

# Fondamenti di Informatica

## Java

SUPSI Dipartimento Tecnologie Innovative

Gianni Grasso

27 novembre 2024

**Classe:** I1B

**Anno scolastico:** 2024/2025

# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Compilazione ed esecuzione . . . . .	4
1.2	Alcune definizioni . . . . .	4
1.3	Identificatori . . . . .	4
<b>2</b>	<b>Tipi di dato</b>	<b>5</b>
2.1	boolean . . . . .	5
2.2	Tipi numerici . . . . .	5
2.2.1	Interi . . . . .	5
2.2.2	char . . . . .	5
2.2.3	Decimali . . . . .	5
2.3	Operatori logici . . . . .	6
2.3.1	Algebra di Boole . . . . .	6
2.4	Operatori aritmetici . . . . .	7
2.4.1	Operatori ++ e -- . . . . .	7
2.5	Letterali . . . . .	7
2.6	Conversioni . . . . .	7
<b>3</b>	<b>Introduzione alle classi</b>	<b>8</b>
3.1	La classe <b>Scanner</b> . . . . .	8
3.1.1	Anomalia dell'input . . . . .	8
3.2	Stringhe . . . . .	9
3.2.1	Confronti tra stringhe . . . . .	9
3.2.2	Conversione di tipi primitivi . . . . .	9
3.2.3	Operazioni sulle stringhe . . . . .	9
3.3	La classe <b>Math</b> . . . . .	10
3.4	Generare numeri casuali . . . . .	10
<b>4</b>	<b>Istruzioni di controllo</b>	<b>11</b>
4.1	Switch . . . . .	11
4.2	Operatore ternario . . . . .	12
4.3	Ciclo <b>for</b> . . . . .	12
4.4	L'istruzione <b>continue</b> e <b>break</b> . . . . .	12
4.5	L'istruzione <b>do while</b> . . . . .	12
<b>5</b>	<b>Array</b>	<b>13</b>
5.1	Sintassi . . . . .	13
5.1.1	Creazione . . . . .	13
5.1.2	Utilizzare gli elementi . . . . .	13
5.1.3	Proprietà . . . . .	13
5.2	<b>foreach</b> . . . . .	14
5.3	Confronto . . . . .	14
5.4	Trovare massimo/minimo . . . . .	14
5.5	Aumentare la lunghezza . . . . .	14
5.6	Gestione della memoria . . . . .	15
5.7	Copiare un array . . . . .	15
5.7.1	Shallow copy . . . . .	15
5.7.2	Deep copy . . . . .	15
5.8	Array multidimensionali . . . . .	16
<b>6</b>	<b>Enumerativi</b>	<b>17</b>
6.1	Operazioni . . . . .	17
6.2	Enum e switch . . . . .	17
<b>7</b>	<b>Sottoprogrammi</b>	<b>19</b>
7.1	Gestione della memoria . . . . .	19
7.1.1	Tipi primitivi . . . . .	19

7.1.2	Tipi riferimento . . . . .	20
7.1.3	Stringhe . . . . .	20
<b>8</b>	<b>Ricorsione</b>	<b>21</b>
8.0.1	Alcune definizioni . . . . .	21
8.0.2	Approccio . . . . .	21
<b>9</b>	<b>Oggetti</b>	<b>22</b>
9.1	Costruttori . . . . .	22

# 1 Introduzione

Java è un linguaggio di programmazione **orientato agli oggetti** progettato per essere il più possibile indipendente dalla piattaforma di esecuzione.

## 1.1 Compilazione ed esecuzione

- **JRE**, Java Runtime Environment, è un ambiente che permette di eseguire applicazioni java, contiene la *Java Virtual Machine (JVM)* e le Java APIs.
- **JDK**, Java Development Kit, una collezione di tools per la programmazione in Java: compilatore, debugger, eccetera. Esso comprende una versione della JRE al suo interno

Nel codice sorgente ogni programma Java è rappresentato da **una classe**. Il nome del file **deve** coincidere con quello della classe.

Ecco la struttura di base di un programma Java:

```
/**
 * Esempio di programma.
 */
public class NomeClasse {
    public static void main(String[] args) {
        //codice del programma
    }
}
```

## 1.2 Alcune definizioni

- **Letterali**, un letterale è la rappresentazione di un qualsiasi valore di tipo primitivo, stringa o null, in altre parole tutto ciò che potrebbe essere assegnato ad una variabile
- **Espressioni**, un'espressione è un costrutto (porzione di codice) che quando viene elaborato assume un singolo valore. Essa può essere composta da variabili, operatori, letterali, separatori e invocazioni di funzioni. Il suo tipo di dato dipende dagli elementi che la compongono
- **Istruzioni**, l'istruzione è l'unità di esecuzione di Java. Ogni istruzione deve essere completata con il punto e virgola (;), fatta eccezione per le istruzioni di controllo di flusso del codice (if, while, ...)

