Vettori e Array

Esempio

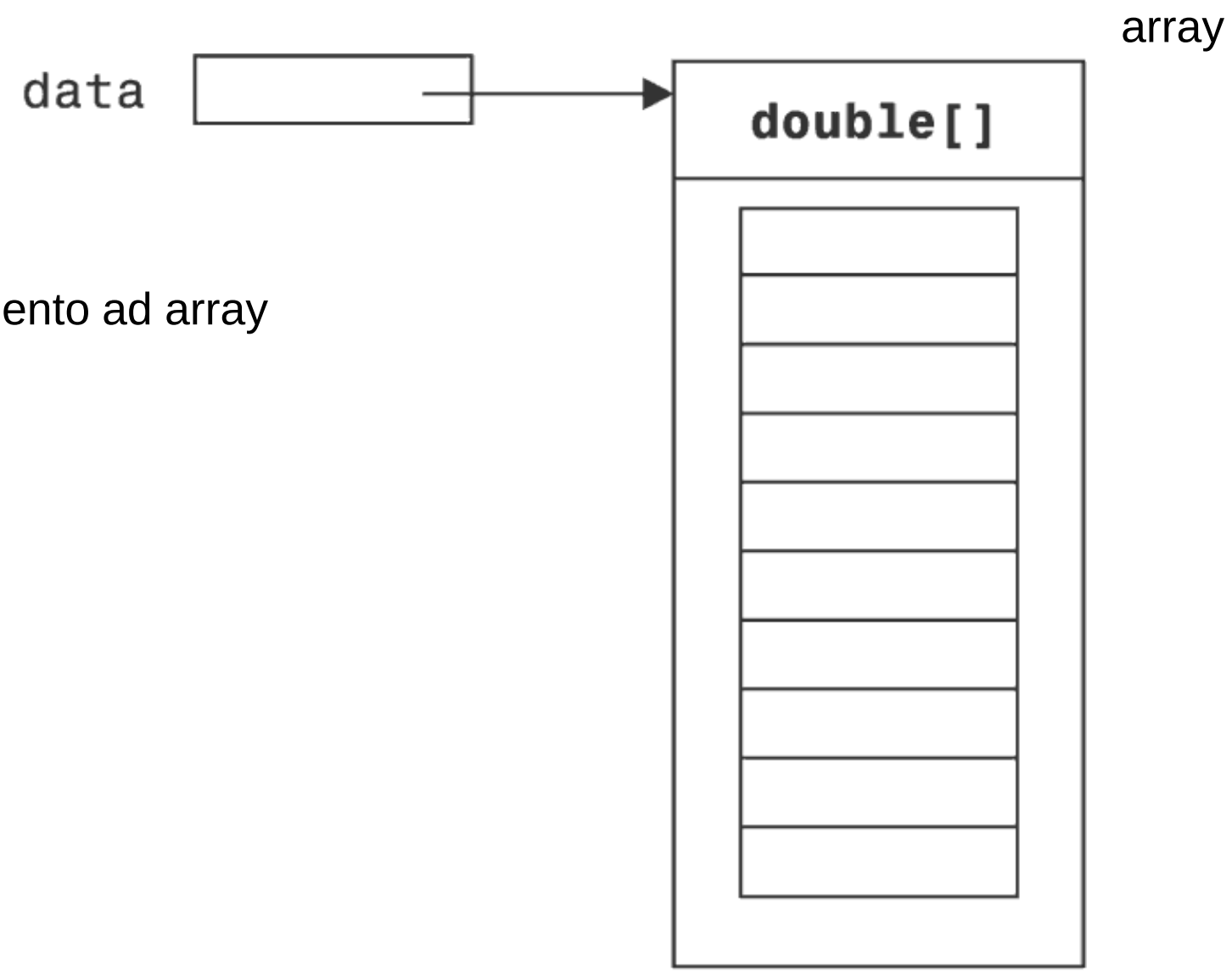
```
public class Purse
{

    public Purse()
    {
        nickels = 0;
        dimes = 0;
        quarters = 0;
    }

    public void addNickels(int count)
    {
        nickels = nickels + count;
    }

    public void addDimes(int count)
    {
        dimes = dimes + count;
    }

    public void addQuarters(int count)
    {
        quarters = quarters + count;
    }

    public double getTotal()
    {
        return nickels * NICKEL_VALUE
          + dimes * DIME_VALUE + quarters *
                QUARTER_VALUE;
    }

    private static final double NICKEL_VALUE = 0.05;
    private static final double DIME_VALUE = 0.1;
    private static final double QUARTER_VALUE = 0.25;

    private int nickels;
    private int dimes;
    private int quarters;
}
```

Collezione di oggetti

- La classe `Purse` non tiene traccia delle monete (come entità), ma memorizza solo il numero di monete di ogni tipo
- In Java, una collezione di oggetti è a sua volta un oggetto
- Per le collezioni di dati in Java
  - 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**
  - 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

Dichiarare un array

- Tipo array = tipo 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
```

Usare gli array

- Ogni elememento 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)
- Se si usa un indice fuori dal range, viene sollevata a runtime l'eccezione:

  ArrayIndexOutOfBoundsException (java.lang)

Esempio:

- Stampiamo gli argomenti della linea di comando

```
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 decrescere a piacimento
- La classe `ArrayList` implementa nei suoi metodi le operazioni più comuni su collezioni di elementi
  - inserimento
  - cancellazione
  - modifica
  - accesso ai dati

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;
}
```

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:
```

File: BankAccount.java

```
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:    */
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:    }
```

File: BankAccount.java

```
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:    }
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;
```

File: BankAccount.java

```
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: }
```

ArrayList

- La classe `ArrayList` è generica (parametrica)
  - contiene elementi di tipo Object
- (Vettori parametrici) La classe `ArrayList<T>` contiene oggetti di tipo `T` (introdotta 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

