



UNIVERSITÀ  
DEGLI STUDI  
DI TRIESTE

# Fondamenti di Informatica (117IN)

A.A. 2022 / 2023

## Lezione 8 - Funzioni

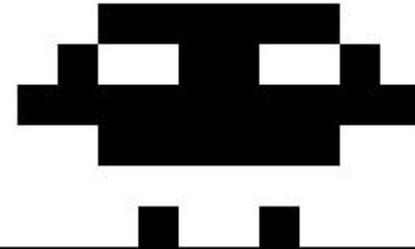
Sylvio Barbon Junior  
sylvio.barbonjunior@units.it

Sommario:

- 1) Funzioni
  - a) Struttura
  - b) Visibilità e Scope
  - c) Overloading, Ricorsione

## L'ultima lezione...

```
1 import java.io.File;
2 import java.awt.image.BufferedImage;
3 import javax.imageio.ImageIO;
4
5 public class Image{
6     public static void main(String[] args) throws Exception {
7         File file= new File("Alien.png");
8         BufferedImage img = ImageIO.read(file);
9         System.out.println("Image (height):"+img.getHeight());
10        System.out.println("Image (width):"+img.getWidth());
11        for (int y = 0; y < img.getHeight(); y++) {
12            System.out.println("");
13            for (int x = 0; x < img.getWidth(); x++) {
14                int pixel = img.getRGB(x,y);
15                if(pixel==-1)
16                    System.out.print("B");
17                else
18                    System.out.print("N");
19            }
20        }
21        System.out.println("");
22    }
23 }
```



```
barbon@xps: ~/Downloads/FI_2023/java
barbon@xps:~/Downloads/FI_2023/java$ java Image
Image (height):8
Image (width):16

BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBNNNNNNBBBB
BBBBNBBNNBBNBBBB
BBBBNNNNNNNNBBBB
BBBBBBBBBBBBBBBB
BBBBBBNBBNBBBB
barbon@xps:~/Downloads/FI_2023/java$
```

Lezione precedente:

primitivi

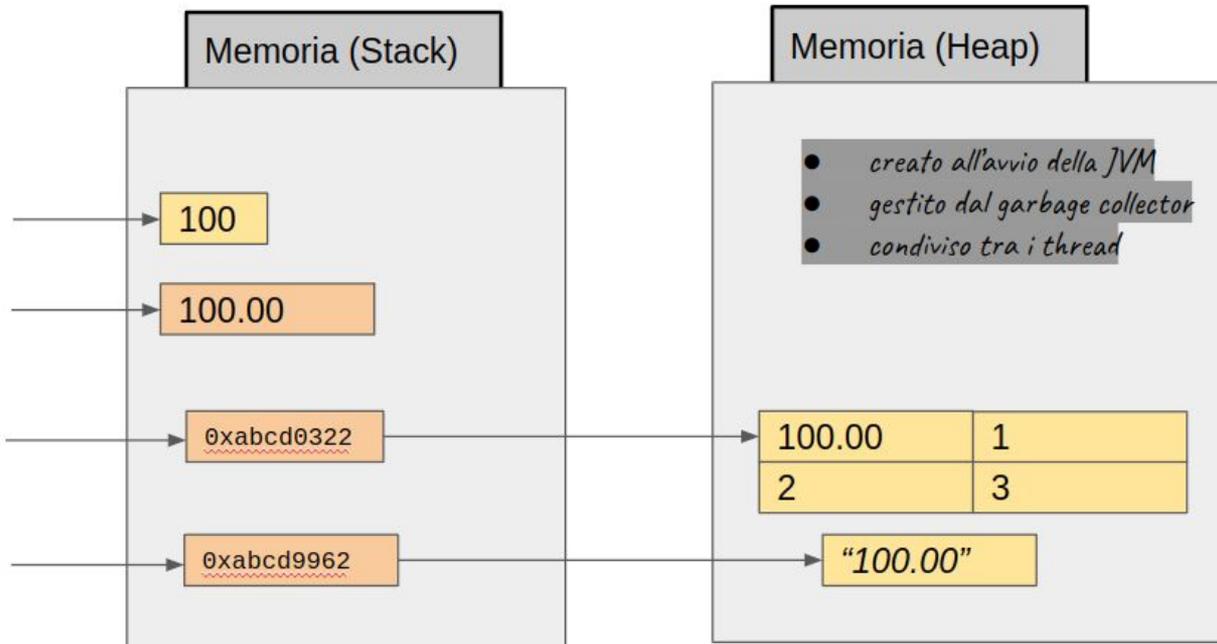
`int i = 100;`

`double d = 100;`

derivati

`int[] c = {100,1,2,3}`

`String str = "100";`



## 1) Funzioni

- La classe principale di un programma è la funzione `main()`;
- Possiamo creare le nostre funzione che saranno richiamata dalla `main()` o altre classi;
- Nella definizione di una funzione occorre indicare:
  - i) `tipo`;
  - ii) `il tipo del risultato`;
  - iii) `nome`;
  - iv) gli eventuali `parametri`;
  - v) specificare `quali azioni` deve eseguire;

```
public static void main(String[] args) {  
    System.out.println("Ciao!");  
}
```

*Il metodo `Java main()` è sempre statico, in modo che il compilatore possa chiamarlo senza la creazione di un oggetto o prima della creazione di un oggetto della classe!!!*

## 1) Funzioni - struttura

- Il corpo di una funzione (essendo un blocco) **non può contenere altre definizioni di funzione**, ma solo definizioni di:
  - i) **variabili**;
  - ii) **classi**;
- Una funzione può restituire come valore **un risultato**, oppure non avere nessun risultato (parola chiave **void**);
- Il corpo deve prevedere almeno una istruzione **return**, con la specifica dell'espressione il cui valore rappresenta il risultato stesso;

## 1) Funzioni - struttura

```
1  public class L7_e1{
2
3      public static void main(String[] args) {
4          int numA = 10;
5          int numB = 12;
6          int result = sommaNum(numA, numB);
7          System.out.println("The result is:"+result);
8      }
9
10     public static int sommaNum(int x, int y){
11         int output = x + y;
12         return(output);
13     }
14
15 }
16
17
```

## 1) Funzioni - struttura

The diagram illustrates the structure of a Java class and its methods. The code is as follows:

```
1 public class L7_e1{
2
3     public static void main(String[] args) {
4         int numA = 10;
5         int numB = 12;
6         int result = sommaNum(numA, numB);
7         System.out.println("The result is:"+result);
8     }
9
10    public static int sommaNum(int x, int y){
11        int output = x + y;
12        return(output);
13    }
14 }
15
16
17
```

Annotations and their corresponding parts in the code:

- Classe**: A large bracket on the left side of the code block, encompassing the entire class definition from line 1 to 17.
- tipo**: A pink box with an arrow pointing to the `void` keyword in the `main` method signature.
- tipo risultato**: A pink box with an arrow pointing to the `int` keyword in the `sommaNum` method signature.
- nome**: A blue box with an arrow pointing to the `main` and `sommaNum` identifiers.
- parametro**: A cyan box with an arrow pointing to the `String[] args` parameter in the `main` method.
- parametri**: A cyan box with an arrow pointing to the `int x, int y` parameters in the `sommaNum` method.
- ritorno**: An orange box with an arrow pointing to the `return` statement in the `sommaNum` method.
- chiamata della funzione.**: A yellow box with an arrow pointing to the `sommaNum(numA, numB)` call inside the `main` method.
- Funzione *main***: A pink box with a bracket on the right side, encompassing the `main` method body (lines 3-8).
- Funzione *sommaNum***: A pink box with a bracket on the right side, encompassing the `sommaNum` method body (lines 10-13).