## 1.3 Identificatori

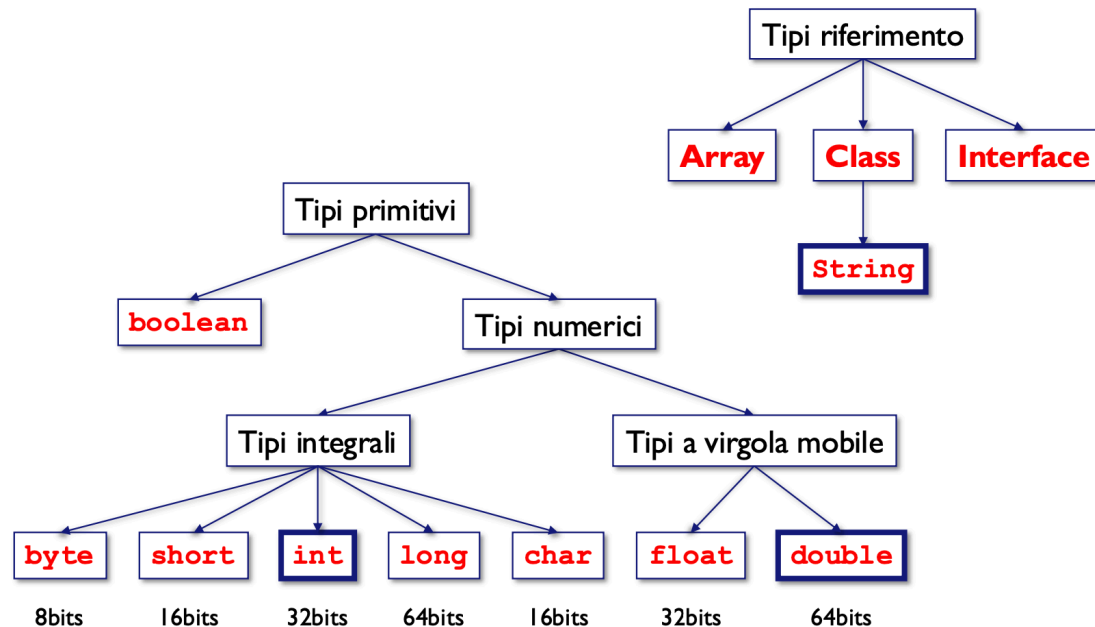
Gli identificatori sono i nomi delle variabili, delle procedure e delle classi. Un identificatore è una sequenza **senza spazi** e di lunghezza illimitata di lettere, cifre e simboli `_` e `$`. Inoltre un identificatore non può essere identico ad una *keyword*, ad un valore booleano (`true` o `false`) o al valore `null`.

- **nomi delle classi**, si utilizza la sintassi *UpperCamelCase*.
- **nomi delle variabili**, si utilizza la sintassi *lowerCamelCase*.

**Nota:** Java è case sensitive.

## 2 Tipi di dato

Java è un linguaggio di programmazione fortemente tipizzato, ogni variabile e ogni espressione ha un tipo conosciuto al momento della compilazione. Ecco uno schema con i principali tipi di dato:



### 2.1 boolean

Il tipo `boolean` può assumere esclusivamente due valori: `true` e `false`. I valori booleani si ottengono anche dalla valutazione di espressioni condizionali, ad esempio:

```
boolean risultato = tasso > 0.05;
```

### 2.2 Tipi numerici

#### 2.2.1 Interi

I tipi numerici interi sono rispettivamente `byte`, `short`, `int` e `long`. L'unica differenza tra di essi è il range di numeri che è possibile rappresentare. Per valori ancora più grandi è possibile utilizzare il tipo di dato `java.math.BigInteger`.

#### 2.2.2 char

Serve per rappresentare i singoli caratteri. Esso permette l'utilizzo dei caratteri **Unicode**, uno standard per la rappresentazione consistente del testo. Ad ogni carattere è assegnato un valore numerico ed è quindi possibile fare operazioni aritmetiche sui caratteri.

#### 2.2.3 Decimali

È molto raro, ma ci sono alcuni numeri decimali che non possono essere rappresentati. In generale per comparare due valori decimali non utilizziamo `==`:

```
static final double EPSILON = 1e-8; //1 * 10^-8
if(Math.abs(a-b) < EPSILON) { //confronta a e b
    ...
}
```

I tipi decimali utilizzati solitamente sono `float` e `double`, per valori ancora più grandi è possibile utilizzare il tipo di dato `java.math.BigDecimal`.

