

SUPSI

Ricorsione

Loris Grossi, Fabio Landoni, Andrea Baldassari

Contenuto realizzato in collaborazione con: T. Leidi, A.E. Rizzoli, S. Pedrazzini

Fondamenti di Informatica

Bachelor in Ingegneria Informatica

Obiettivo

Comprendere la dichiarazione e l'utilizzo delle procedure e delle funzioni ricorsive all'interno di programmi Java.

Obiettivi della lezione:

- Conoscere il concetto di ricorsione e il relativo funzionamento.
- Saper scrivere una procedura/funzione ricorsiva.
- Saper applicare la ricorsione quale tecnica risolutiva di problemi di programmazione.
- Conoscere le fasi e la gestione della memoria nel caso di chiamate a procedure/funzioni ricorsive.
- Conoscere il concetto di condizione d'uscita e di ricorsione infinita.
- Conoscere i diversi tipi di ricorsione (head, middle, tail e mutual).
- Conoscere la differenza fra algoritmo iterativo e algoritmo ricorsivo.

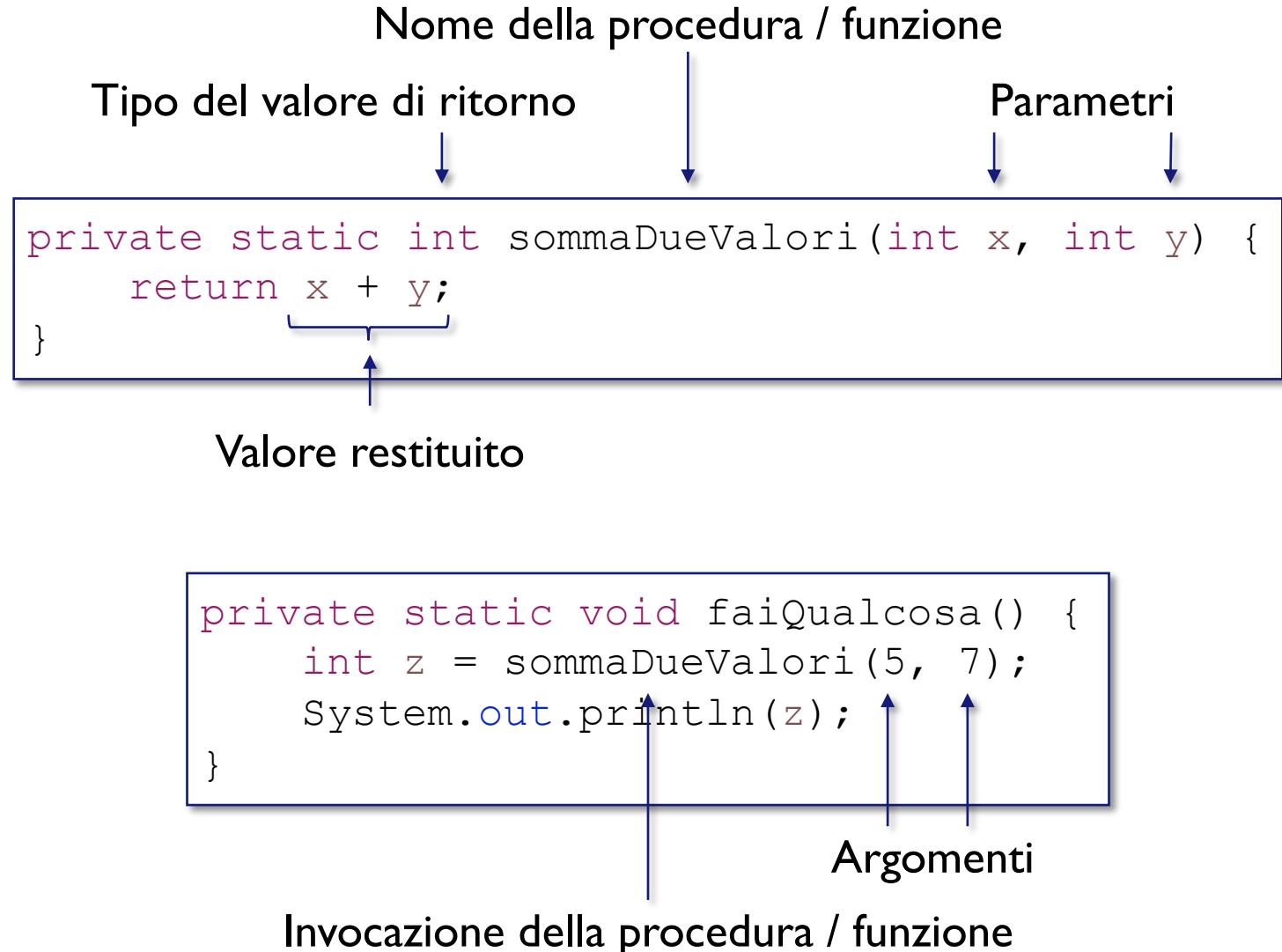
Sottoprogrammi: procedure e funzioni

Un **sottoprogramma** (subroutine) può essere una **procedura o una funzione**.

Ogni procedura / funzione è un **blocco di istruzioni che ha un nome** e che **può essere invocato** all'interno di una qualsiasi espressione nel codice.

Le procedure / funzioni favoriscono la **strutturazione e il riutilizzo del codice sorgente**.

Procedure e funzioni: esempio



Completamento di procedure e funzioni

Una procedura o funzione **termina** quando:

- vengono **eseguite tutte le istruzioni** al suo interno,
- viene eseguita l'istruzione **return**,
- viene lanciata un'eccezione (ne discuteremo il prossimo semestre).

L'istruzione `return` in una procedura

Se il tipo del valore di ritorno è **void** si tratta di una **procedura** (non restituisce alcun valore).