## 1) Funzioni - Visibilità

- La visibilità di proprietà e funzioni:
  - i) **Public**: sono accessibili sia dall'interno che dall'esterno della classe;
  - ii) **Protected**: utilizzati all'interno della classe stessa e all'interno delle classi derivate
  - iii) **Private**: possono essere utilizzati soltanto all'interno della classe stessa.
  - iv) **Default** (se non specifichiamo): accessibile solo da tutte le classi nel medesimo package.
- Queste definizioni verranno trattate in modo approfondito quando si tratta di **orientazione oggetti**.
- Le **variabile** possono essere dichiarati con le proprietà di visibilità;

## 1) Funzioni - Visibilità e Scope

- **Scope** di una variabile l'area del codice nel quale un identificatore resta associato ad un indirizzo di memoria;
- In Java si distinguono tre tipi di variabili:
  - i) variabili **locali** (all'interno di un metodo);
  - ii) variabili di **istanza** (all'esterno di ogni metodo;)
  - iii) variabili di **classe** ('static');

*ogni blocco (cioè ogni gruppo di linee di codice racchiuso da parentesi graffe { }) definisce uno scope e ogni variabile locale ha come scope l'area di codice che inizia dalla definizione della variabile stessa e termina con il blocco corrente.*

*sono anche chiamati come globali!*

*per le variabili statiche esiste un'unica locazione di memoria*

iv) **static + final**: **final** per dichiarare una variabile che potrà essere inizializzata una sola volta e con stati, si definiscono una **costanti**;

## 1) Funzioni - Visibilità e Scope

```
1 public class L7_e2{
2
3     public static final double PI = 3.141592; ← costanti
4
5     public static void main(String args[]){
6         double raggio = Double.parseDouble(args[0]); ← parametro del terminale
7
8         System.out.println("L'area della sfera è: "+volumeSfera(raggio));
9     }
10
11
12     public static double volumeSfera(double r){
13         return (4/3 * PI * r * r * r); ← locale, solo nello scopo "volumeSfera"
14     }
15
16 }
```

locale, solo nello scopo "main"

barbon@barbon-PC:~/Scaricati/FI/codici\$ java L7\_e2 2

L'area della sfera è: 25.132736

## 1) Funzioni - Visibilità e Scope

```
1 public class L7_e3{
2
3     public static final double PI = 3.141592;
4     static double raggio;
5
6     public static void main(String args[]){
7         raggio = Double.parseDouble(args[0]);
8
9         System.out.println("L'area della sfera è: "+volumeSfera());
10    }
11
12
13    public static double volumeSfera(){
14        return (4/3 * PI * raggio * raggio * raggio);
15    }
16
17 }
```

**visibilità default** → `public static final double PI = 3.141592;`

**costanti** ← `public static final double PI = 3.141592;`

**variabile d'istanza** → `static double raggio;`

**senza parametri** → `public static double volumeSfera(){`

```
barbon@barbon-PC:~/Scaricati/FI/codici$ java L7_e3 2
L'area della sfera è: 25.132736
```

## 1) Funzioni - Overloading

- Funzioni differenti possono avere lo stesso **identificatore** (sovrapposizione o overloading), esse devono però avere una **differente firma** (signature):
  - i) parametri formali differenti in numero;
  - ii) e/o tipo;
  - iii) e/o ordine;

```
public static double volumeSphera(double raggio){  
    return (4/3 * PI * raggio * raggio * raggio);}
```

```
public static double volumeSphera(int raggio){  
    return (4/3 * PI * raggio * raggio * raggio);}
```

```
public static double volumeSphera(String str){  
    double raggio = Double.parseDouble(str);  
    return (4/3 * PI * raggio * raggio * raggio);}
```

## 1) Funzioni - Overloading

```
1 public class L7_e4{
2
3     public static final double PI = 3.141592;
4
5     public static void main(String args[]){
6         double raggio = Double.parseDouble(args[0]);
7
8         System.out.println("L'area della sfera è: "+volumeSfera(raggio));
9         System.out.println("L'area della sfera è: "+volumeSfera(args[0]));
10    }
11
12
13    public static double volumeSfera(double raggio){
14        return (4/3 * PI * raggio * raggio * raggio);
15    }
16
17    public static double volumeSfera(int raggio){
18        return (4/3 * PI * raggio * raggio * raggio);
19    }
20
21    public static double volumeSfera(String str){
22        double raggio = Double.parseDouble(str);
23        return (4/3 * PI * raggio * raggio * raggio);
24    }
25
26 }
```

```
barbon@barbon-PC: ~/Scaricati/FI/codici
barbon@barbon-PC:~/Scaricati/FI/codici$ java L7_e4 25
L'area della sfera è: 49087.375
L'area della sfera è: 49087.375
barbon@barbon-PC:~/Scaricati/FI/codici$
```