## 2.3 Operatori logici

Per eseguire operazioni con valori di tipo `boolean`:

- `&&` and
- `||` or
- `!` not
- `^` xor

**Nota:** Gli operatori `&&` e `||` sono cortocircuitati. Significa che la seconda espressione viene valutata solo se necessario. Ad esempio:

```
(x != 0) && (y / x > 1)
```

Se `x = 0` allora la condizione `x != 0` restituisce `false` ed è inutile (oltre che dannoso) verificare anche la seconda condizione.

### 2.3.1 Algebra di Boole

Proprietà dell'algebra:

- **commutativa:**

```
A && B = B && A
A || B = B || A
```

- **associativa:**

```
(A && B) && C = A && (B && C)
(A || B) || C = A || (B || C)
```

- **idempotenza:**

```
A && A = A
A || A = A
```

- **assorbimento:**

```
A && (A || B) = A
A || (A && B) = A
```

- **distributiva:**

```
A && (B || C) = (A && B) || (A && C)
A || (B && C) = (A || B) && (A || C)
```

- **esistenza di minimo e massimo:**

```
A && false = false
A || true = true
```

- **esistenza del complemento:**

```
A && (!A) = false
A || (!A) = true
```

- **teoremi di DeMorgan:**

```
!(A && B) = (!A) || (!B)
!(A || B) = (!A) && (!B)
```

## 2.4 Operatori aritmetici

Si applicano a tutti i **tipi numerici**:

- + addizione
- - sottrazione
- \* moltiplicazione
- / divisione
- % modulo (resto della divisione)

### 2.4.1 Operatori ++ e --

L'operazione  $x = x + 1$  può essere scritta come `x++`. L'operazione  $x = x - 1$  può essere scritta come `x--`.

- Operatore postfisso `x++`, prima valuto `x` e poi incremento
- Operatore prefisso `++x`, prima incremento `x` e poi valuto

## 2.5 Letterali

Un letterale è la rappresentazione di un valore di tipo primitivo, una stringa o il valore `null`.

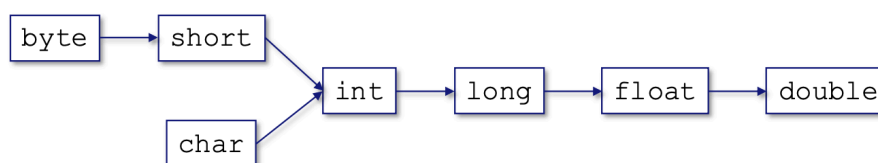
**Nota:** un letterale è di tipo `double` solo se contiene la virgola (5.), altrimenti è di tipo `int`.

**Nota:** un letterale è di tipo `long` solo se termina con la lettera `l` o `L`, altrimenti è di tipo `int`.

**Nota:** un letterale è di tipo `float` solo se termina con la lettera `f` o `F`, altrimenti è di tipo `double`.

## 2.6 Conversioni

Le conversioni **implicite** vengono effettuate automaticamente dal compilatore. Esse avvengono nel caso si passi da un tipo di dato più piccolo ad uno più grande o da un tipo di dato con una precisione minore ad uno con la precisione maggiore.



**Nota:** le conversioni **esplicite**, vanno al contrario rispetto allo schema.

## 3 Introduzione alle classi

### 3.1 La classe Scanner

La classe `Scanner` è un semplice scanner di testo che può analizzare tipi primitivi e stringhe. Per utilizzare uno scanner è necessario importare la classe, istanziarlo e **chiuderlo**:

```
import java.util.Scanner; //importa la classe

public class EsempioHasNextInt {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); //istanzia uno scanner

        System.out.print("Inserisci un numero intero: ");
        String x = scanner.next(); //utilizza lo scanner

        scanner.close(); //chiude lo scanner
    }
}
```

#### 3.1.1 Anomalia dell'input

Quando utilizziamo uno scanner per leggere un numero, premendo invio lasciamo `\n` nel buffer, in questo modo quando poi proviamo a leggere una stringa, verrà letto soltanto il valore lasciato nel buffer. Per evitare questo ogni volta che leggiamo un valore numerico con uno scanner, chiamiamo successivamente il metodo `nextLine()`, in modo che svuoti il buffer prima della lettura della stringa.