Aggiungere un elemento

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

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

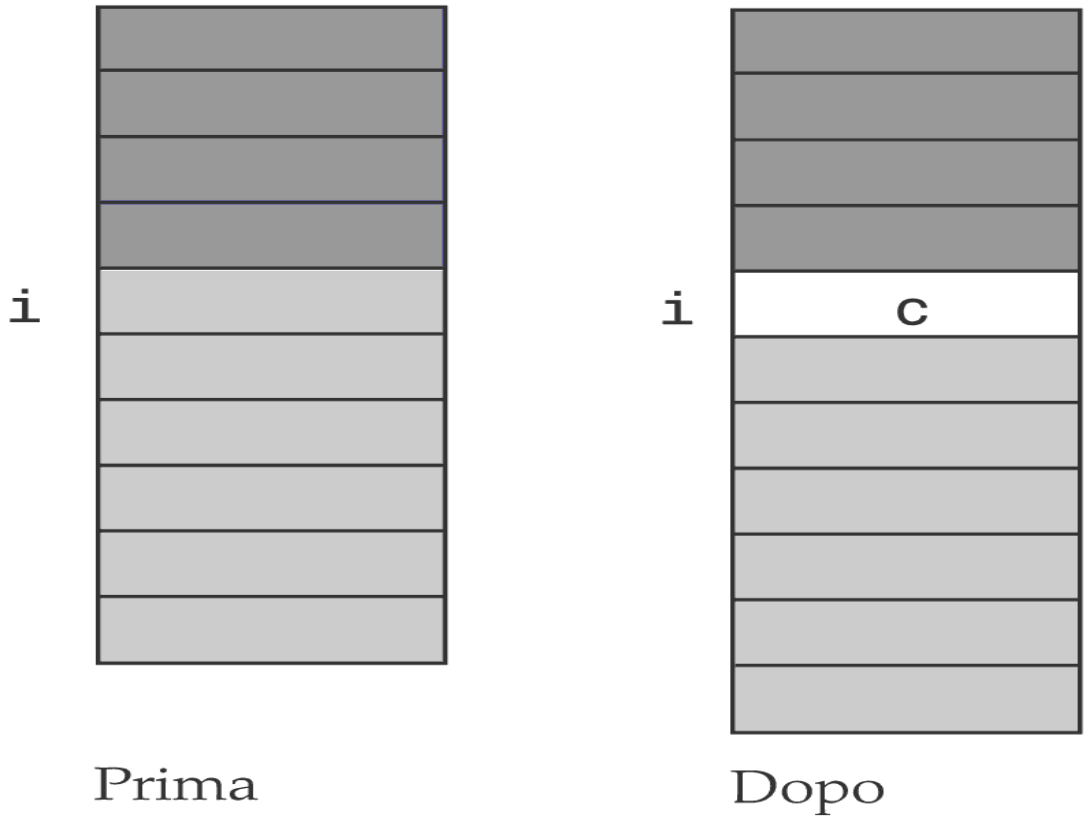
- Dopo l'inserimento, la dimensione della collezione aumenta di uno

Aggiungere un elemento

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

```
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.add(1, aNickel);
//quarter ora è il terzo oggetto della lista
```

Aggiungere un elemento c alla posizione i: invocazione add(i, c)



Accedere agli elementi

- Bisogna usare il metodo get(indice)

```
coins.get(2);
```

- Nel caso generale ArrayList gestisce oggetti di tipo Object (classe grezza)
  - Possiamo passare qualsiasi oggetto al metodo add

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

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

Accedere agli elementi

- Se si usa la versione grezza di ArrayList per utilizzare i metodi dell'oggetto inserito occorre fare il casting, altrimenti si possono solo usare i metodi di Object

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

- Il casting 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 parametrico, il cast non è necessario

```
ArrayList<BankAccount> accounts =
                        new ArrayList<BankAccount>();

BankAccount anAccount = accounts.get(i);
anAccount.getBalance();
```

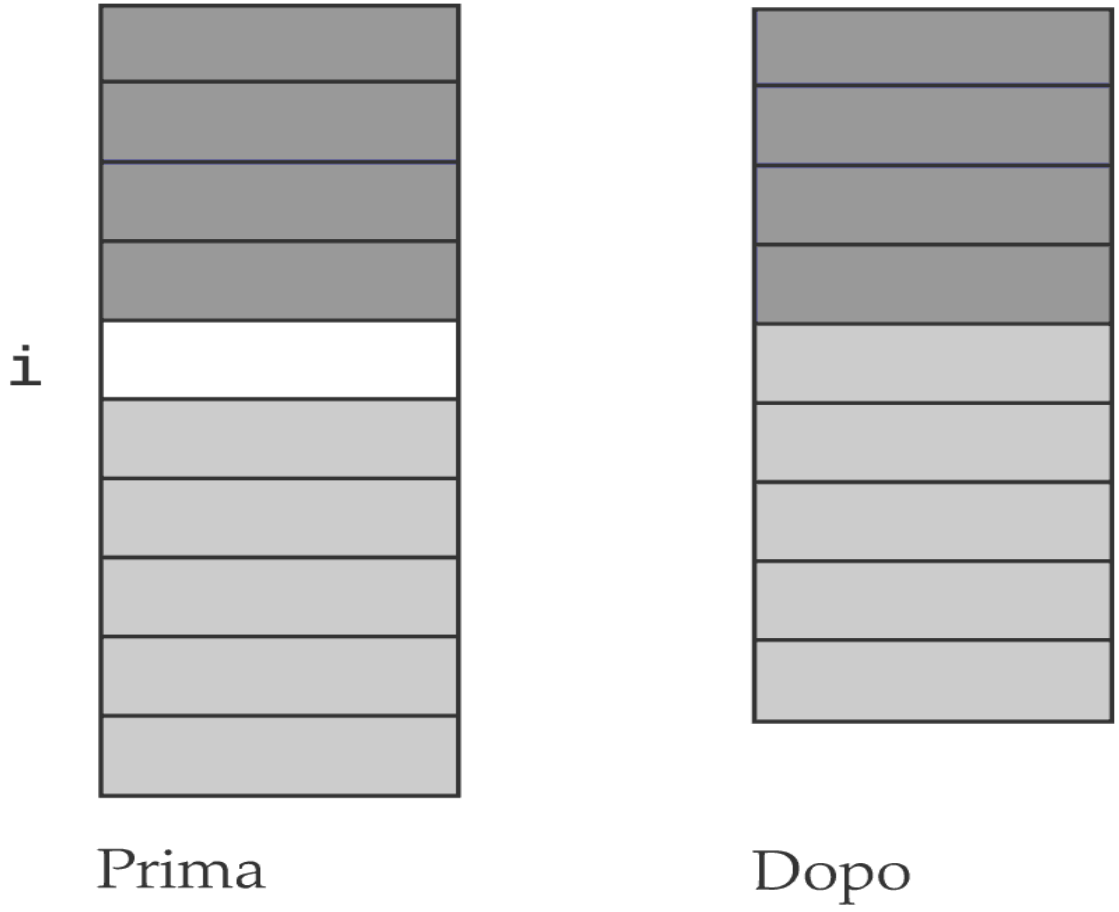
- Preferibile usare ArrayList parametrici

