# Corso di base JAVA

*Mauro Donadeo*
*mail: mauro.donadeo@gmail.com*

# Metodi e Variabili statiche, Scomposizione di stringhe, Array


HT Lab

# Metodi statici

Esistono classi che non servono a creare oggetti ma contengono **metodi statici** e **costanti**

- Queste si chiamano solitamente **classi di unità**
- La classe **Math** è un esempio di questo tipo di classi

# Metodi statici

Esistono classi che non servono a creare oggetti ma contengono **metodi statici** e **costanti**

- Queste si chiamano solitamente **classi di unità**
- La classe **Math** è un esempio di questo tipo di classi

```
1 public class Financial{
2     public static double percentOf(double p, double a){
3         return (p / 100) * a;
4     }
5 // qui si possono aggiungere altri metodi finanziari
6 }
```

Non è necessario **creare oggetti** di tipo **Financial** per usare i metodi della classe

**double tax = Financial.percentOf(taxRate,total);**

# Variabili statiche

Vogliamo modificare **BankAccount** in modo che:

- il suo stato contenga anche **numero di conto**

```
1 public class BankAccount
2 { ...
3     private int accountNumber;
4 }
```

- il numero di conto sia assegnato dal costruttore:
  - ogni conto deve avere un numero diverso
  - i numeri assegnati devono essere progressivi, iniziano da 1.

# Soluzione

## Prima idea

Usiamo una variabile per memorizzare l'ultimo numero di conto assegnato

```
1  public class BankAccount
2  { ...
3      private int accountNumber;
4      private int lastAssignedNumber;
5  ...
6      public BankAccount()
7      {
8          lastAssignedNumber++;
9          accountNumber = lastAssignedNumber;
10     }
11 }
```

## Il costruttore non funziona perché?

Questo costruttore non funziona perché **lastAssignedNumber** è una **variabile di esemplare**:

- ne esiste una copia per ogni oggetto;
- il risultato è che tutti i conti creati hanno un numero di conto uguale a 1

# Variabili statiche

Ci serve una **variabile condivisa da tutti gli oggetti della classe**

- una variabile con questa semantica si ottiene con la dichiarazione **static**

```
private static int lastAssignedNumber;
```

- Una vatiabile **static** **(variabile di classe)** è condivisa da tutti gli oggetti della classe;
- Ne esiste **un'unica copia** indipendentemente da quanti oggetti siano creati.

```
1  public class BankAccount{
2      ...
3      private int accountNumber;
4      private static int lastAssignedNumber = 0;
5      ...
6      public BankAccount(){
7          lastAssignedNumber++;
8          accountNumber = lastAssignedNumber;
9      }
10 }
```

Ogni metodo (o costruttore) di una classe può **accedere** alle variabili statiche della classe **modificarle**

- Le variabili statiche **non** possono (da un punto di vista logico) essere inizializzate nei costruttori:
  - Il loro valore verrebbe inizializzato nuovamente **ogni volta che si costruisce un oggetto**, perdendo il vantaggio di avere una variabile condivisa.
- Bisogna inizializzarle quando queste si dichiarano;
- Questo può valere anche per le variabili di esemplare, anziché usare un costruttore:
  - **non** è una buona pratica di programmazione.

È invece pratica comune (senza controindicazioni) usare **costanti** statiche, come la classe **Math**.

```
1 public class Math
2 { ...
3     public static final double PI
          =3.14159265358979323846;
4 }
```

Tali costanti sono di norma **public** e per ottenere il loro valore si usa il nome della classe seguito dal punto e dal nome della costante, **Math.PI**

Sappiamo che in Java esistono quattro diversi tipi di variabili:

- variabili **locali** (all'interno di un metodo)
- variabili **parametro** (dette **parametri formali**)
- variabili **di esemplare** (o di istanza)
- variabili **statiche** o di classe

Hanno in comune il fatto di contenere valori appartenenti ad un tipo ben preciso. Differiscono per quanto riguarda il loro **ciclo di vita**

- cioè nell'intervallo di tempo in cui, dopo essere state create, continuano ad occupare lo spazio in memoria riservato loro.