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

// Lettura di un numero intero
System.out.print("Inserisci un numero intero: ");
int numero = scanner.nextInt(); // Legge il numero, lascia '\n' nel buffer

// Dopo aver letto un numero con nextInt(), rimane il carattere '\n' nel buffer.
// Il metodo nextLine() viene utilizzato qui per consumare quel '\n' ed evitare
// problemi nella successiva lettura di una stringa.
scanner.nextLine();

// Lettura di una stringa
System.out.print("Inserisci una stringa: ");
String stringa = scanner.nextLine();
```

Per verificare che l'utente inserisca un valore del tipo aspettato quando usiamo uno scanner possiamo usare:

```
System.out.print("Inserisci un numero intero: ");

// Ciclo che continua fino a quando non viene inserito un numero intero valido
while (!scanner.hasNextInt()) { // hasNextInt() controlla se il prossimo input
    // è un numero intero, ma non consuma il valore
    System.out.print("Input non valido. Inserisci un numero intero: ");
    scanner.nextLine(); // Scarta l'input non valido
}

// Una volta ottenuto un intero, lo memorizziamo e lo stampiamo
int number = scanner.nextInt();
```



## 3.2 Stringhe

### 3.2.1 Confronti tra stringhe

```
boolean uguali = str1.equals(str2);
```

### 3.2.2 Conversione di tipi primitivi

- Da String a int

```
String str = "10";  
int num = Integer.parseInt(str);
```

- Da String a double

```
String str = "17.42e-2";  
double num = Double.parseDouble(str);
```

- Da valore numerico a double

```
int i = 42;  
String s1 = "" + i;  
String s2 = String.valueOf(i);  
String s3 = Integer.toString(i);
```

- Da String a valore numerico

```
int i1 = Integer.parseInt("17030075");  
int i2 = Integer.valueOf("17030075");
```

### 3.2.3 Operazioni sulle stringhe

- `s1.length()`, lunghezza della stringa
- `s1.charAt(i)`, carattere alla posizione `i`
- `s1.substring(i, j)`, nuova stringa formata dai caratteri da posizione `i` inclusa fino a `j` esclusivamente
- `s1.indexOf(ch)`, ottieni l'indice del carattere o della stringa all'interno di `s1`
- `s1.toLowerCase()`, ottieni una copia tutto in minuscolo
- `s1.toUpperCase()`, ottieni una copia tutto in maiuscolo
- `s1.equalsIgnoreCase(s2)`, confronto ignorando maiuscole e minuscole
- `s1.trim()`, ottieni una copia senza spazi a inizio e fine

### 3.3 La classe Math

La classe Math mette a disposizione le seguenti funzioni:

- `Math.pow(base, esponente)`, elevamento a potenza
- `Math.sqrt(numero)`, radice quadrata
- `Math.log(numero)`, logaritmo naturale
- `Math.log10(numero)`, logaritmo in base 10
- `Math.sin(radiani)`, seno di un angolo in radianti
- `Math.cos(radiani)`, coseno di un angolo in radianti
- `Math.tan(radiani)`, tangente di un angolo in radianti
- `Math.random()`, genera un numero casuale tra 0.0 (compreso) e 1.0 (escluso)

E le seguenti costanti:

- `Math.E`, base del logaritmo naturale
- `Math.PI`, pi greco

### 3.4 Generare numeri casuali

Per generare numeri casuali con `Math.random()` si utilizza:

```
int randomInt = (int) ((max - min) * Math.random() + min)
```

Dove `max` rappresenta il valore massimo del range (non compreso) e `min` quello minimo (compreso).

Se ad esempio volessi avere un range contenente tutti i numeri da 10 a 50 (entrambi compresi) sarebbe:

```
int randomInt = (int) ((51 - 10) * Math.random() + 10) //il 51 non è compreso
```

## 4 Istruzioni di controllo

### 4.1 Switch

Ci sono due sintassi per l'istruzione `switch` in Java:

- ```
switch (espressione) {  
    case valore1:  
        ...;  
        break;  
    case valore2:  
        ...;  
        break;  
    case valore3:  
        ...;  
        break;  
    default:  
        ...;  
}
```

**Nota:** L'opzione `default` è opzionale. Se non è presente il `break`, il programma continuerà al `case` successivo, anche se la condizione non è soddisfatta.

È anche possibile concatenare più `case` su una sola linea se la condizione è la stessa:

- ```
switch (espressione) {  
    case valore4: case valore6: case valore9:  
        ...;  
        break;  
    case valore2:  
        ...;  
        break;  
    default:  
        ...;  
}
```
- ```
switch (espressione) {  
    case valore1 -> {  
        ...;  
    }  
    case valore2 -> {  
        ...;  
    }  
    case default -> {  
        ...;  
    }  
}
```

**Nota:** questa versione non necessita dei `break`, impedisce il `fall-through`

Analogamente all'esempio di prima possiamo fare:

```
switch (espressione) {  
    case valore1, valore2, valore3 -> ...;  
    case valore4, valore5, valore6 -> ...;  
    default -> ...;  
}
```

**Nota:** per assegnare un valore quando si utilizzano i blocchi di codice dobbiamo usare la keyword `yield`. Se vogliamo assegnare un valore di default dobbiamo utilizzarlo per forza.

Gli `switch` funzionano con:

- `char`
- `byte`
- `short`
- `int`
- `String`

**Nota:** non funziona con valori decimali come ad esempio `double`.

## 4.2 Operatore ternario

Detto anche `inline if`, restituisce il primo valore se la condizione è vera mentre restituisce il secondo se la condizione è falsa:

```
condizione ? valoreSeVero : valoreSeFalso
```

Ad esempio:

```
if (n % 2 == 0) {  
    prossimo = n / 2;  
} else {  
    prossimo = 3 * n + 1;  
}  
  
// sono uguali  
prossimo = (n % 2 == 0) ? (n / 2) : (3 * n + 1);
```

## 4.3 Ciclo for

```
for (inizializzazione; condizione; aggiornamento) {  
    seqIstruzioni;  
}
```

**Nota:** nessuno dei parametri del ciclo `for` (`inizializzazione`; `condizione`; `aggiornamento`) è obbligatorio.

È anche possibile utilizzare più variabili o condizioni:

```
for (double i = 0.5, j = 39; i + j < 40; i++, j--) {  
    ...  
}
```

## 4.4 L'istruzione `continue` e `break`

Queste due istruzioni vengono utilizzate all'interno di un ciclo:

- `continue` fa continuare il ciclo dall'iterazione successiva
- `break` esce dal ciclo corrente

## 4.5 L'istruzione `do while`

Prima esegue le istruzioni (almeno una volta) poi verifica se la condizione è vera.

```
do {  
    seqIstruzioni;  
} while (condizione);
```

## 5 Array

- Tipi di dato non strutturati, ogni identificatore è riferito ad un unico elemento
  - `int`
  - `float`
  - `char`
- Tipi di dato strutturati, mediante ogni identificatore è possibile accedere agli elementi aggregati
  - `arrays`
  - `classi`

**Nota:** in java gli array hanno una lunghezza fissa e possiedono elementi di un unico tipo.

### 5.1 Sintassi

#### 5.1.1 Creazione

```
// Dichiarazione e creazione di un array con valori di default
int[] voti;
voti = new int[5];

// In un'unica riga
int[] voti = new int[5];

// Con inizializzazione diretta
int[] voti = {28, 30, 25, 27, 29};
```

**Nota:** nei primi due casi l'array non viene inizializzato e di conseguenza ad ogni cella dell'array viene assegnato un valore di default, nel caso del tipo `int` è 0.

#### 5.1.2 Utilizzare gli elementi

```
// Assegnare un valore ad un elemento
voti[0] = 28;

Assegnare un elemento di un array a una variabile
int primoVoto = voti[0];
```

#### 5.1.3 Proprietà

```
// Lunghezza dell'array
int numeroVoti = voti.length;

// L'ultimo valore dell'array
int ultimoVoto = voti[voti.length - 1];

// QUESTO GENERA ERRORE DURANTE L'ESECUZIONE (NON DURANTE LA COMPILAZIONE)
int errore = voti[voti.length];
```

**Nota:** `length` non è un metodo come nel caso delle stringhe quindi non vanno messe le parentesi.

## 5.2 foreach

È possibile scorrere l'intero contenuto di un array con un ciclo `foreach`:

```
for(int value : voti) {  
    System.out.println(value);  
}
```

## 5.3 Confronto

Due array sono uguali se hanno la stessa lunghezza e se hanno gli stessi valori nelle stesse posizioni.

Possiamo implementare questa logica a mano o utilizzare il metodo `Arrays.equals`:

```
import java.util.Arrays;  
  
System.out.println("Gli array sono uguali? " + Arrays.equals(numeri1, numeri2));
```

## 5.4 Trovare massimo/minimo

```
int max = numeri[0];  
for(int i = 1; i < numeri.length; i++) {  
    if(numeri[i] > max) {  
        max = numeri[i];  
    }  
}
```

## 5.5 Aumentare la lunghezza

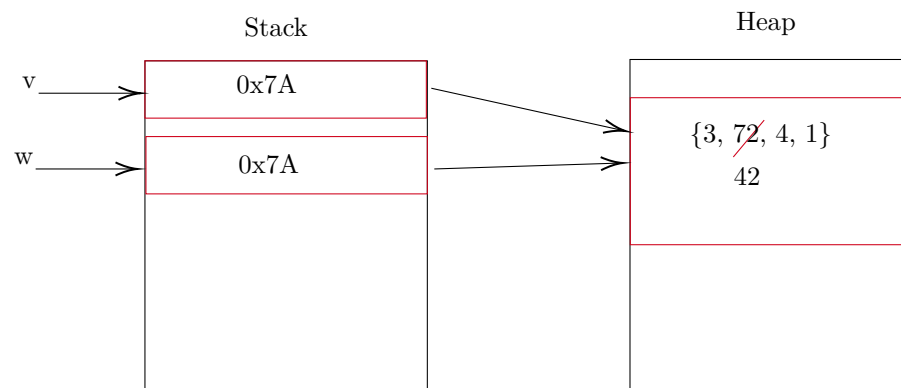
```
int[] valori = {1, 2, 3, 4};  
  