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)



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”);
Coin previousCoin = coins.set(0, aNickel);

//la posizione 0 viene sovrascritta
```

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);
```

File: ArrayListTester.java

```
17:
18:       System.out.println("size=" + accounts.size());
19:       BankAccount first = accounts.get(0);
20:       System.out.println("first account number="
21:             + first.getAccountNumber());
22:       BankAccount last = accounts.get(accounts.size() - 1);
23:       System.out.println("last account number="
24:             + last.getAccountNumber());
25:    }
26: }
```

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;
}
```

Classe Purse: getTotal( ) efficiente

```
import java.util.ArrayList;

public class Purse{

   public Purse(){
      coins = new ArrayList<Coin>();
      total = 0;
   }

   public void add(Coin aCoin){
      coins.add(aCoin);
      total += aCoin.getValue();
   }

   public double getTotal(){
      return total;
   }
   private ArrayList<Coin> coins;
   private double total;
}
```

Range degli indici per ArrayList

- Gli indici ammissibili per i metodi che fanno riferimento ad oggetti memorizzati (get, remove, set,..) sono:

  0,1,……, size()-1

- Gli indici ammissibili per i metodi che inseriscono nuove posizioni (add) sono:

  0,1,……, size()

- Se si specifica un indice fuori da questi domini viene generata a runtime l'eccezione:

  IndexOutOfBoundsException (java.lang)

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

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` generico (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 generico:

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

- Alternativa tradizionale:

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

Ricerca Lineare

```
public class Purse
{
   public boolean hasCoin(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;
    }
      ...
}
```

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()
   {
      //inizializza il min al primo valore
      Coin min =(Coin) coins.get(0);
      for (Coin c: coins)
      {
          if (c.daiValore() < min.daiValore())
                min =c;
      }
      return min;
   }
   ...
}
```

File 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:    */
```

File Bank.java

```
20:    public void addAccount(BankAccount a)
21:    {
22:       accounts.add(a);
23:    }
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:
```

File Bank.java

```
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: {
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:
```

File Bank.java

```
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
59:       if there 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
69: }
70:
```

File Bank.java

```
71:    /**
72:       Gets the bank account with the largest balance.
73:       @return the account with the largest balance, or
74:       null if the 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: }
```

File BankTester.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(c + " accounts with balance >= "
      + threshold);
```

File BankTester.java

```
16:
17:         int accountNumber = 1015;
18:         BankAccount a = firstBankOfJava.find(accountNumber);
19:         if (a == null)
20:            System.out.println("No account with number "
      + accountNumber);
21:         else
22:            System.out.println("Account with number "
      + accountNumber
23:                   + " has balance " + a.getBalance());
24:
25:         BankAccount max = firstBankOfJava.getMaximum();
26:         System.out.println("Account with number "
27:               + max.getAccountNumber()
28:               + " has the largest balance.");
29:     }
30: }
```

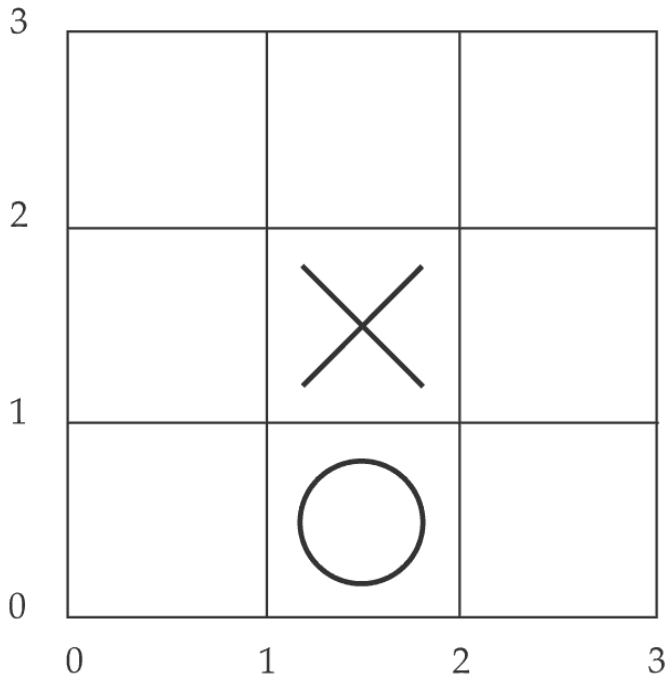
File BankTester.java

Output

```
2 accounts with balance >= 15000.0
Account with number 1015 has balance 10000.0
Account with number 1001 has the largest balance.
```