Per una procedura, ***I'istruzione return è opzionale*** e viene utilizzata qualora fosse necessario interromperne l'esecuzione.

```
private static void mostraValoreSePositivo(int valore) {  
    if (valore >= 0)  
        System.out.println(valore);  
}
```

```
private static void mostraValoreSePositivo(int valore) {  
    if (valore < 0)  
        return;  
    System.out.println(valore);  
}
```

L'istruzione return in una funzione

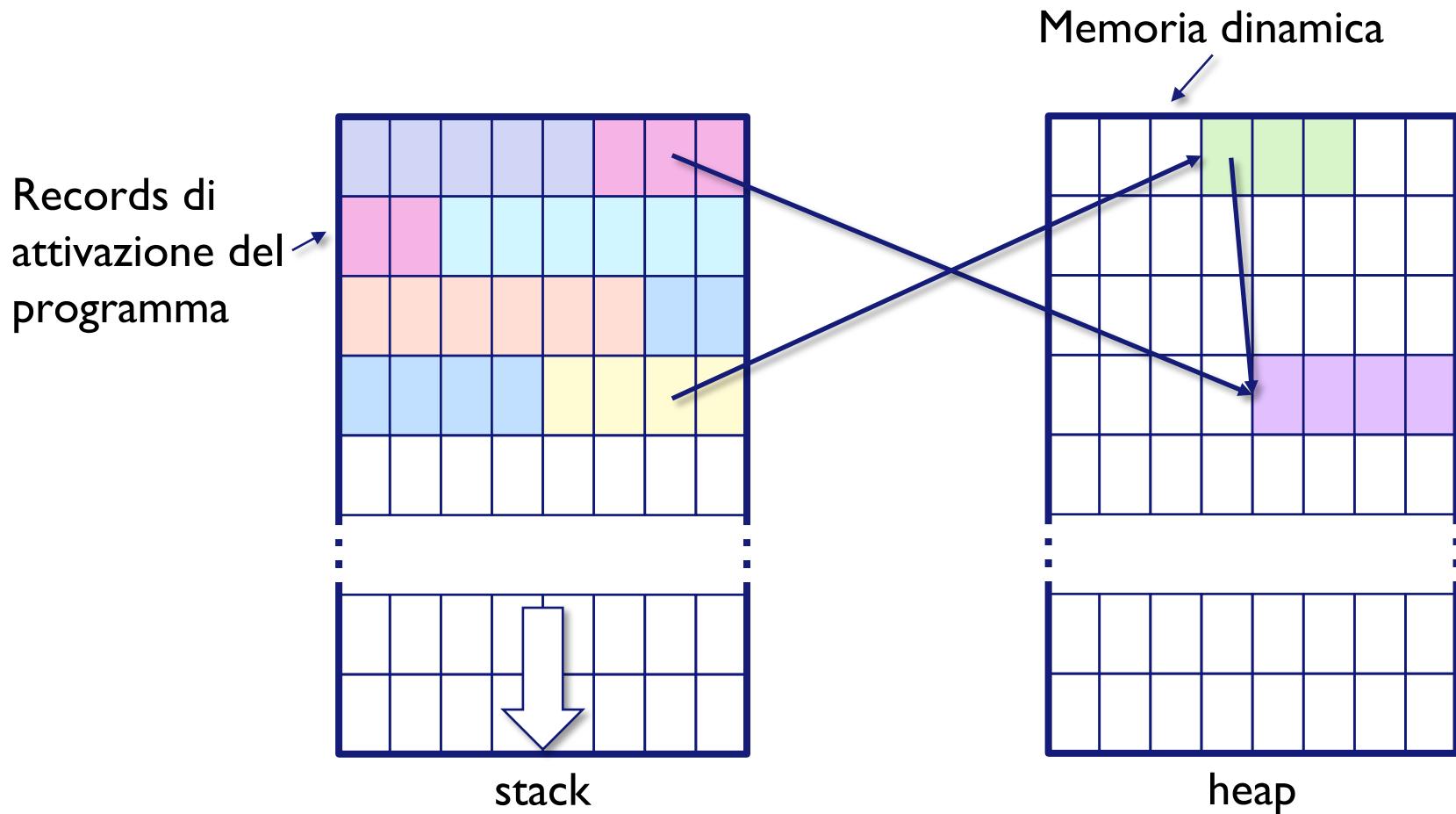
In una funzione, ***l'istruzione return è obbligatoria*** e viene utilizzata per ***restituire un valore*** di ritorno.

Anche nel caso delle funzioni, l'istruzione `return` interrompe l'esecuzione.

```
private static int leggiValorePositivo(Scanner scanner) {  
    System.out.print("Inserisci un valore positivo: ");  
    int x = scanner.nextInt();  
    if (x < 0) {  
        System.out.println("Valore negativo!");  
        return 0;  
    }  
    return x;  
}
```

Gestione dinamica della memoria

Gestione dinamica della memoria: operatore *new* e *garbage collector*.



Passaggio con copia del valore o del riferimento

Ogni volta che si invoca un sottoprogramma viene eseguito il **passaggio dei parametri**.

Ci sono **due principali alternative**:

- passaggio con copia del valore,
- passaggio con copia del riferimento.

Passaggio con copia del valore: nel record di attivazione del sottoprogramma viene riservato uno spazio di memoria corrispondente al tipo del parametro o variabile. In seguito, in questo spazio viene **copiato il valore dell'argomento**.

Passaggio con copia del riferimento: nel record di attivazione del sottoprogramma viene riservato lo spazio per un riferimento (indirizzo di memoria). In seguito, in questo spazio viene **copiato l'indirizzo della zona di memoria dove si trova l'argomento**.

Passaggio con copia del valore o del riferimento

In Java:

- i tipi di dato **primitivi** vengono sempre passati a procedure e funzioni **con copia del valore**,
- i tipi di dato **riferimento (arrays e oggetti)** vengono sempre passati **con copia del riferimento**.