int[] tmp = new int[valori.length + 1];  
  
for(int i = 0; i < valori.length; i++)  
    tmp[i] = valori[i];  
  
valori = tmp;
```

## 5.6 Gestione della memoria

```
int[] v = {3, 72, 4, 1};  
int[] w = v;
```

*// Cambia per entrambi gli array perchè puntano allo stesso indirizzo di memoria*  
w[1] = 42;

Memoria volatile (RAM)



## 5.7 Copiare un array

### 5.7.1 Shallow copy

- Allocare la memoria necessaria
- Copiare valore per valore

```
int[] v = {3, 72, 4, 1};  
  
int[] w = new int[v.length]  
  
for(int i = 0; i++ i < v.length; i++) {  
    w[i] = v[i];  
}
```

che corrisponde a fare

```
int[] v = {3, 72, 4, 1};  
  
int[] w = Arrays.copyOf(v, v.length);
```

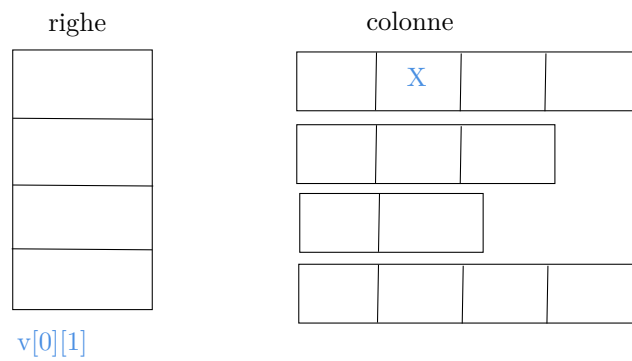
### 5.7.2 Deep copy

- Allocare la memoria necessaria
- Copiare valore per valore
- Se i valori degli array sono oggetti, allocare la memoria e copiare i valori ricorsivamente

**Nota:** nel caso di tipi primitivi, shallow copy e deep copy producono lo stesso risultato. Le cose cambiano se ci sono dei tipi riferimento.

## 5.8 Array multidimensionali

- Array di array
- Devono essere tutti dello stesso tipo
- Ogni riga può avere una lunghezza diversa (array frastagliati)





## 6 Enumerativi

Un enumerativo definisce un nuovo tipo di dato. Se ad esempio vogliamo creare un tipo **Semaforo** con i soli valori **ROSSO**, **GIALLO**, **VERDE** potremmo usare un **enum** e utilizzarlo nel seguente modo:

```
enum Semaforo {
    ROSSO, GIALLO, VERDE
}

public class Prova {
    public static void main(String[] Args) {
        Semaforo statoSemaforo = Semaforo.ROSSO;
        Semaforo statoSemaforo = Semaforo.GIALLO;
        Semaforo statoSemaforo = Semaforo.VERDE;
    }
}
```

**Nota:** gli **enum** si creano fuori dalla classe.

**Nota:** il valore di un tipo **enum** è costante, non si può cambiare il contenuto delle graffe fuori dalla classe.

### 6.1 Operazioni

- `v1.ordinal()` permette di scoprire la posizione del valore `v1` nella lista dei valori di tipo enumerativo
- `NomeTipoEnum.valueOf(s1)` converte la stringa `s1` nel valore corrispondente all'enumerativo `NomeTipoEnum`
- `NomeTipoEnum.values()` restituisce un array contenente tutti i valori dell'enumerativo `NomeTipoEnum`

### 6.2 Enum e switch

```
enum Stagione {
    PRIMAVERA, ESTATE, AUTUNNO, INVERNO
}

public class Prova {
    public static void main(String[] Args) {
        Stagione stagione = Stagione.valueOf(input.nextLine().toUpperCase());
        switch(stagione) {
            case PRIMAVERA
                ...
                break
            case ESTATE
                ...
                break
            case AUTUNNO
                ...
                break
            case INVERNO
                ...
                break
        }
    }
}
```

**Note:** Attenzione, se si utilizza la sintassi nuova, per poter compilare il codice bisogna avere inserito tutti i casi oppure definire un valore di default.