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] = " ";
   }
```

```
/**
      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;
   }
```

```
    /**
       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 TrisStarter.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 TrisStarter
{
   public static void main(String[] args)
   {
      String player = "x";

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

```
  while(true){
    System.out.println(game.toString());

    System.out.println("Inserisci riga per " + player +
                       "(-1 per uscire):");

    int riga = in.nextInt();

    if (riga < 0) return;

    System.out.println("Inserisci colonna per "+player":");
    int colonna = in.nexInt();

    game.set(row, column, player);
    if (player == "x") player = "o";
    else player = "x";
   }
  }
}
```

For generico con array bidimensionali

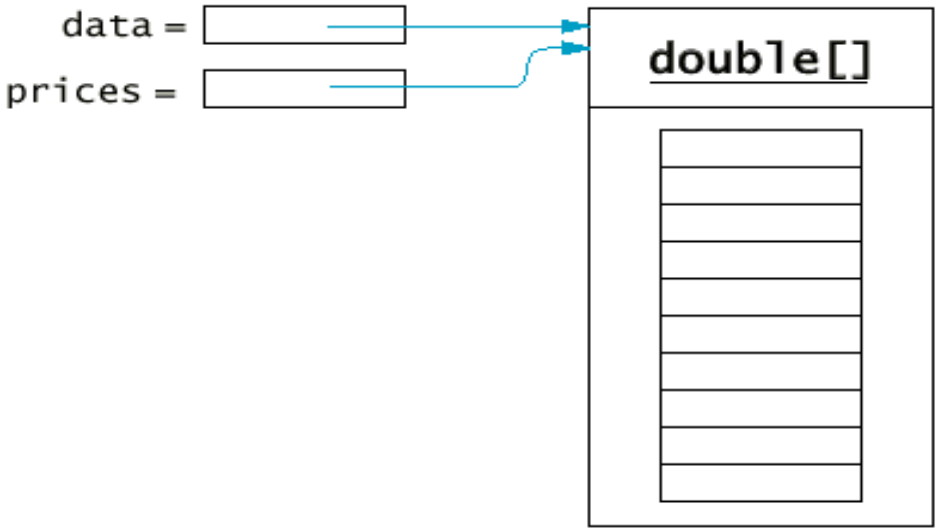
- Array bidimensionale = array di array monodimensionali
- Es:

```
int[][] a = new int[2][2];

for(int[] r: a)
    for (int x: r)
        System.out.println(x);
```

Copiare Array

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

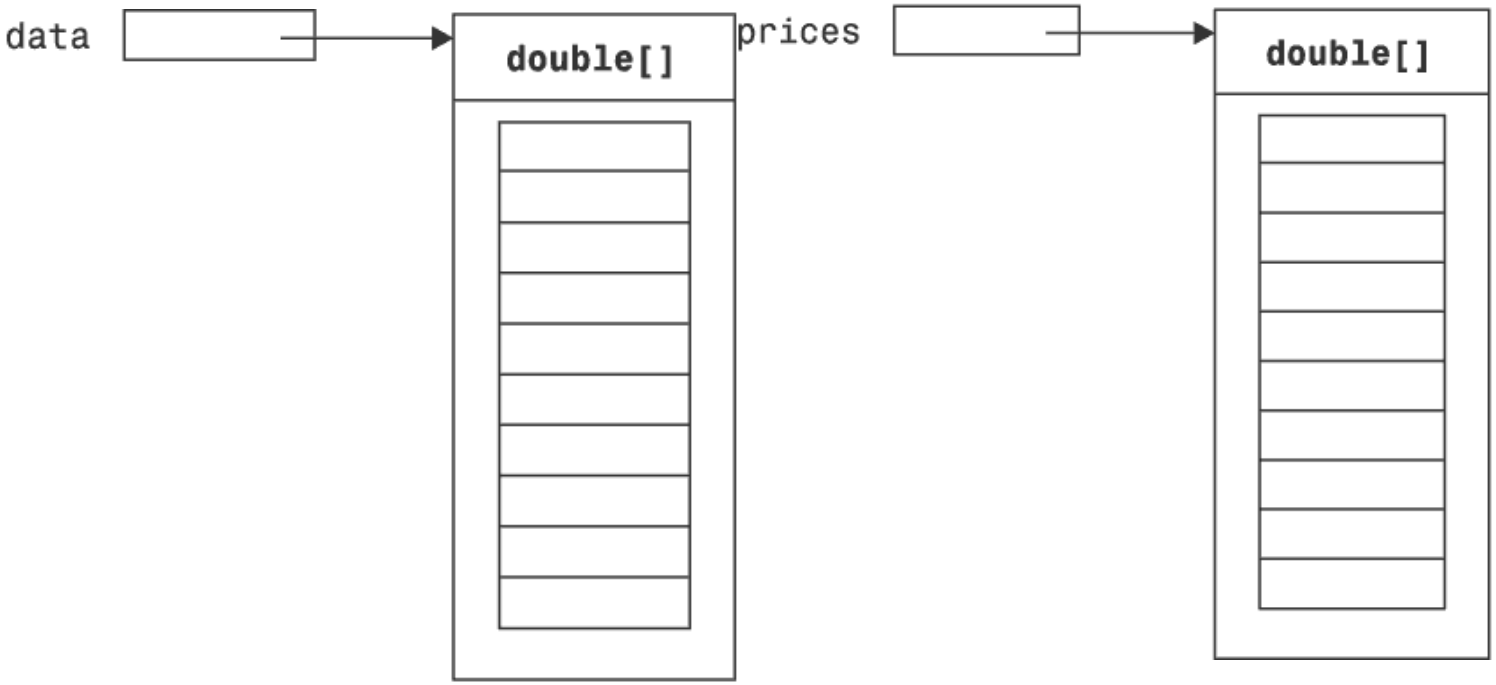
```
double[] data = new double[10];
  // riempi array . . .
  double[] prices = data;
```