## Variabile locale

- **viene creata** quando viene eseguito l'enunciato in cui è definita;
- **viene eliminata** quando l'esecuzione del programma esce dal **blocco di enunciati** in cui la variabile è definita

## Variabile parametro (formale)

- **viene creata** quando viene invocato il metodo
- **viene eliminata** quando l'esecuzione del metodo termina

## Variabile statica

- **viene creata** quando la macchina virtuale Java carica la classe per la prima volta
- **viene eliminata** quando l'esecuzione del metodo termina
- a fini pratici possiamo dire che **esiste sempre**

## Variabile di esemplare

- **viene creata** quando viene creato l'oggetto a cui appartiene
- **viene eliminata** quando l'oggetto viene eliminato

- Per evitare conflitti, dobbiamo conoscere l'**amito di visibilità** di ogni tipo di variabile
  - Ovvero la **porzione** del programma all'interno della quale si può accedere ad essa;
- **Esempio:** due variabili locali con lo stesso nome. Funziona perché gli ambiti di visibilità sono **sono disgiunti**

```
1 public class RectangleTester{
2     public static double area(Rectangle rect){
3         double r = rect.getWidth() * rect.getHeight();
4         return r; }
5     public static void main(String[] args){
6         Rectangle r = new Rectangle(5, 10, 20, 30);
7         double a = area(r);
8         System.out.println(r); }
9 }
```

Anche qui gli ambiti di visibilità sono **disgiunti**

```
1 if (x >= 0){
2     double r = Math.sqrt(x);
3     ... } // la visibilita' di r termina qui
4 else{
5     Rectangle r = new Rectangle(5, 10, 20, 30);
6     // OK, questa e' un'altra variabile r
7 ...
8 }
```

Invece l'ambito di visibilità di una variabile **non** può contenere la definizione di un'altra variabile locale con lo stesso nome:

```
1 Rectangle r = new Rectangle(5, 10, 20, 30);
2 if (x >= 0)
3 {
4     double r = Math.sqrt(x);
5     // Errore: non si puo' dichiarare un'altra var. r
          qui
```

## Visibilità di membri di classe

- Membri **private** hanno visibilità di classe
  - Qualsiasi metodo di una classe può accedere a variabili e metodi della stessa classe
- Membri **public** hanno visibilità al di fuori della classe
  - A patto di renderne **qualificato** il nome, ovvero:
    - Specificare il nome della classe per membri static: **Math.PI, Math.sqrt(x)**
  - Specificare l'oggetto per i membri **non static**
- Non è necessario qualificare i membri appartenenti ad una stessa classe.

# Scomposizione di stringhe

Tutti gli esempi visti fino ad ora prevedevano l'inserimento dei **dati in ingresso uno per riga**, ma spesso è più comodo o più naturale per l'utente inserire **più dati per riga**

- ad esempio, cognome dello studente e voto

Dato che **nextLine** legge un'intera riga, bisogna imparare ad estrarre le sottostringhe relative ai singoli dati che compongono la riga

- non si può usare **substring**, perché in generale non sono note la lunghezza e la posizione dei singoli dati nella riga.

Per scomporre una stringa in token usando **Scanner**, innanzitutto bisogna creare un oggetto della classe fornendo **la stringa** come parametro al costruttore

```
Scanner in = new Scanner(System.in);
String line = in.nextLine();
Scanner t = new Scanner(line);
```

Successive invocazioni del metodo **next** restituiscono successive sottostringhe, fin quando l'invocazione di **hasNext** restituisce true

```
1 while (t.hasNext()){
2     String token = t.next();
3     // elabora token
4 }
```

# Esempio: contare le parole di un testo

```
1  import java.util.Scanner;
2  public class WordCounter{
3      public static void main(String[] args){
4          Scanner in = new Scanner(System.in);
5          int count = 0;
6          while (in.hasNextLine()){
7           String line = in.nextLine();
8           if (line.equals("Q"))
9  break;
10          Scanner t = new Scanner(line);
11              while (t.hasNext()){
12                  t.next(); // non devo elaborare
13                  count++;
14              }
15          }
16          System.out.println(count + " parole");
17      }
18 }
```

Scrivere un programma che:

- chiede all'utente di introdurre due stringhe (una per riga), s1 e s2; ciascuna stringa è costituita da tutti i caratteri presenti sulla riga, compresi eventuali spazi iniziali, finali e/o intermedi
- verificare se la stringa **s2**, è una sottostringa **s1**, cioè se esiste una coppia di numeri interi, x e y, per cui **s1** a partire da x fino a y contiene la stringa **s2**;

Il programma può usare, della classe **String**, i soli metodi **charAt** e **length**

