

# Introduzione a Java (seconda parte)

## Classi “wrapper” (involucro)

---

Si trovano in `java.lang` e sono usate per guardare a elementi di un tipo primitivo come se fossero oggetti.

<u>Tipo Primitivo</u>	<u>Wrapper class</u>
<code>boolean</code>	<code>Boolean</code>
<code>byte</code>	<code>Byte</code>
<code>char</code>	<code>Character</code>
<code>short</code>	<code>Short</code>
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>float</code>	<code>Float</code>
<code>double</code>	<code>Double</code>

Un oggetto di questo tipo può essere costruito passando il valore al rispettivo costruttore, ad es.

```
int intero = 100;  
Integer wintero = new Integer(intero);
```

Ogni wrapper class mette a disposizione metodi e costanti, es:

```
int j = wintero.intValue();  
int i = Integer.parseInt("110");  
int min = Integer.MIN_VALUE;  
int max = Integer.MAX_VALUE;
```

## Input / Output

---

Come in C e C++ l'input/output è supportato da una libreria e non dal linguaggio. La libreria si chiama **java.io**

Vediamo come effettuare input / output in modo semplice, attraverso la Java console window (cioè la finestra terminale di Linux)

Stream = sequenza/flusso di bytes in input o output

Quando si esegue un'applicazione Java vengono automaticamente creati tre oggetti stream (della classe `java.lang.System`):

**System.in** (input di byte da tastiera)  
`int read()` della classe `InputStream` legge un byte alla volta e ritorna -1 alla fine dello stream di input

**System.out** (output a video)  
`print()`, `println()` della classe `PrintStream` permettono di visualizzare dati di tipo primitivo, `String` e `Object`

**System.err** (messaggi di errore a video)

## Un esempio di lettura da input

---

```
public class ContaCaratteri {
    public static void main(String[] args) throws java.io.IOException {
        int num = 0;

        // leggo lo stream di input fino alla fine dello
        // stream (si termina con doppio Ctrl d)
        while (System.in.read() != -1){
            num++;
        }
        System.out.println("\n Input di " + num + " caratteri.");
    }
}
```

non gestiamo eventuali  
errori in lettura


va a capo!

leggere un solo byte alla volta non è molto comodo...

## lettura da input con BufferedReader

---

```
public class EsempioLettura {
    public static void main(String[] args) throws java.io.IOException {
        int a;
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        System.out.print("Inserisci il valore: ");
        a = Integer.parseInt(in.readLine());
    }
}
```



- La classe `InputStreamReader` di `java.io` consente di leggere un carattere (e non un singolo byte!) alla volta
- La classe `BufferedReader` di `java.io` permette di effettuare la lettura di caratteri da uno stream di input in modo bufferizzato (e quindi efficiente)
- Il metodo `readLine()` della classe `BufferedReader` legge una sequenza di caratteri terminata con newline

## La classe Math

---

La classe `Math` si trova all'interno del package `java.lang` e definisce costanti matematiche e metodi per le più comuni funzioni matematiche. Ad Esempio:

**Costanti:**

```
public static final double E = 2.7182818284590452354;
public static final double PI = 3.14159265358979323846;
```

**Metodi**

```
public static double abs(double x)
public static long round(double x)
public static double sqrt(double x)
public static double min(double x, double y)
public static double max(double x, double y)
public static double random()
...
```

Tutti i metodi sono *static*, cioè sono associati alla classe e non alle singole istanze. I metodi statici si richiamano premettendo sempre il nome della classe di appartenenza, es: `Math.sqrt(3.455)`

## Metodi Statici

---

Abbiamo visto che per usare un metodo serve sempre un'istanza della classe di appartenenza. Ad esempio:

```
Rettangolo r = new Rettangolo(1,1);  
If (r.quadrato())  
    System.out.println("è quadrato");
```

Questo va bene quando il metodo "opera" sulle variabili relative all'istanza chiamante, ovvero sullo stato dell'istanza stessa.

Ci sono però dei casi in cui questo è una limitazione. Ad esempio il metodo

```
fattoriale(n)
```

che restituisce il fattoriale del numero naturale n è di fatto una funzione: prende in input n e produce in output il suo fattoriale.

## Metodi Statici

---

Inoltre, qualsiasi sia la classe a cui appartiene il metodo `fattoriale(n)`, esso non dipende dallo stato dell'istanza che lo chiama.

In questo caso dover creare un'istanza è inutile e artificioso.

Per i casi come questo Java prevede la possibilità di dichiarare metodi **statici**, che possono essere chiamati premettendo semplicemente il nome della classe, senza che occorra creare un'istanza. Ad esempio:

```
Integer.parseInt("100");  
Math.max(1,2);
```

...

I metodi statici sono comunque inseriti in una classe ma sono indipendenti dalle istanze della classe stessa: per questo motivo si dice che sono associati alla classe e non alle sue istanze.

Infine, i metodi statici hanno una limitazione rispetto ai metodi non statici: **non possono** (ovviamente!) **accedere ai campi non statici della classe**.