Attenzione: in Java non è possibile fare un “pass by reference” come è possibile fare in C++. In Java viene sempre eseguita una copia (del valore o del riferimento).

Ricorsione

Ogni **procedura o funzione**, al suo interno, è in grado di richiamare altre procedure o funzioni, che a loro volta possono chiamarne di ulteriori.

Un caso particolare di queste chiamate si ha quando la procedura o funzione richiama se stessa.

Si parla di procedura o funzione ricorsiva, e quindi di ricorsione.

Ricorsione

Nella ricorsione, un algoritmo si auto utilizza ed **esegue le medesime operazioni “in loop”**.

Quindi, ogni chiamata a procedura o funzione della ricorsione è **simile ad un’iterazione d’istruzione di ripetizione** (for, while, do ... while). Però, la formulazione dell’algoritmo è differente.

Anche il comportamento differisce, in particolare per quanto concerne l’occupazione della memoria.

Tipicamente, la ricorsione produce delle progressioni di risultati. Invocazioni vicine producono risultati simili.



Ricorsione

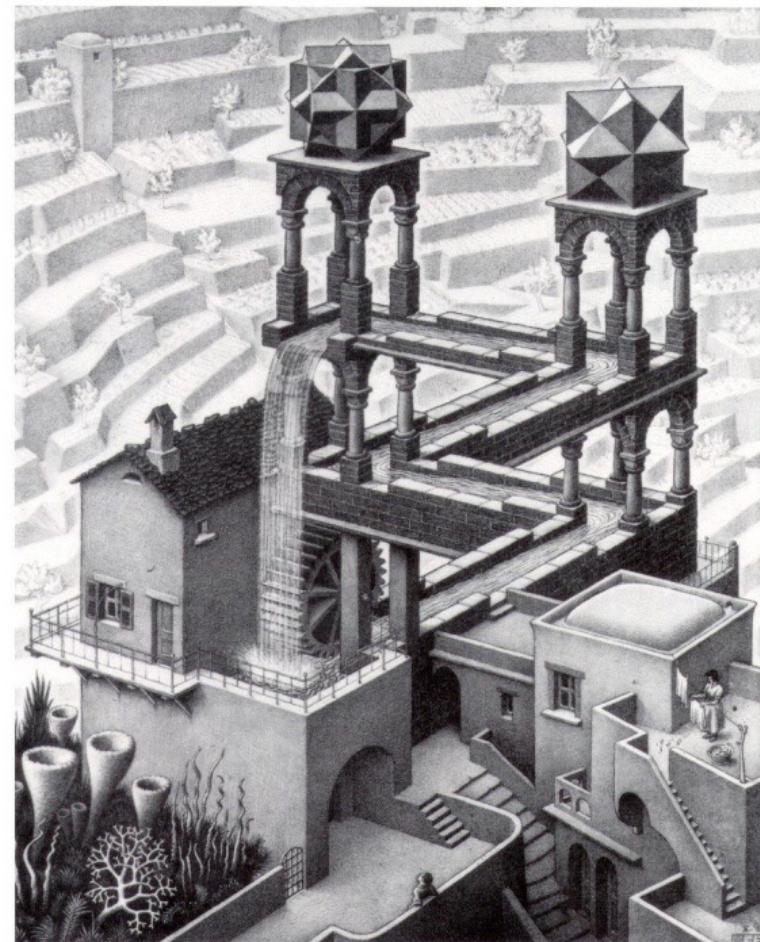
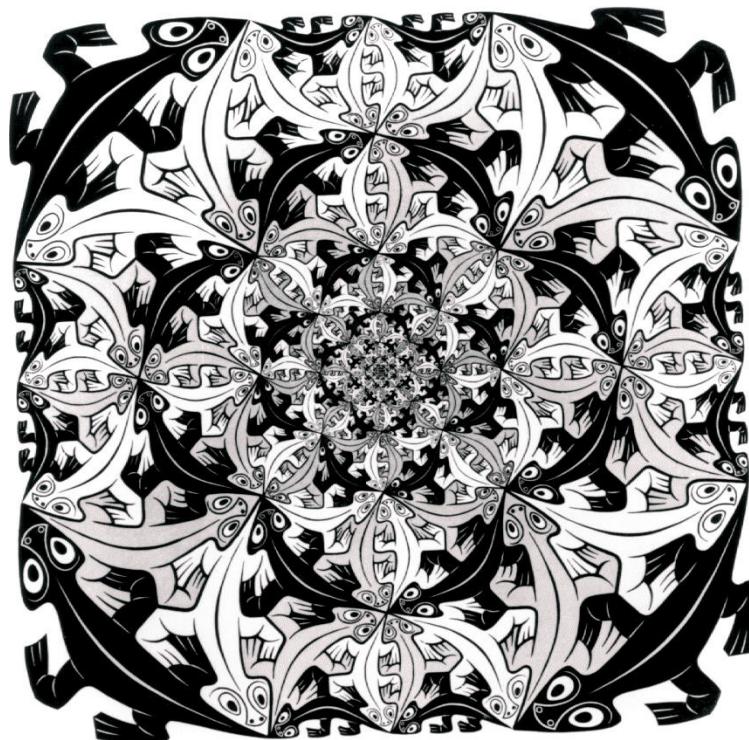
Possiamo osservare la ricorsione nella natura:





Ricorsione

Alcuni artisti si sono ispirati a processi ricorsivi:



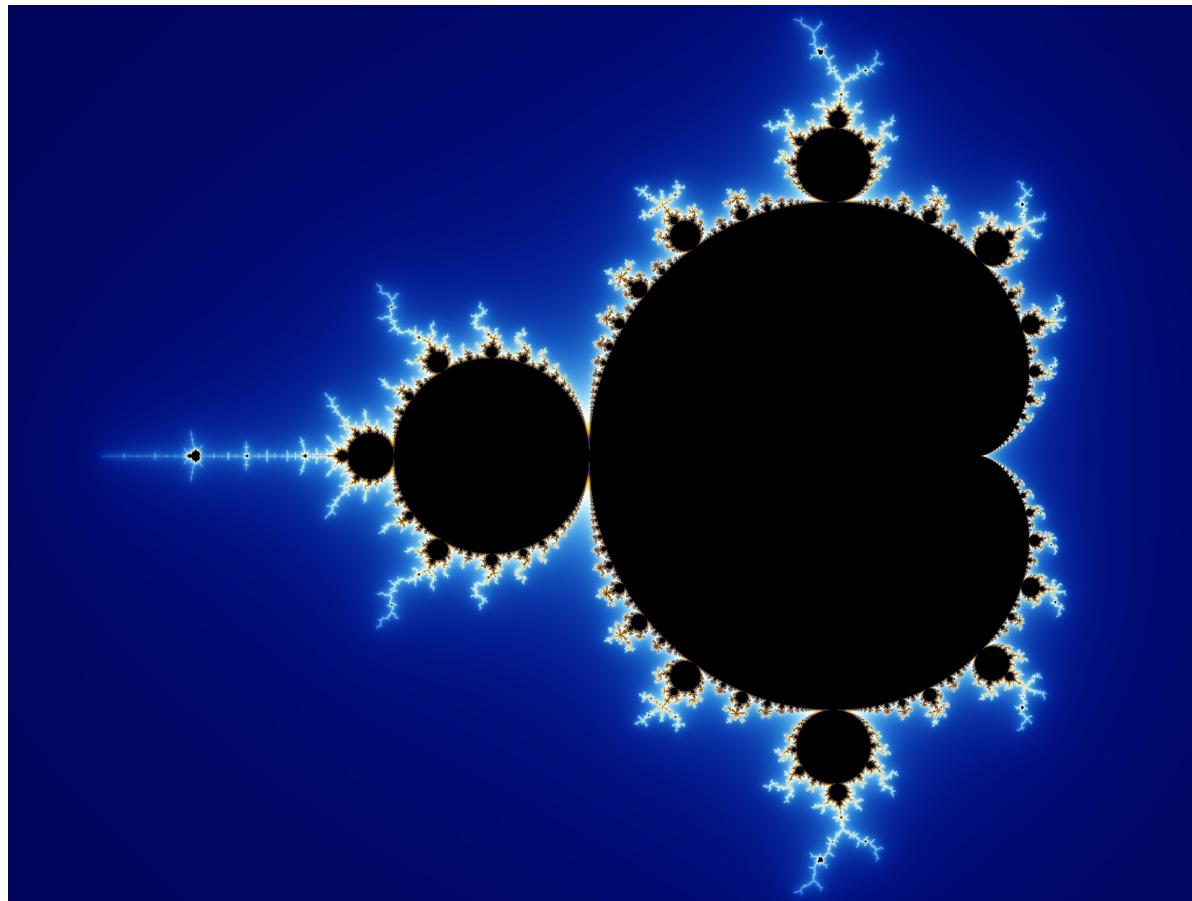
Maurits Cornelis Escher (1898 – 1972). Dutch graphic artist.

Often mathematically inspired. Impossible constructions and explorations of infinity.



L'insieme di Mandelbrot

Benoît Mandelbrot (1924 – 2010) è stato un matematico polacco naturalizzato francese, noto per i suoi lavori sulla geometria frattale.





I frattali

“Un frattale è un oggetto geometrico dotato di omotetia interna: si ripete nella sua forma allo stesso modo su scale diverse, e dunque ingrandendo una qualunque sua parte si ottiene una figura simile all’originale.”