- Un conto corrente possiede un numero di conto progressivo: ad un nuovo conto corrente viene assegnato il primo numero intero disponibile;
- I metodi deposit e withdraw restituiscono un valore di tipo logico, true se e solo se l'operazione è ammissibile e va a buon fine (ma non devono visualizzare nessun messaggio d'errore); se l'operazione non è ammissibile, il saldo del conto non deve essere modificato

Modificare la classe che effettua il test del conto bancario e accetta ripetutamente comandi dall'utente introdotti da tastiera, finché l'utente non introduce il comando di terminazione del programma:

| | |
|---|---|
| Q | Quit il programma termina; |
| B | Balance: visualizza il saldo del conto; |
| D x | Deposit versa nel conto la somma x; |
| W x | Withdraw: preleva dal conto la somma x; |
| A x | Add interest: accredita sul conto gli interessi, calcolati in base alla percentuale x del saldo attuale; |

# Array

## Problema

- **Scrivere un programma che legge dallo standard input una sequenza di dieci numeri in virgola mobile, uno per riga**
- chiedere all'utente un numero intero **index** e visualizzare il numero che nella sequenza occupava la posizione indicata da **index**.

- Occorre **memorizzare tutti i valori della sequenza**

## Problema

- **Scrivere un programma che legge dallo standard input una sequenza di dieci numeri in virgola mobile, uno per riga**
- chiedere all'utente un numero intero **index** e visualizzare il numero che nella sequenza occupava la posizione indicata da **index**.

- Occorre **memorizzare tutti i valori della sequenza**
- Potremmo usare dici variabili diverse per memorizzare i valori, selezionati poi con una lunga sequenza di alternative, **ma se i valori dovessero essere mille?**

# Memorizzare una serie di valori

Lo strumento messo a disposizione dal linguaggio Java per memorizzare una sequenza di dati si chiama **array** (che significa "sequenza ordinata")

- La struttura **array** esiste in quasi tutti i linguaggi di programmazione

Un array in Java è **un oggetto** che realizza una **raccolta di dati che siano tutti dello stesso tipo**.
Potremo avere quindi un array di numeri interi, array di numeri in virgola mobile, array di stringhe, array di conti bancari.

## Costruire un array

Come ogni **oggetto**, un array deve essere **costruito** con l'operatore **new**, dichiarando il **tipo di dati** che potrà contenere.

```
new double [10]
```

Il tipo di dati di un array può essere qualsiasi tipo di dati valido in Java

- uno dei tipi di dati fondamentali o una classe

e nella costruzione deve essere seguito da una **coppia di parentesi quadre** che contiene la **dimensione** dell'array, cioè il numero di elementi che potrà contenere.

# Riferimento ad un array

Come succede con la costruzione di ogni oggetto, l'operatore **new** restituisce un **riferimento** all'array appena creato, che può essere memorizzato in una **variabile oggetto** dello stesso tipo.

**double**[] **values** = **new** **double**[10]

## Attenzione

Nella definizione della variabile oggetto devono essere presenti le parentesi quadre, ma non deve essere indicata la dimensione dell'array; la variabile potrà riferirsi solo ad array di quel tipo, mi di qualunque dimensione.

# Utilizzare un array

Al momento della costruzione, tutti gli elementi dell'array vengono inizializzati ad un valore, seguendo **le stesse regole viste per le variabili esemplare**

- Per accedere ad un elemento dell'array si usa:

```
double[] values = new double[10];
double oneValue = values[3];
```

- La stessa sintassi si usa per **modificare** un elemento dell'array

```
double[] values = new double[10];
values[5] = 3.4;
```

```
1 double[] values = new double[10];
2 double oneValue = values[3];
3 values[5] = 3.4;
```

- Il numero utilizzato per accedere ad un particolare elemento dell'array si chiama **indice**
- L'indice può assumere un valore compreso tra **0 (incluso)** e la **dimensione** dell'array (**esclusa**), cioè segue le stesse convenzioni viste per le posizioni dei caratteri di una stringa:
  - il primo elemento ha indice zero
  - l'ultimo elemento ha indice **dimensione - 1**

Un array è un oggetto un pò strano: **non ha metodi pubblici, né statici né di esemplare**

- L'unico elemento pubblico di un oggetto di tipo array è la sua dimensione a cui si accede attraverso la sua variabile pubblica di esemplare **length** (attenzione, non è un metodo).

```
double[] values = new double[10]
int a = values.length;
```

# Esercitazione

- Scrivere un programma che legge dallo standard input una sequenza di dieci numeri in virgola mobile, uno per riga
- chiedere all'utente un numero intero **index** e visualizzare il numero che nella sequenza occupava la posizione indicata da **index**