```
enum Stagione {  
    PRIMAVERA, ESTATE, AUTUNNO, INVERNO  
}  
  
public class Prova {  
    public static void main(String[] Args) {  
        String mesi = switch (stagione) {  
            case PRIMAVERA -> "Marzo, Aprile, Maggio";  
            case ESTATE -> "Giugno, Luglio, Agosto";  
            case AUTUNNO -> "Settembre, Ottobre, Novembre";  
        }  
    }  
}
```

Questo esempio non compila perchè manca il caso INVERNO.

## 7 Sottoprogrammi

Un sottoprogramma in Java viene chiamato metodo. I metodi vanno dichiarati fuori dal `main` e dentro alla classe. Un sottoprogramma può essere una funzione o una procedura. Le funzioni ritornano qualcosa mentre le procedure no.

```
private static int getUserInput(Scanner input, String message) {  
    System.out.println(message);  
  
    return input.nextInt();  
}
```

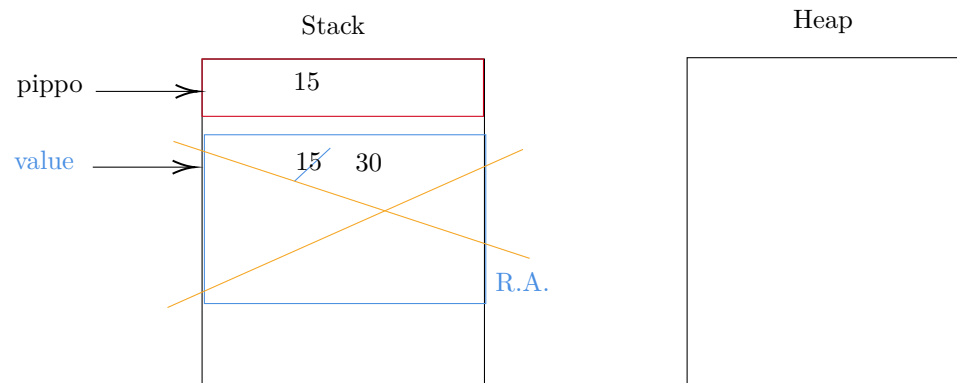
**Nota:** possono esistere due metodi con lo stesso nome purchè possiedano parametri diversi (overloading).

### 7.1 Gestione della memoria

#### 7.1.1 Tipi primitivi

```
int pippo = 15;
```

```
//Questa funzione ritorna il doppio del valore passato come parametro  
makeItDouble(pippo); //Viene copiato il valore di pippo (value)
```

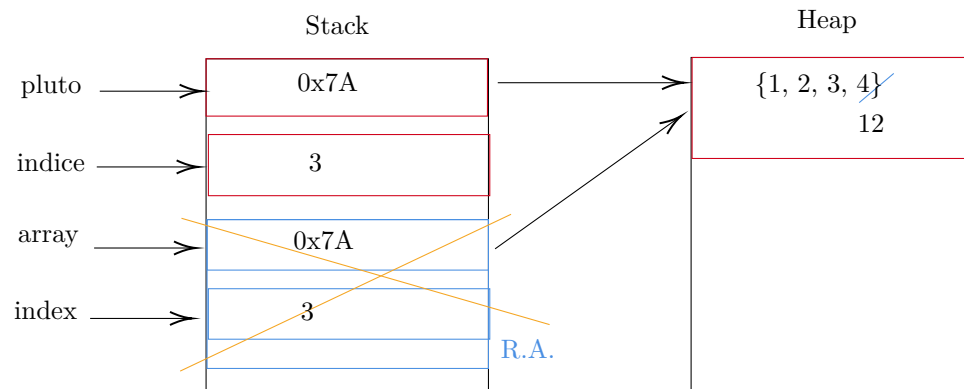


**Nota:** in questo caso il valore della variabile non cambia.

### 7.1.2 Tipi riferimento

```
int[] pluto = {1, 2, 3, 4};
int indice = 3;

makeItTriple(pluto, indice);
```



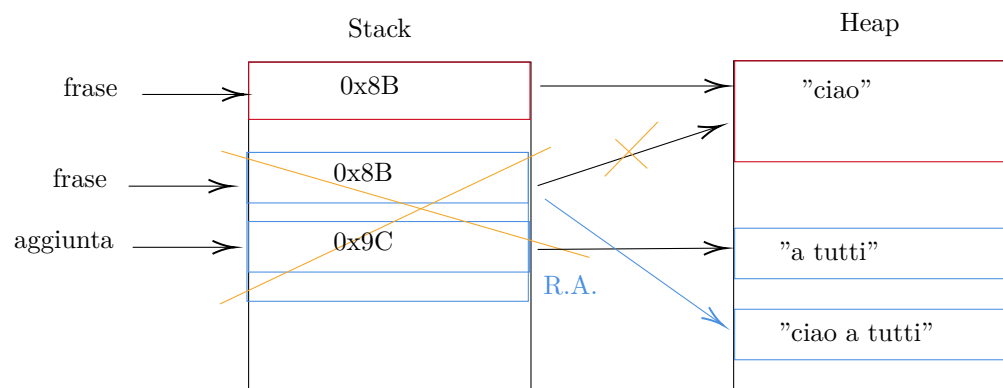
**Nota:** in questo caso il valore dell'array effettivamente cambia.

### 7.1.3 Stringhe