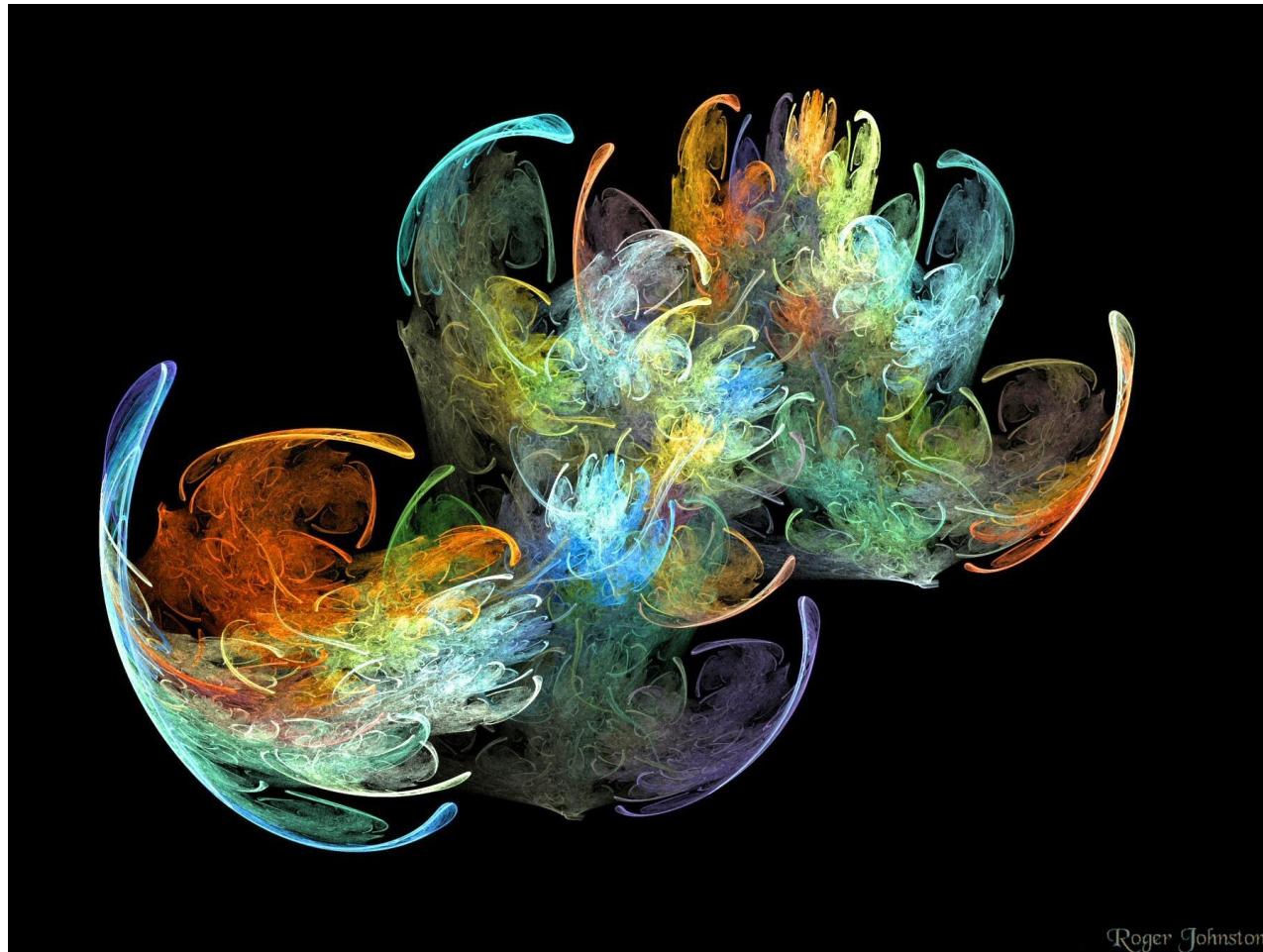
— Wikipedia

I frattali sono **patterns auto-simili**. Con auto-simile si intende che sono simili da vicino e da distante. I frattali possono essere esattamente identici ad ogni scala di grandezza, oppure molto simili a differenti scale. I frattali sono **auto-simili all’infinito, iterativamente**.

I frattali non si limitano ai patterns geometrici, possono descrivere anche processi nel tempo. Patterns frattali di differente livello e auto-similitudine sono stati osservati e studiati nelle immagini, nel suono, nella natura, nella tecnologia, nell’arte, ...

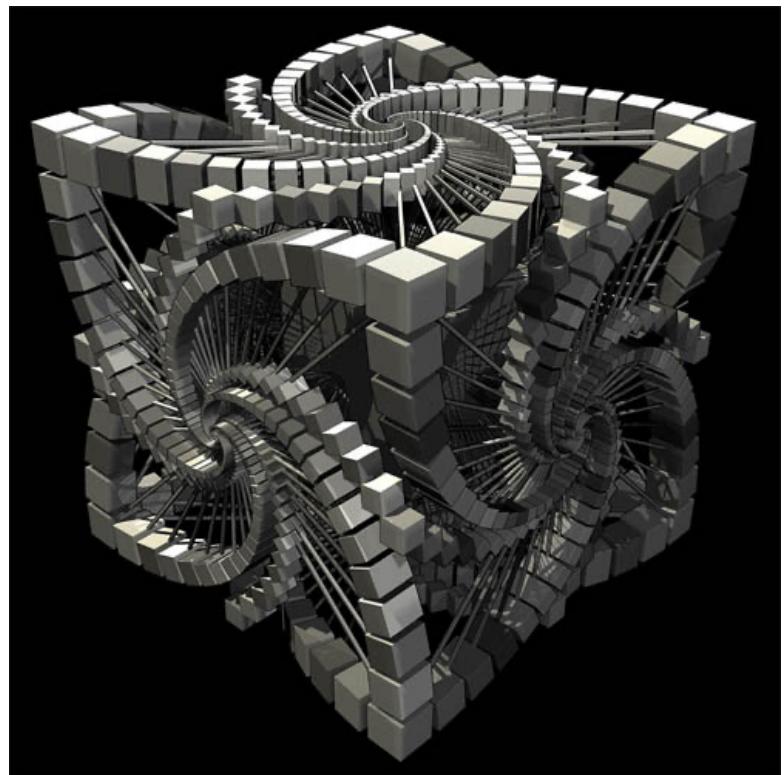
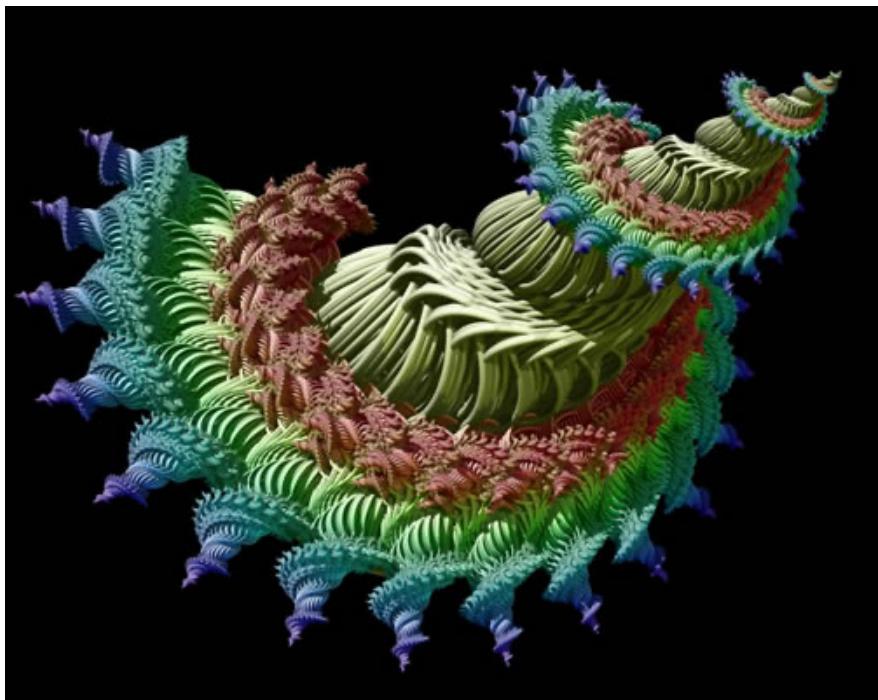