# Inizializzazione di un array

Quando si assegnano i valori agli elementi di un array si può procedere così:

```
1 int[] primes = new int[3];
2 primes[0] = 2;
3 primes[1] = 3;
4 primes[2] = 5;
```

ma se si conoscono tutti gli elementi da inserire si può usare questa sintassi (**migliore**)

**int[] primes = {2, 3, 5};**

oppure (**accettabile ma meno chiara**)

**int[] primes = new int[]{2, 4, 5};**

# Passare un array come parametro

**S**

pesso si scrivono metodi che ricevono array come parametri espliciti

```
1  public static double sum(double[] values){
2      if (values == null){
3          System.out.println("errore elemento non
               inizializzato");
4          return null;
5      }
6      if (values.length == 0)
7          return 0;
8      double sum = 0;
9      for (int i = 0; i < values.length; i++)
10         sum = sum + values[i];
11     return sum;
12 }
```

## Un metodo può anche usare un array come **valore di ritorno**

```
1  public static int[] resize(int[] oldArray, int newLength
       ){
2      if (newLength < 0 || oldArray == null){
3          System.out.println("parametri non corretti");
4          return null;
5      }
6      int[] newArray = new int[newLength];
7      int count = oldArray.length;
8      if (newLength < count)
9          count = newLength;
10     for (int i = 0; i < count; i++)
11         newArray[i] = oldArray[i];
12     return newArray;
13 }
14 ...
15 int[] values = {1, 7, 4};
16 values = resize(values, 5);
17 values[4] = 9;
```

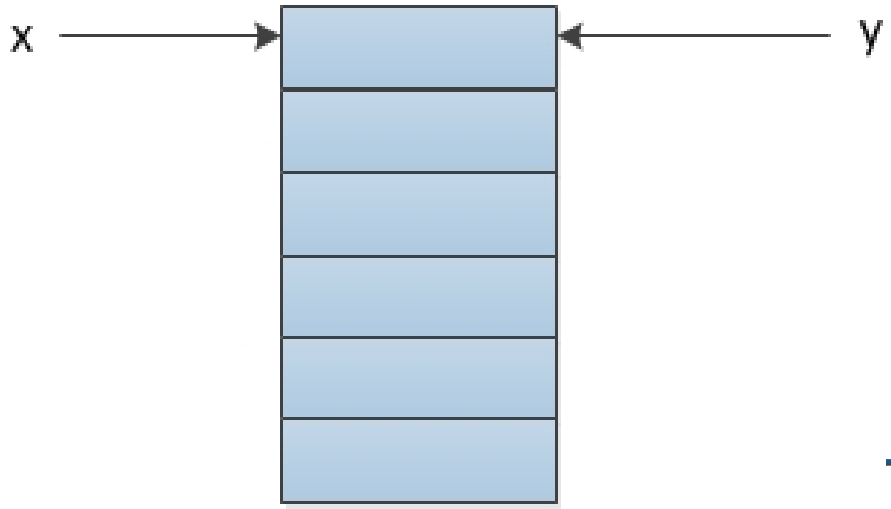
# Copiare un array

Ricordando che una variabile che si riferisce ad un array è una variabile oggetto

- contiene un riferimento all'oggetto

Copiando il contenuto della variabile in un'altra **non si copia l'array** ma si ottiene un altro riferimento allo **stesso oggetto array**

```
1 double[] x = new double[6];
2 double[] y = x;
```



Se si vuole ottenere **una copia dell'array**, bisogna:

- **creare un nuovo array dello stesso tipo e con la stessa dimensione**
- **copiare ogni elemento del primo array nel corrispondente elemento del secondo array**

```
1 double[] values = new double[10];
2 // inseriamo i dati nell'array
3 ...
4 double[] otherValues = new double[values.length];
5 for (int i = 0; i < values.length; i++)
6     otherValues[i] = values[i];
```