Copiare Array

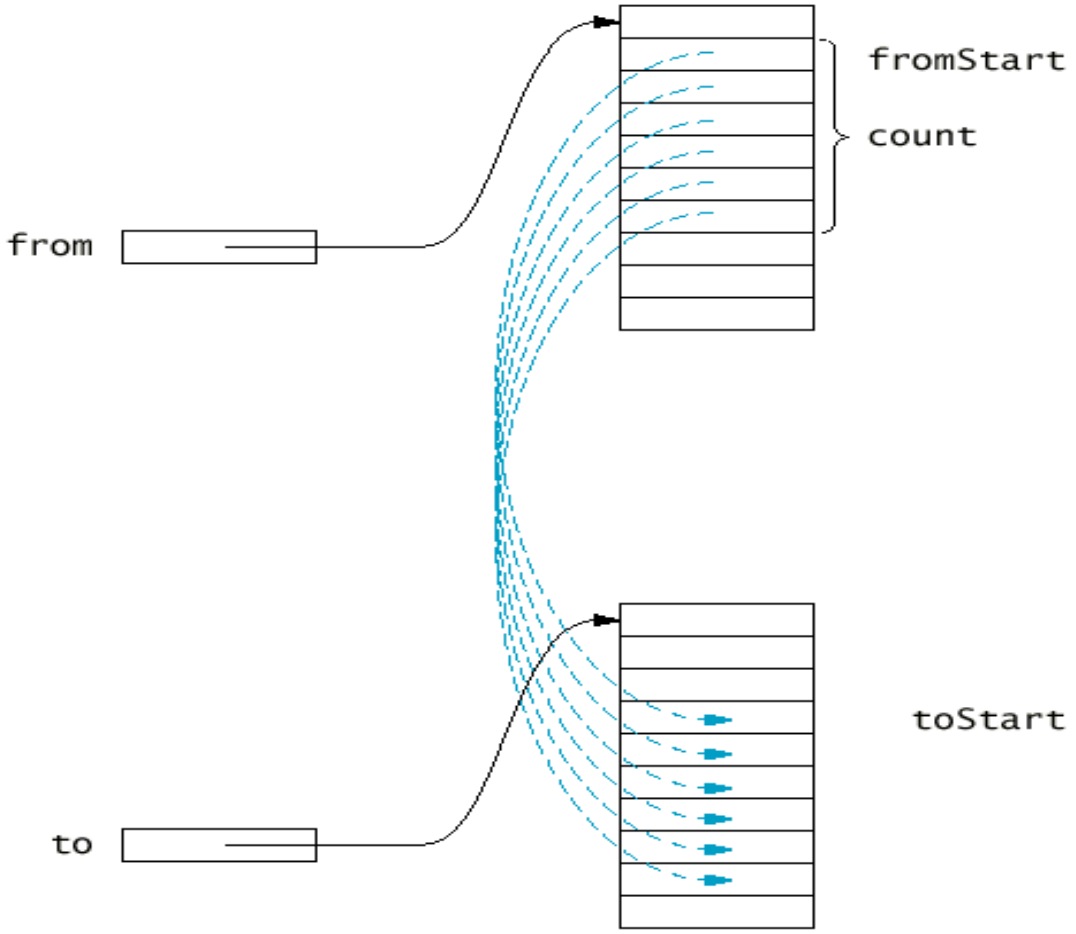
Per fare una vera copia occorre invocare il metodo clone

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



Copiare elementi da un array all'altro

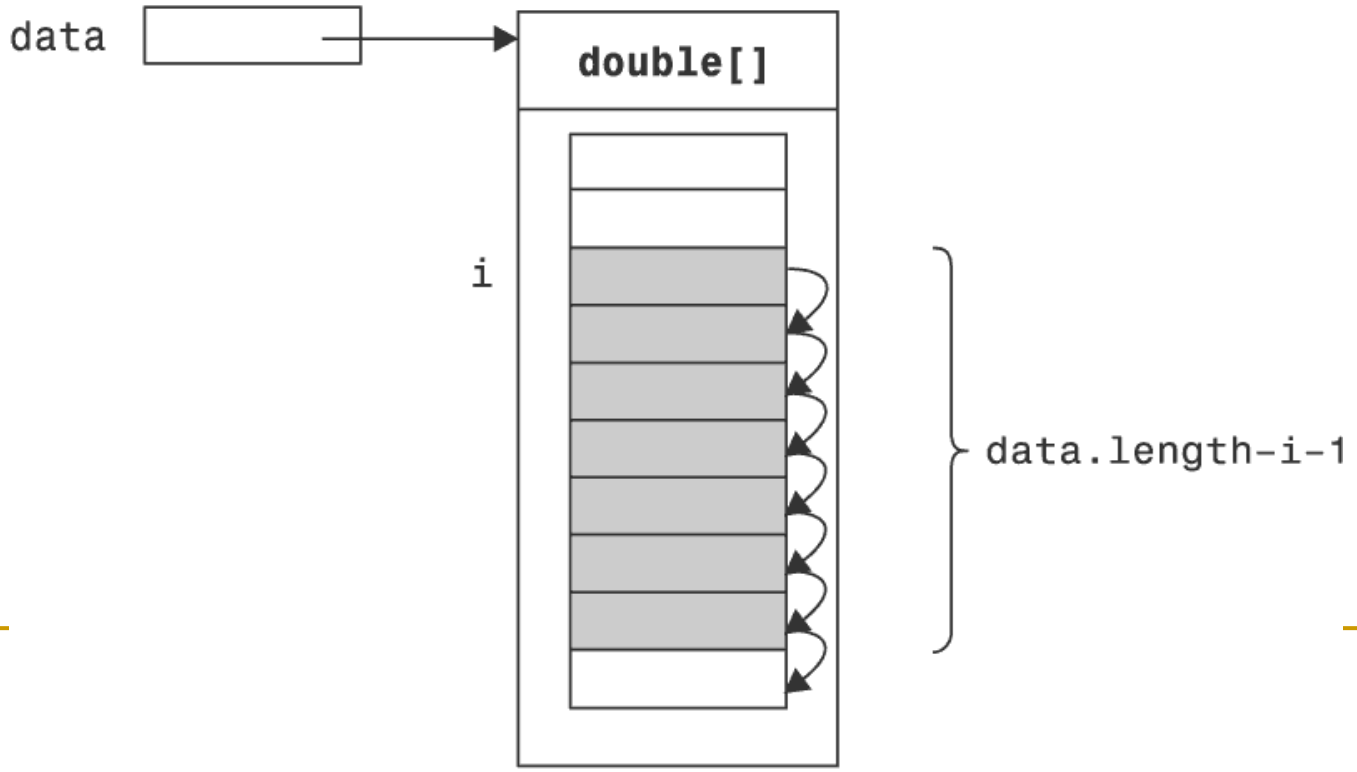
```
System.arraycopy(from, fromStart, to, toStart, count);
```