## Main è un metodo statico

---

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

Il metodo di avvio di Java è statico. L'interprete Java carica prima di tutto la classe specificata, senza crearne alcuna istanza. Per questo motivo l'esecuzione parte da un metodo statico, che deve chiamarsi main, non deve ritornare nulla, e prendere un array di stringhe come argomento.

## Esempio: la classe Point (1)

---

```
public class Point {
    public static final Point ORIGIN = new Point();
    private double x;    // ascissa
    private double y;    // ordinata

    // post: costruisce un punto che rappresenta l'origine
    public Point() {
        this.x = this.y = 0;
    }

    // post: costruisce un punto con le coordinate passate
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // post: ritorna l'ascissa
    public double getX() {return x;}

    // post: ritorna l'ordinata
    public double getY(){return y;}
}
```

**campi privati, accessibili solo dalla classe**

## Esempio: la classe Point (2)

---

```
// post: imposta l'ascissa secondo il parametro
public void setX(double x) {
    if (this != ORIGIN)
        this.x = x;
}

// post: imposta l'ordinata secondo il parametro
public void setY(double y) {
    if (this != ORIGIN)
        this.y = y;
}

// post: ritorna true se "questo" punto e' uguale a p
public boolean equals(Point p) {
    if (this == p)
        return true;
    else
        return (x == p.x && y == p.y);
}

// post: ritorna una stringa che rappresenta il punto
public String toString() {
    return new String("(" + x + "," + y + ")");
}
```

## Esempio: la classe Point (3)

---

```
// post: sposta il punto sul piano cartesiano secondo i parametri
public void translate(double dx, double dy) {
    x += dx;
    y += dy;
}

// post: restituisce la distanza tra questo punto e il parametro
public double distance(Point p) {
    double dx = this.x - p.x;
    double dy = this.y - p.y;
    return Math.sqrt(dx*dx + dy*dy);
}

// post: restituisce la distanza tra questo punto e l'origine
public double distance() {
    return Math.sqrt(x*x + y*y);
}
}
```

## Stringhe

---

La classe `String` (di `java.lang`) viene fornita da Java per operare sulle stringhe

Es: `System.out.println("Ciao a tutti");` crea un nuovo oggetto di tipo `String`, lo inizializza al valore specificato e passa il riferimento al metodo `println`

A differenza degli array, non serve specificare la lunghezza della stringa al momento della creazione, e non serve usare l'operatore `new`. Es: `String pippo = "Laboratorio";`

L'operatore di concatenazione di stringhe è `+`

Es: `pippo = pippo + " ASD " + "e Programmazione"`

Il metodo `length()` ritorna la lunghezza della stringa. I caratteri sono indicati da zero a `length() - 1`.

Es: `int n = pippo.length();`

Il metodo `charAt()` ritorna il carattere alla posizione specificata

Es: `pippo.charAt(1)` ritorna il carattere 'a'

## Stringhe

---

Gli oggetti stringhe sono read-only e non modificabili. Ogni volta che si eseguono operazioni che sembrano modificare un'oggetto di tipo stringa, in realtà si tratta della costruzione di un nuovo oggetto

Casting con oggetti stringa: quando si concatena un oggetto stringa con un tipo primitivo o un qualsiasi altro oggetto, esso viene automaticamente convertito in stringa

Il casting esplicito di un oggetto o di un tipo primitivo al tipo stringa non è consentito

```
String s = "la temperatura è di gradi " + 25;           \\ SI
String t = "la temperatura è di gradi " + (String)25;  \\ NO
String z = 25;                                         \\ NO
String w = (String)25;                                  \\ NO
String y = Integer.toString(25);                       \\ SI
```

## Stringhe: ricerca occorrenze

Alcuni metodi per trovare occorrenze di particolari caratteri o sottostringhe in una stringa:

Metodo	Ritorna l'indice di...
<code>indexOf(char ch)</code>	prima posizione di <code>ch</code>
<code>indexOf(char ch, int start)</code>	prima posizione di <code>ch</code> $\geq$ <code>start</code>
<code>indexOf(String str)</code>	prima posizione di <code>str</code>
<code>indexOf(String str, int start)</code>	prima posizione di <code>str</code> $\geq$ <code>start</code>
<code>lastIndexOf(char ch)</code>	ultima posizione di <code>ch</code>
<code>lastIndexOf(char ch, int start)</code>	ultima posizione di <code>ch</code> $\leq$ <code>start</code>
<code>lastIndexOf(String str)</code>	ultima posizione di <code>str</code>
<code>lastIndexOf(String str, int start)</code>	ultima posizione di <code>str</code> $\leq$ <code>start</code>

**Es:** `String pippo = "blablabla"`  
`pippo.indexOf('a')` ritorna il valore 2  
`pippo.lastIndexOf("bla")` ritorna il valore 6

## Stringhe: metodi di utilità

Alcuni metodi di utilità per le stringhe sono:

`public String replace(char oldChar, char newChar)`  
ritorna una nuova stringa dove tutte le istanze di `oldChar` sono rimpiazzate con il carattere `newChar`

`public String toLowerCase()`  
ritorna una nuova stringa dove gli eventuali caratteri in maiuscolo sono rimpiazzati con i relativi caratteri in minuscolo

`public String toUpperCase()`  
ritorna una nuova stringa dove gli eventuali caratteri in minuscolo sono rimpiazzati con i relativi caratteri in maiuscolo

`public String trim()`  
ritorna una nuova stringa dove gli eventuali spazi iniziali e finali sono rimossi

**Es:** `String pippo = "abracadabra";`  
`String pippo = pippo.replace("a", "b");` ritorna "abracadabra"

## Confronti tra stringhe

---

**Attenzione: non è possibile confrontare direttamente due oggetti con "=="..** Infatti l'operatore "==" confronta le referenze degli oggetti in esame e non i loro contenuti.

La classe String mette a disposizione due metodi per il confronto:

`public boolean equals(String str):` ritorna true se la stringa corrispondente alla referenza corrente ha lo stesso contenuto della stringa corrispondente alla referenza `str`

Es: `pippo.equals("beta")`

`public int compareTo(String str) :` ritorna zero se la stringa corrente è uguale alla stringa corrispondente a `str`; ritorna un numero negativo (risp. positivo) se la stringa corrente è minore (risp. maggiore) alla stringa corrispondente a `str` rispetto all'ordinamento lessicografico. Ad esempio:

```
String pippo = "aaa";
```

```
pippo.compareTo("zzz") ritorna un numero negativo
```

```
pippo.compareTo("a") ritorna un numero positivo
```

## Array

---

- Oggetti che raggruppano dati dello stesso tipo
- Si possono dichiarare array i cui elementi sono di tipo qualsiasi (primitivo o riferimento)

```
char s[];  
char [] s; //forma alternativa
```

```
Data giorni[];  
Data [] giorni; //forma alternativa
```

- La dichiarazione alloca memoria solo per il riferimento
- Per creare un array bisogna usare l'operatore `new`

```
s = new char[20];  
giorni = new Data[2];  
giorni[0] = new Data();  
giorni[1] = new Data();
```

## Inizializzazione di un array

---

Quando si crea un array, tutti gli elementi sono inizializzati automaticamente ai valori di default (`null` nel caso di tipi riferimento)

Per fare un'inizializzazione esplicita:

```
String nomi[] = { // inizializza un array di 3 stringhe
    "Chiara",
    "Paola",
    "Silvia",
};

Data giorni[] = { // inizializza un array di due
    new Data(), // oggetti di tipo Data
    new Data(),
};
```

## Array multidimensionali

---

### Tablette di tabelle

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

L'oggetto creato dalla prima chiamata di `new` è un array che contiene 2 elementi, ciascuno di tipo array di interi

### Tablette rettangolari

```
int rett [][]= new int[2][3];
rett[0] = new int[3];
rett[1] = new int[3];
```

L'oggetto creato dalla prima chiamata di `new` è un array che contiene 2 elementi, ciascuno di tipo array di 3 interi

## Come copiare il contenuto di un array

---

*Un array non è ridimensionabile!*

Si può usare la stessa variabile di tipo array per un riferirsi ad un array completamente diverso

```
int elementi[] = new int[6];
elementi = new int[10];
```

Per copiare il contenuto di un array in un altro, si usa il metodo `System.arraycopy()`

```
int piccolo[] = { 1,2,3,4,5,6 };
int grande[]  = { 11,12,13,14,15,16,17,18,19,20 };
System.arraycopy( piccolo,0,grande,0,piccolo.length );
```

a questo punto l'array grande ha il seguente contenuto:

```
{ 1, 2, 3, 4, 5, 6, 17, 18, 19, 20 }
```

## Gli array sono oggetti...

---

Gli oggetti array provvedono il campo **length**, che è una costante il cui tipo è `int` e che contiene il numero di elementi dell'array

```
class Test {
    public static void main(String[] args) {
        int[] a = new int[3];
        System.out.println("num elementi = " + a.length);
    }
}
```

produce in output

```
num elementi = 3
```

## La classe Vector

---

La classe `Vector` si trova all'interno del package `java.util`

- Un vettore è una struttura dati costituita da un blocco di memoria contiguo
- Lo spazio di memorizzazione viene gestito automaticamente in modo che, in seguito ad un tentativo di inserire un elemento in un vettore pieno, venga assegnato un blocco di memoria più grande, gli elementi del vettore vengano copiati nel nuovo blocco, ed il vecchio blocco venga abbandonato
- Il vettore è quindi un array più flessibile, cioè un array la cui dimensione può aumentare o diminuire dinamicamente

Trovate la specifica della classe `Vector` sulle API di Java. Alcuni metodi sono:

```
public void add(int index, object element)
public void remove(int index)
public boolean contains(Object elem)
public int indexOf(Object elem)
```

Una classe simile è `ArrayList` (sempre parte di `java.util`)