Overloading

## 1) Funzioni - Ricorsione

- Una funzione può invocare non solo un'altra funzione, ma anche se stessa;
- La funzione è **ricorsiva** (si ha una nuova istanza della medesima funzione mentre l'istanza attuale non è ancora conclusa);
- Un **classico esempio** è quello del calcolo del fattoriale di un numero intero positivo  $n$ , che può essere espresso ricorsivamente nel seguente modo:

i) fattoriale ( $n$ ) =

$$(1) \quad 1 \qquad \qquad \qquad \text{se } n = 0$$

$$(2) \quad n * \text{fattoriale } (n-1) \qquad \text{se } n > 0$$

## 1) Funzioni - Ricorsione

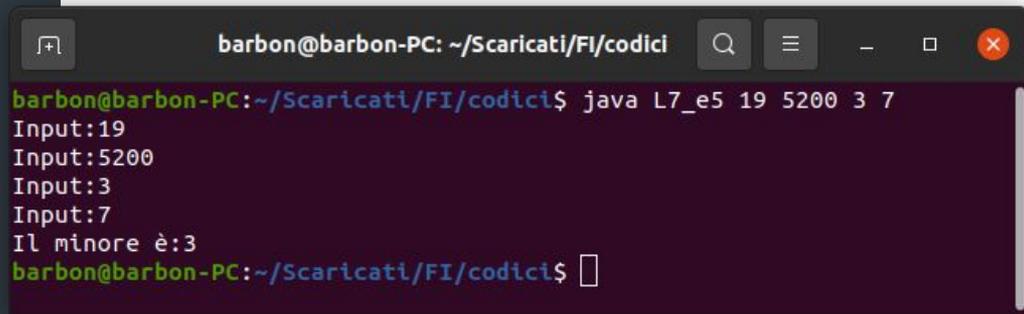
```
1 public class Ricorsione{
2     public static void main(String args[]){
3         int n = Integer.parseInt(args[0]);
4         System.out.println("Fattoriale di "+n+" è "+fatt(n));
5     }
6
7     static int fatt(int n){
8         if (n == 0){
9             return 1;
10        }
11        return n*fatt(n-1);
12    }
13 }
```

The diagram illustrates the recursive call stack for calculating the factorial of 4. The stack starts with `fatt(4)` and goes down to `fatt(0)`. Each call returns a value back to the previous call. The calculations shown are: `fatt(0)` returns 1; `fatt(1)` returns  $1 \cdot 1 = 1$ ; `fatt(2)` returns  $2 \cdot 1 = 2$ ; `fatt(3)` returns  $3 \cdot 2 = 6$ ; `fatt(4)` returns  $4 \cdot 6 = 24$ .

```
barbon@barbon-PC:~/Scaricati/FI/codici$ java Ricorsione 4
Fattoriale di 4 è 24
```

## 1) Funzioni

```
1 //algoritmo che restituisce il numero minore
2 public class L7_e5{
3
4     public static void main(String args[]){
5         int input[] = new int[args.length];
6
7         for(int i=0;i<args.length;i++){
8             System.out.println("Input:"+args[i]);
9             input[i] = Integer.parseInt(args[i]);
10        }
11
12
13        int ouput = minore(input);
14        System.out.println("Il minore è:"+ouput);
15    }
16
17    public static int minore(int input[]){
18
19        int piccolo=9999;
20        for(int i=0; i<input.length; i++){
21            if (input[i]<piccolo){
22                piccolo = input[i];
23            }
24        }
25        return(piccolo);
26    }
27 }
```



```
barbon@barbon-PC: ~/Scaricati/FI/codici
barbon@barbon-PC:~/Scaricati/FI/codici$ java L7_e5 19 5200 3 7
Input:19
Input:5200
Input:3
Input:7
Il minore è:3
barbon@barbon-PC:~/Scaricati/FI/codici$
```

Grazie!