Uso di System.arraycopy

- Posso aggiungere un elemento in posizione i
  - Sposto di una posizione in avanti tutti gli elementi a partire da i

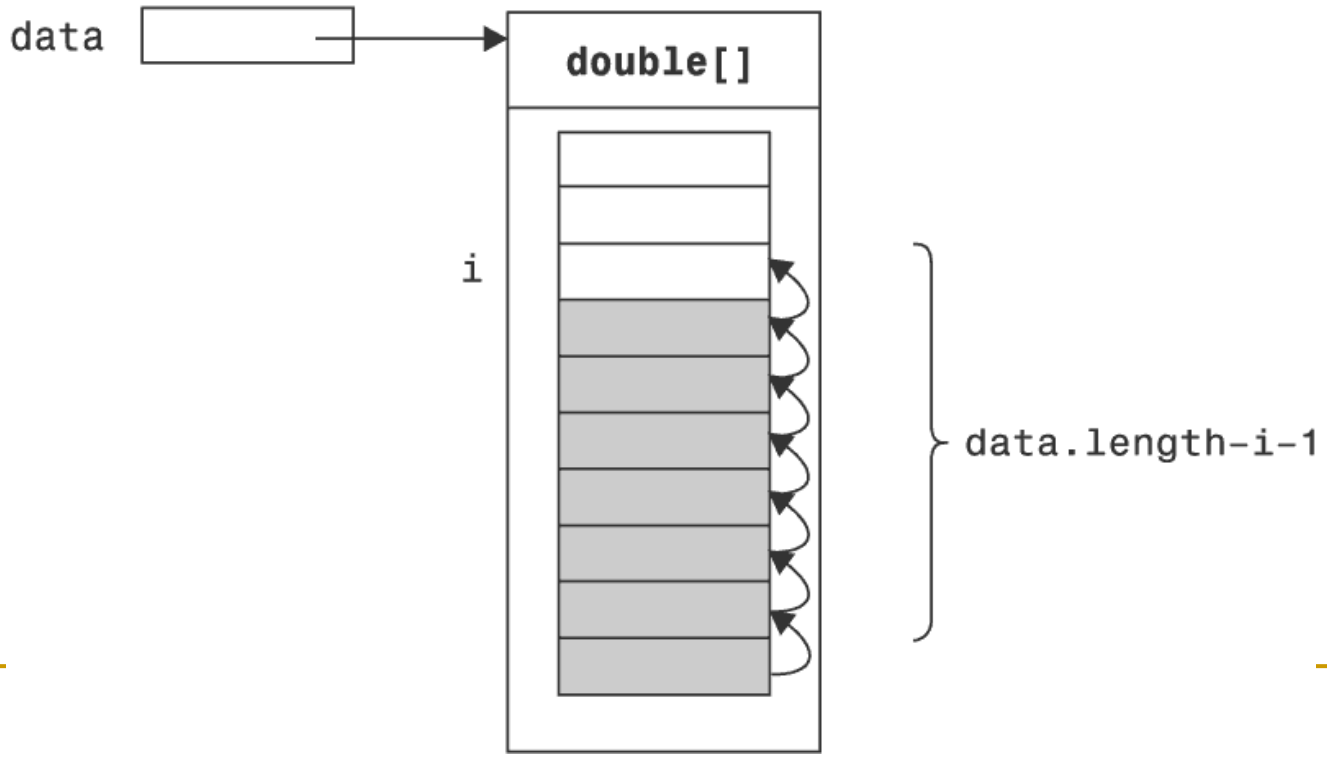
```
System.arraycopy(data, i, data, i+1, data.length-i-1);
data[i]=x;
```



Uso di System.arraycopy

- Posso eliminare un elemento in posizione i
  - Sposto di una posizione in indietro tutti gli elementi a partire da i+1

