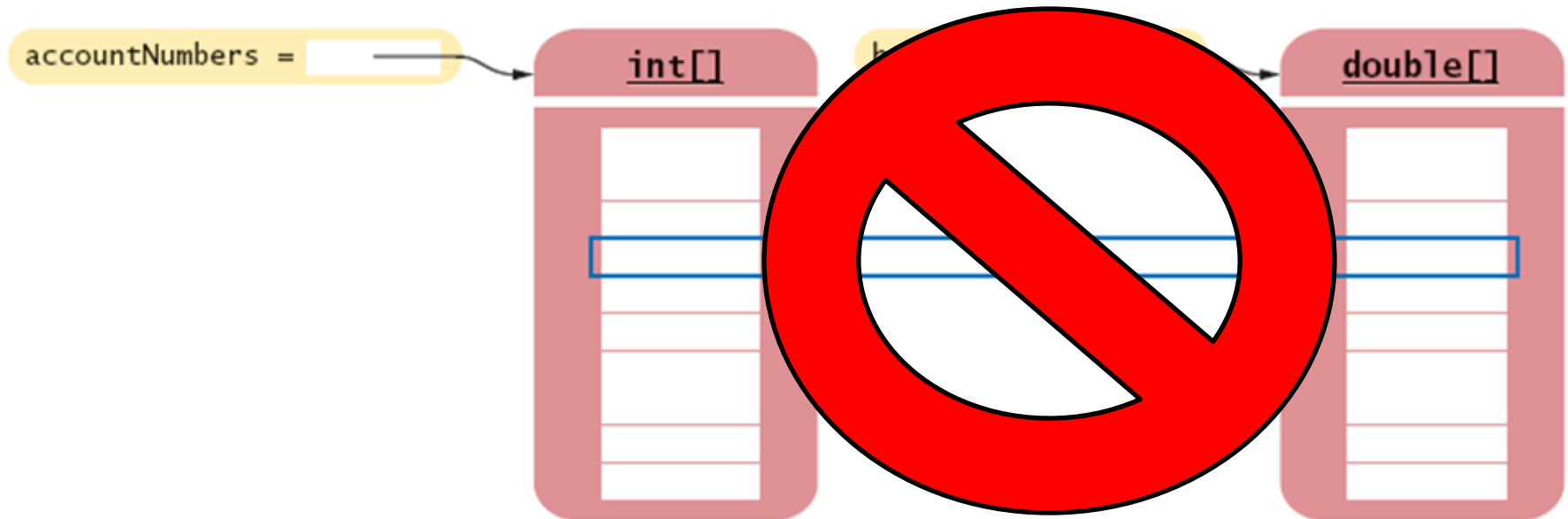


Array paralleli

- Non utilizzate array paralleli

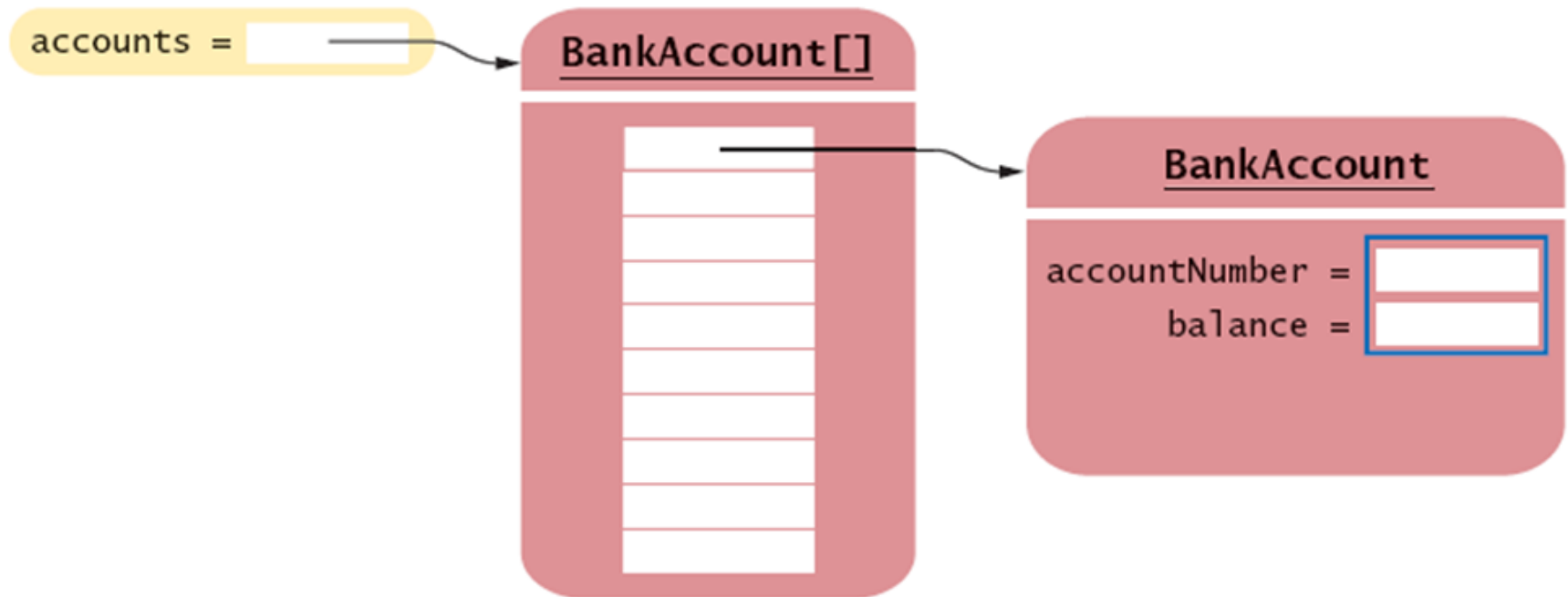
```
String[] names;
```

```
double[] salaries;
```



Array di oggetti

- Riorganizzate i dati in array di oggetti
BankAccount[] accounts;



Metodi con numero variabile di parametri

- Dalla versione 5 di Java è possibile dichiarare metodi che ricevono un numero variabile di parametri

```
data.add(1, 3, 7);  
data.add(4);  
data.add();
```

- Il metodo `add()` deve essere dichiarato come:

```
public void add(double ... xs);
```

- I puntini `...` indicano che il metodo può ricevere un numero qualsiasi di valori `double`
- L'implementazione del metodo analizza l'array dei parametri ed elabora i valori

```
for (x : xs) { sum = sum + x; }
```