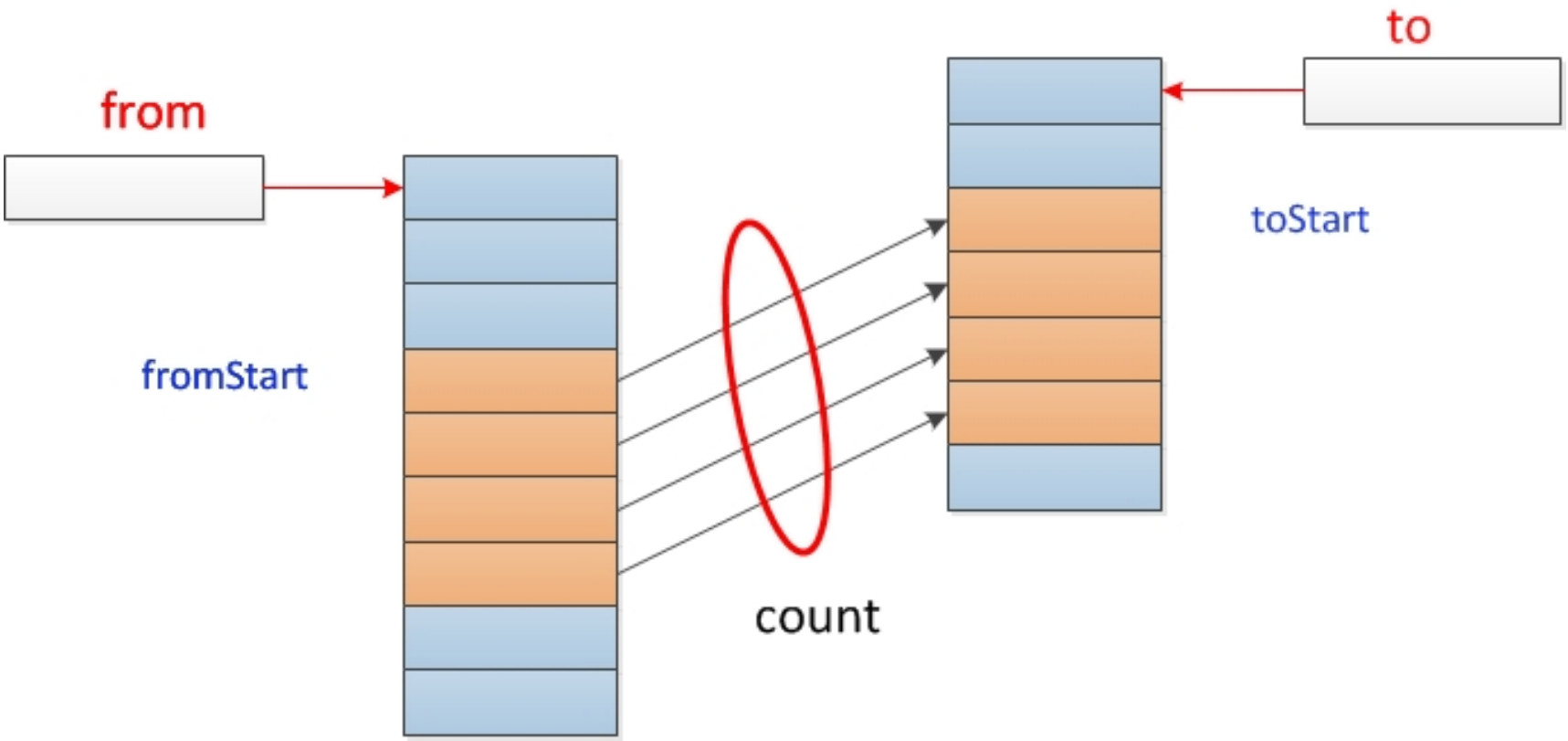
Invece di usare un ciclo è possibile (e **più efficiente**) invocare il metodo statico **arraycopy** della classe **System**

```
System.arraycopy(values,0,otherValues,0,values.legnth);
```

Il metodo **System.arraycopy** consente di copiare una porzione di un array in un altro array

# System.arraycopy

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



# Esercitazione

Il crivello di Eratostene è un noto algoritmo per la ricerca dei numeri primi minori di un certo valore massimo MAX, ed è così specificato

- i predispone un array di MAX valori booleani ogni elemento dell'array "rappresenta" il numero intero corrispondente al proprio indice nell'array
- se e solo se l'elemento è true, allora il numero corrispondente è stato eliminato dall'insieme dei numeri primi, cioè non è un numero primo
- all'inizio si suppone che tutti i numeri siano primi; successivamente si considera ciascun numero intero maggiore di uno, in ordine crescente, e si eliminano tutti i numeri che ne sono multipli, contrassegnando opportunamente l'array.
- al termine, i numeri rimasti sono tutti e soli i numeri primi cercati, non essendo multipli di alcun numero.

Scrivere un programma che realizza il Crivello di Eratostene per identificare i numeri primi minori di un valore (intero positivo) MAX fornito dall'utente attraverso l'ingresso standard.