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



Array riempiti solo in parte

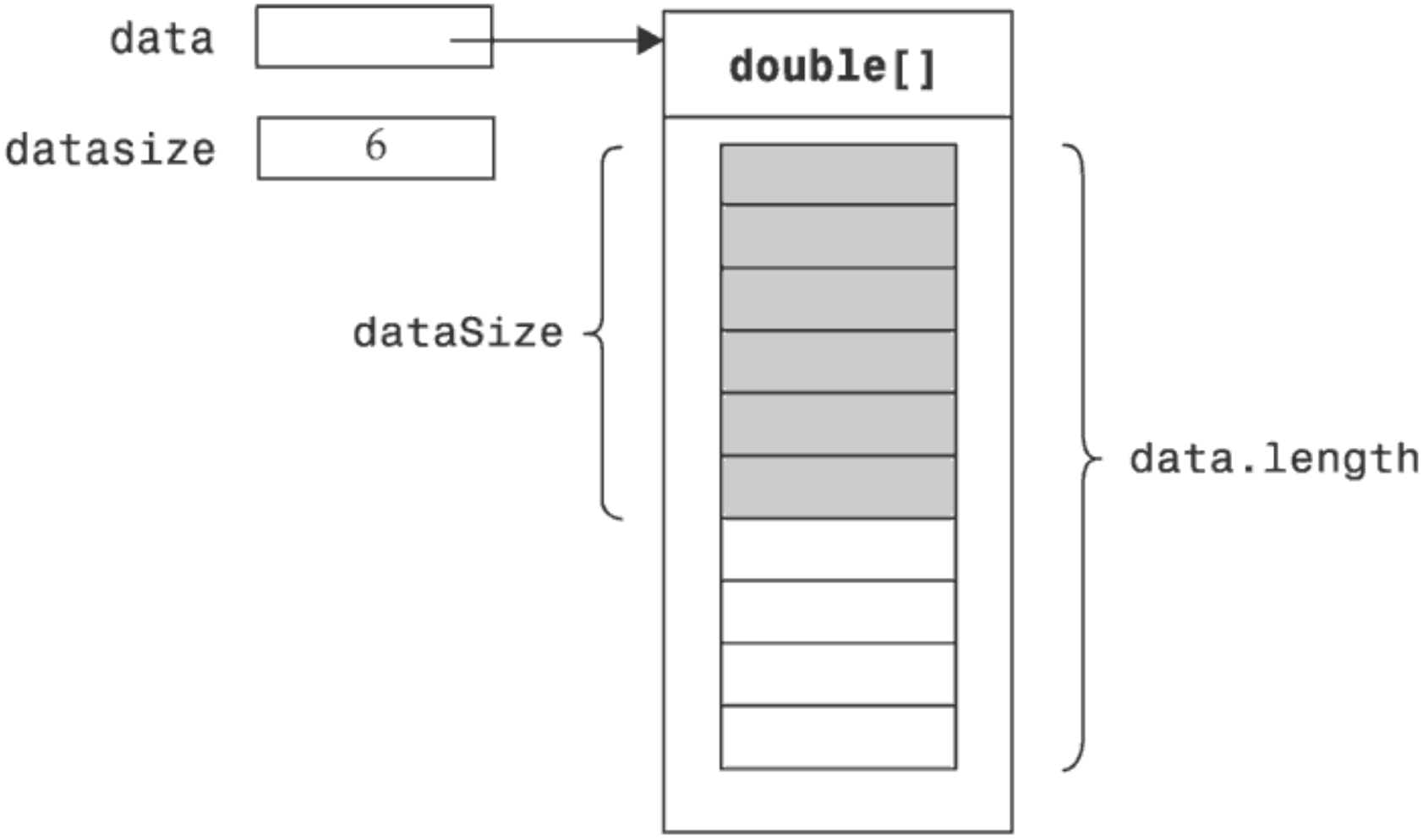
- Il max numero di elementi nell'array è prefissato
- Se non riempiamo tutto l'array dobbiamo tenere traccia del numero di elementi

```
final int DATA_LENGTH = 100;
double[] data = new double[DATA_LENGTH];
int dataSize = 0; //variabile complementare
//data.length è la capacità dell'array
//dataSize è la dimensione reale
```

- Se inseriamo elementi dobbiamo incrementare la dimensione

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

Array riempiti solo in parte



Array riempiti solo in parte

- In un ciclo, fermarsi a `dataSize` e non a `data.length`

```
for (int i = 0; i < dataSize; i++)
   sum = sum + data[i];
```

- Non riempire l'array oltre i suoi limiti

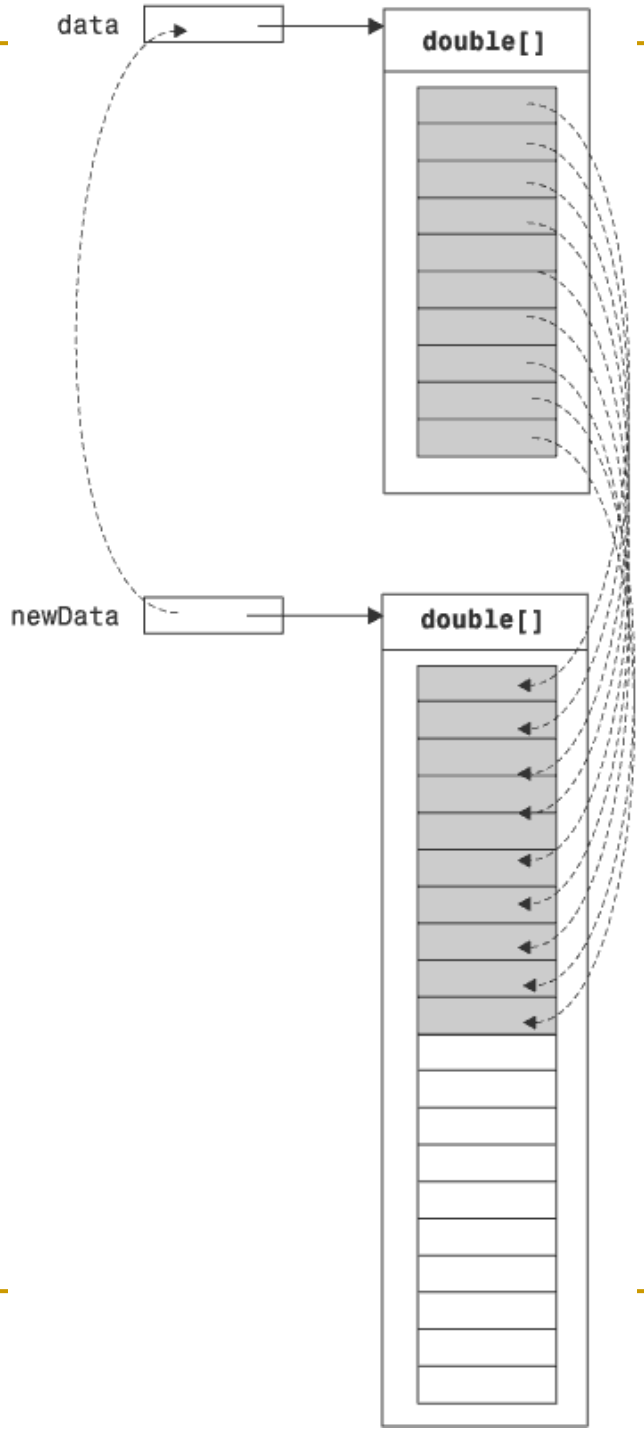
```
if (dataSize >= data.length)
   System.out.println("Mi dispiace, l'array è pieno");
```