I frattali

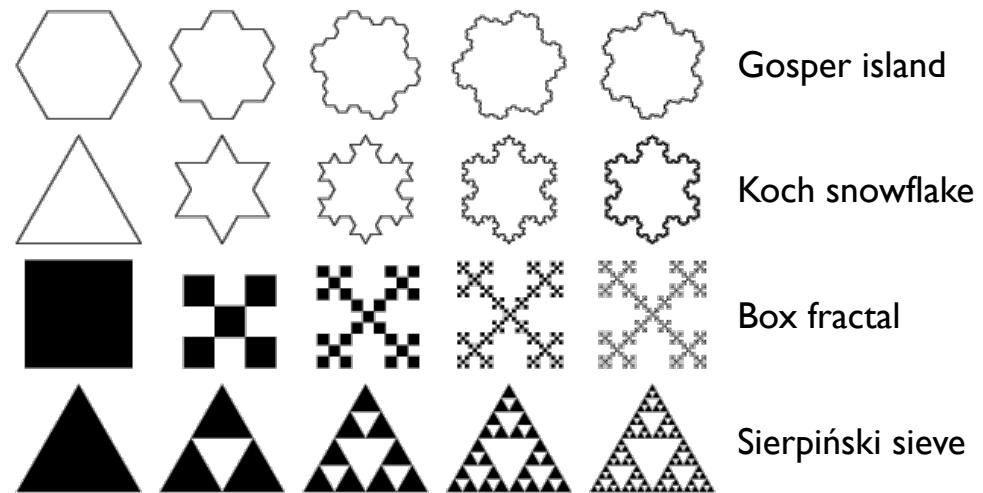
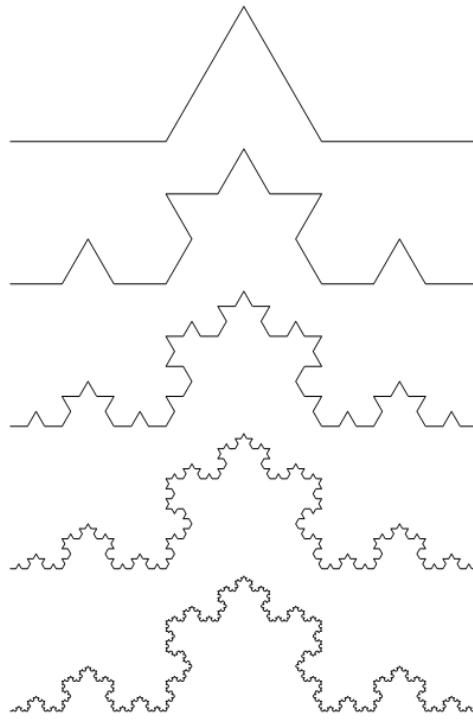




I frattali



Come funziona la ricorsione?



Come si utilizza nell'ingegneria? L'idea è di **scomporre il problema in parti più semplici** fino a che non si deve risolvere un problema molto semplice (divide et impera).

La ricorsione in Java

In Java è possibile utilizzare la ricorsione tramite chiamate ricorsive a procedure o funzioni.

Una procedura o funzione ricorsiva è una procedura o funzione che **chiama se stessa**.

La ricorsione in Java

L'idea è quella di scomporre il problema in parti più semplici fino a quando si arriva a risolvere un problema molto semplice.

Requisiti:

- **condizione di entrata**: lo stato in cui si trova la procedura o funzione al momento della chiamata.
- **chiamata ricorsiva**: all'interno della procedura o funzione è presente una o più chiamate a se stessa.
- **condizione di uscita**: necessaria per interrompere la ricorsione.
Fondamentale per evitare una ricorsione infinita!



Esempio: i numeri naturali

Definizione ricorsiva di un numero naturale:

$$\begin{cases} 0 & \in \mathbb{N} \\ n_k = n_{k-1} + 1 & n_k \in \mathbb{N} \text{ se } n_{k-1} \in \mathbb{N} \end{cases}$$



Esempio: l'elevamento a potenza

Definizione di elevamento a potenza:

$$a^n = \underbrace{a * a * \cdots * a}_{n \text{ volte}}$$

Definizione ricorsiva di elevamento a potenza:

$$a^n = \begin{cases} 1 & \text{se } n = 0 \\ a * a^{n-1} & \text{se } n > 0 \end{cases}$$

$$2^0 = 1$$

$$2^1 = 2 * 2^0 = 2 * 1 = 2$$

$$2^2 = 2 * 2^1 = 2 * 2 * 1 = 4$$

$$2^3 = 2 * 2^2 = 2 * 2 * 2 * 1 = 8$$

$$a^n = a * a^{n-1} = a * a * a^{n-2} = \underbrace{a * \cdots * a}_{n \text{ volte}}$$



Esempio: l'elevamento a potenza iterativo

```
public class ElevamentoPotenza {  
    private static long elevamentoPotenza(int base, int esponente) {  
        long risultato = 1;  
        for (int i = 0; i < esponente; i++)  
            risultato *= base;  
  
        return risultato;  
    }  
  
    public static void main(String[] args) {  
        int base = 3;  
        int esponente = 4;  
        System.out.println(base + " ^ " + esponente + " = " +  
                           elevamentoPotenza(base, esponente));  
    }  
}
```