```
String frase = "ciao";

modificaStringa(frase);

modificaStringa(frase) {
    frase += aggiunta;
}
```



**Nota:** in questo caso il valore della stringa non cambia.

## 8 Ricorsione

```
private static int sumUpToN(int n) {
    System.out.println("n: " + n)
    if(n == 1) {
        return 1;
    }
    int result = sumUpToN(n - 1);
    return result
}
```

**Nota:** ogni problema ricorsivo può essere risolto anche in modo iterativo:

```
private static int sumUpToN(int n) {
    int result = 0;
    for(int i = n; i > 0; i++) {
        result += i;
    }
    return result;
}
```

**Nota:** la ricorsione è una soluzione più "elegante" ma utilizza più risorse, la soluzione iterativa è più efficiente da un punto di vista di tempo e risorse.

### 8.0.1 Alcune definizioni

- **Fase di svolgimento:** la procedura continua a chiamare se stessa finchè non incontra la condizione di terminazione
- **Fase di riavvolgimento:** incontra la condizione di terminazione e le chiamate annidate si chiudono
- **Tail recursion:** la ricorsione è alla fine
- **Head recursion:** la ricorsione è all'inizio
- **Middle recursion / multi recursion:** la ricorsione è in mezzo
- **Mutual recursion / indirect recursion:** due funzioni X e Y si chiamano a vicenda
- **Ricorsione multipla:** una funzione richiama più volte se stessa

### 8.0.2 Approccio

- Dividere il problema in problemi più semplici ma della stessa natura
- Trovare il caso base
- Risolvere il problema

## 9 Oggetti

Una classe in java è definita come segue

```
class Libro {
    String titolo;
    String nomeAutore;
    String cognomeAutore;
    int numeroPagine;
}

public class BibliotecaTest{
    public static void main(String[] args) {
        Libro libro1 = new Libro();
        libro1.titolo = "Viaggio intorno al mondo";
        libro1.nomeAutore = "Pippo";
        libro1.cognomeAutore = "Pluto";
        //libro1.numeroPagine = 100;

        System.out.println("Titolo: " + libro1.titolo);
    }
}
```

**Nota:** le variabili all'interno della classe Libro vengono chiamate campi della classe (titolo, nomeAutore, ...).

**Nota:** L'istanza di una classe è un oggetto (libro1).

**Nota:** Se quando istanziamo un oggetto non specifichiamo il valore di un campo verrà assegnato un valore di default (0 per gli interi, null per le stringhe, false per i booleani).

### 9.1 Costruttori

```
class Libro {
    String titolo;
    String nomeAutore;
    String cognomeAutore;
    int numeroPagine;

    // COSTRUTTORE
    Libro(String titolo, String nomeAutore, String cognomeAutore, int numeroPagine) {
        this.titolo = titolo;
        this.nomeAutore = nomeAutore;
        this.cognomeAutore = cognomeAutore;
        this.numeroPagine = numeroPagine;
    }
}

public class BibliotecaTest{
    public static void main(String[] args) {
        Libro libro1 = new Libro("Viaggio intorno al mondo", "Pippo", "Pluto", 100);

        System.out.println("Titolo: " + libro1.titolo);
    }
}
```

**Nota:** è possibile definire più costruttori a patto che abbiano un numero di parametri diversi o il tipo dei parametri diversi.

È anche possibile richiamare un costruttore all'interno di un altro costruttore per non duplicare linee di codice utilizzando la keyword `this`.

```
class Libro {
    String titolo;
    String nomeAutore;
    String cognomeAutore;
    int numeroPagine;

    // COSTRUTTORE
    Libro(String titolo, String nomeAutore, String cognomeAutore, int numeroPagine) {
        this(titolo, nomeAutore, cognomeAutore)

        this.numeroPagine = numeroPagine < 0 ? 10 : numeroPagine;
    }

    // COSTRUTTORE
    Libro(String titolo, String nomeAutore, String cognomeAutore) {
        this.titolo = titolo;
        this.nomeAutore = nomeAutore;
        this.cognomeAutore = cognomeAutore;
    }
}
```