- Oppure creare un nuovo array più grande, copiare gli elementi e assegnare il nuovo array alla variabile vecchia

```
double[] newData = new double[2 * data.length];
System.arraycopy(data, 0, newData, 0, data.length);
data = newData;
```

Esempio

```
class StringArray {
   private String[] stringhe;
   private int stringheSize;

   public StringArray () {
     stringhe = new String[100];
     stringheSize = 0;
   }
   public void addString (String s) {
     stringhe[stringheSize] = s;
     stringheSize++;
   }
   public String toString() {
     String s="";
     for (int i = 0; i < stringheSize; i++)
       s = s + stringhe[i];
     return s;
   }
}
```



Far crescere un array

File: ExtendibleTable.java (1)

```
public class ExtendibleTable{

  public ExtendibleTable(){
       data = new double[DATA_LENGTH];
       dataSize = 0;
  }

  public double get(int i) {
     if (i < 0 || i >= dataSize) throw new IndexOutOfBoundsException();
     return data[i];
  }


 public void set(int i, double x) {
     if (i < 0 || i >= dataSize) throw new IndexOutOfBoundsException();
    data[i] = x;
     }
```

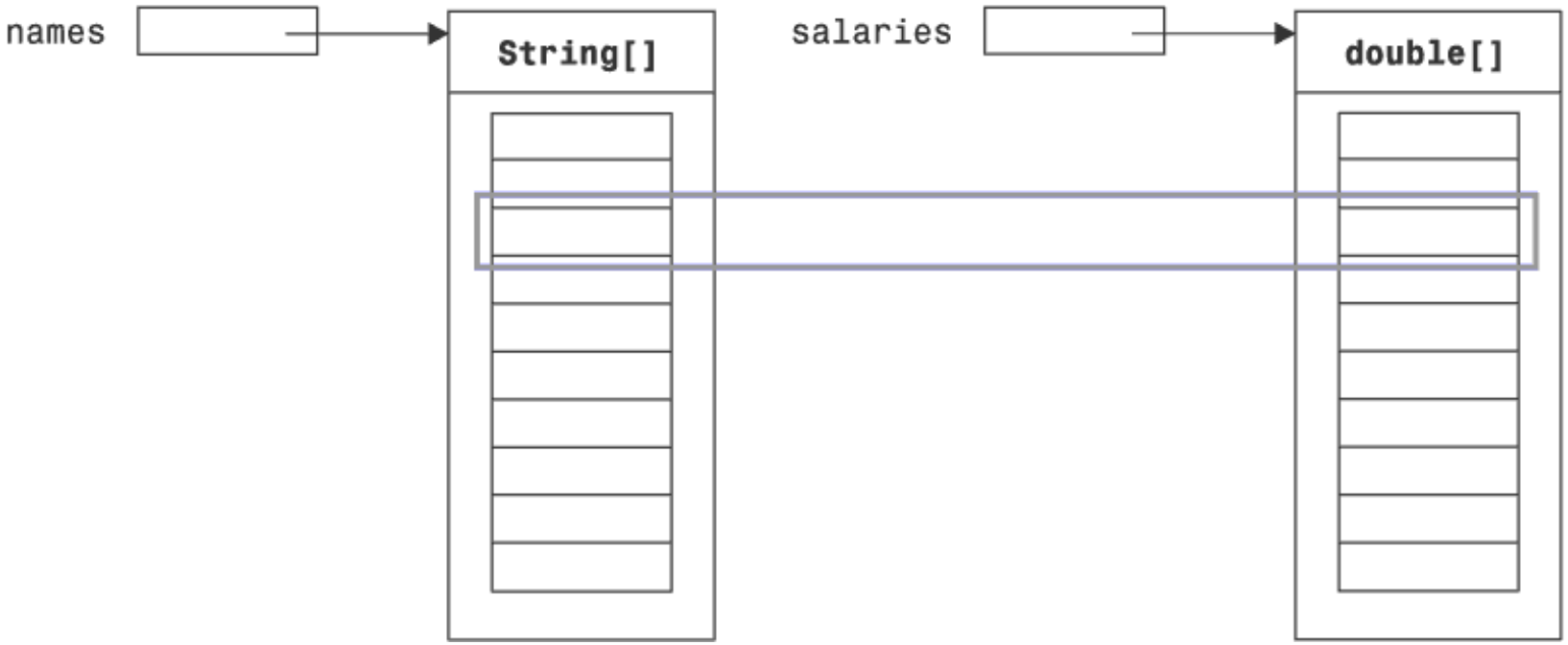
File: ExtendibleTable.java (2)

```
  public void add(double x){
     if (dataSize >= data.length){
          double[ ] newD = new double[2 * data.length];
          System.arraycopy(data, 0, newD, 0, data.length);
          data = newD;
     }
    data[dataSize] = x;
    dataSize++;
  }

  final static int DATA_LENGTH = 100;

  private double[ ] data;
  private int dataSize;
}
```

Array paralleli

- Non utilizzate array paralleli

```
String[] names;
double[] salaries;
```



Array di oggetti

- Riorganizzate i dati in array di oggetti

```
Employee[] employees;
```