Esempio: l'elevamento a potenza ricorsivo

```
public class ElevamentoPotenza {  
    private static long elevamentoPotenza(int base, int esponente) {  
        if (esponente == 0)  
            return 1;  
  
        return base * elevamentoPotenza(base, esponente - 1);  
    }  
  
    public static void main(String[] args) {  
        int base = 3;  
        int esponente = 4;  
        System.out.println(base + "^" + esponente + " = " +  
                           elevamentoPotenza(base, esponente));  
    }  
}
```

Esempio: il fattoriale

Definizione di fattoriale:

$$n! = \prod_{k=1}^n k = 1 * 2 * \dots * (n - 2) * (n - 1) * n$$

Definizione ricorsiva di fattoriale:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1)! & \text{se } n > 0 \end{cases}$$

$$0! = 1$$

$$1! = 1 * 0! = 1 * 1 = 1$$

$$2! = 2 * 1! = 2 * 1 * 1 = 2$$

$$3! = 3 * 2! = 3 * 2 * 1 * 1 = 6$$

$$n! = n * (n - 1)! = n * (n - 1) * (n - 2) * \dots * 2 * 1 * 1$$

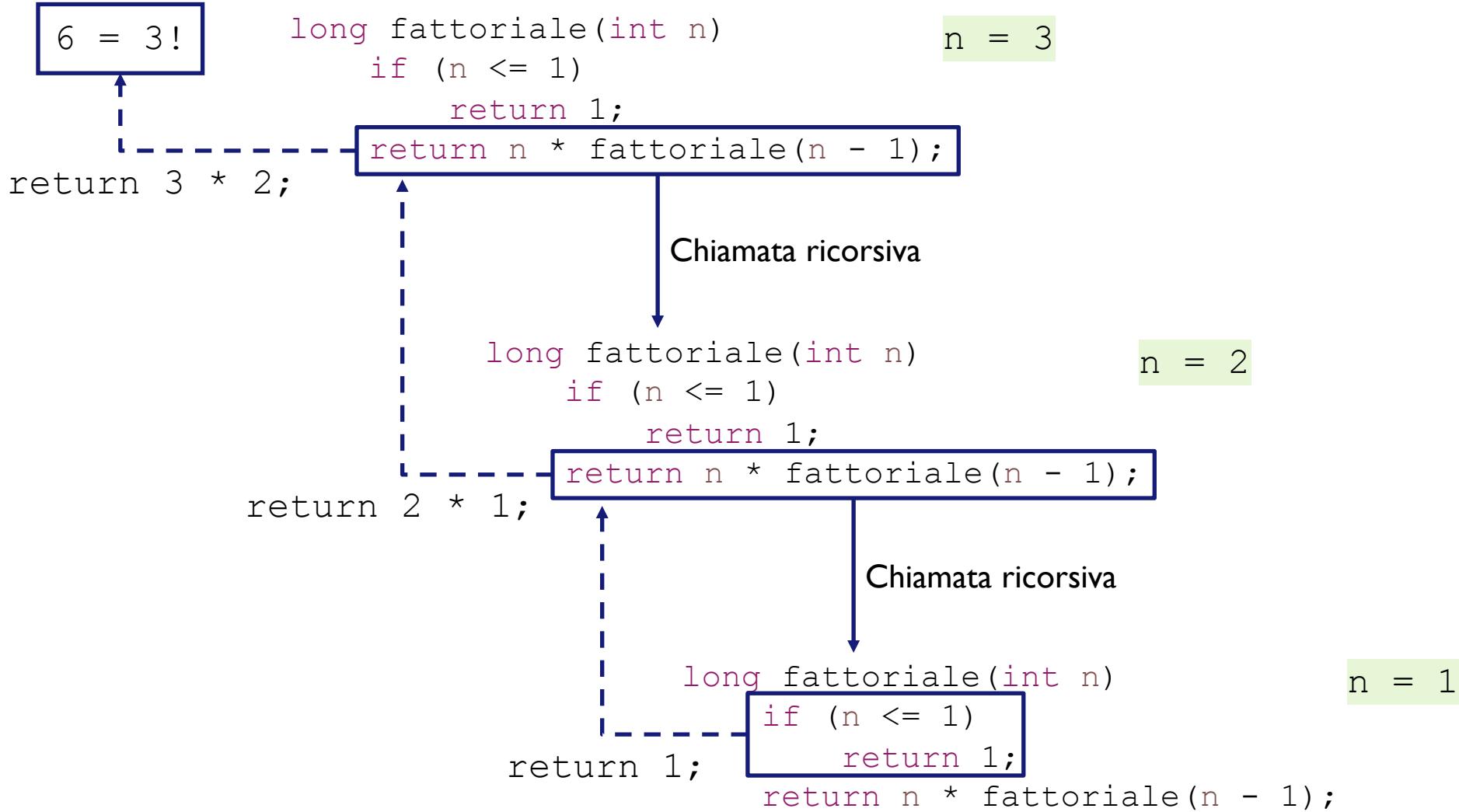
Esempio: il fattoriale iterativo

```
public class Fattoriale {  
    private static long fattoriale(int n) {  
        long fattoriale = 1;  
        for (int i = 2; i <= n; i++)  
            fattoriale *= i;  
  
        return fattoriale;  
    }  
  
    public static void main(String[] args) {  
        int num = 3;  
        System.out.println(num + " ! = " + fattoriale(num));  
    }  
}
```

Esempio: il fattoriale ricorsivo

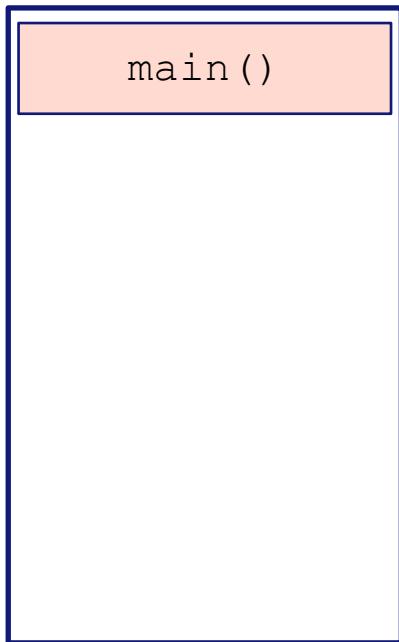
```
public class Fattoriale {  
    private static long fattoriale(int n) {  
        if (n <= 1)  
            return 1;  
  
        return n * fattoriale(n - 1);  
    }  
  
    public static void main(String[] args) {  
        int num = 3;  
        System.out.println(num + " ! = " + fattoriale(num));  
    }  
}
```

Esempio: il fattoriale

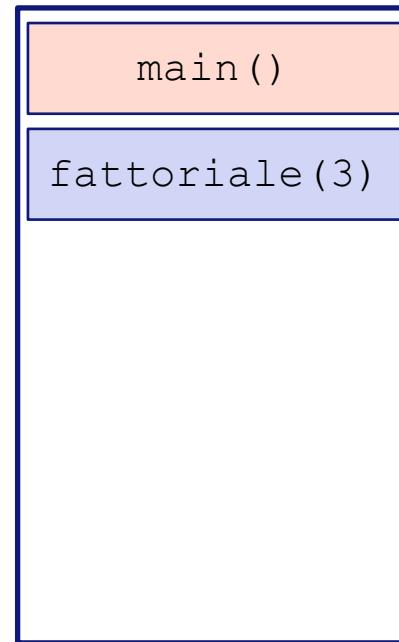


La ricorsione e lo stack

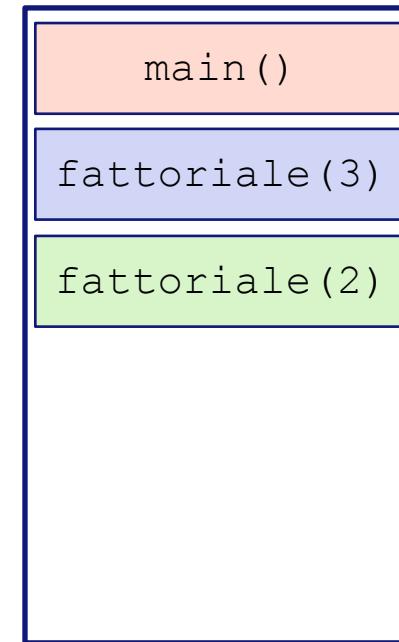
Situazione all'inizio
dell'esecuzione del
main()



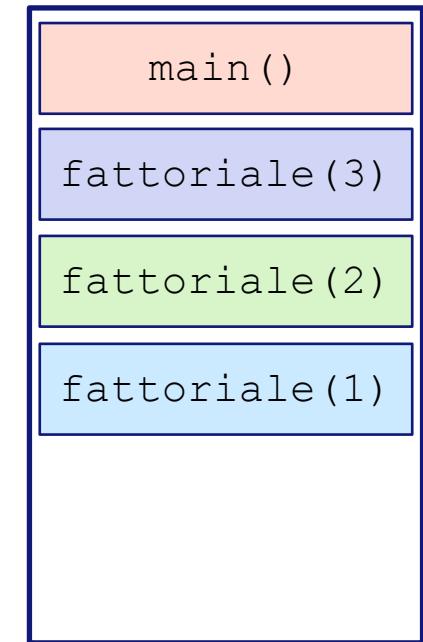
Il main() invoca
fattoriale(3)



fattoriale(3)
invoca
fattoriale(2)



fattoriale(2)
invoca
fattoriale(1)

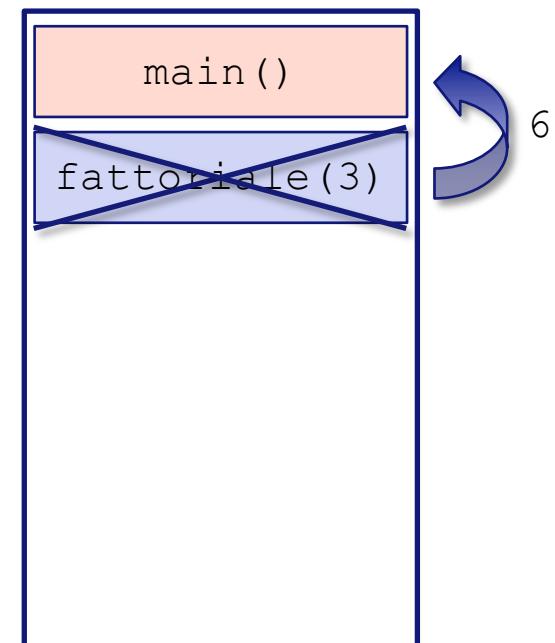
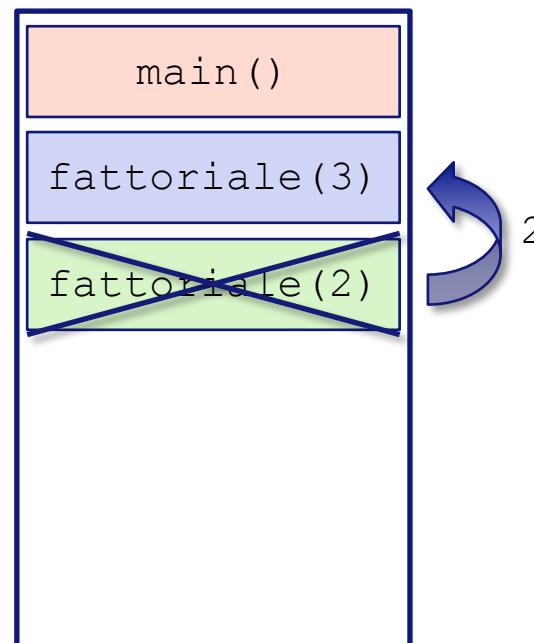
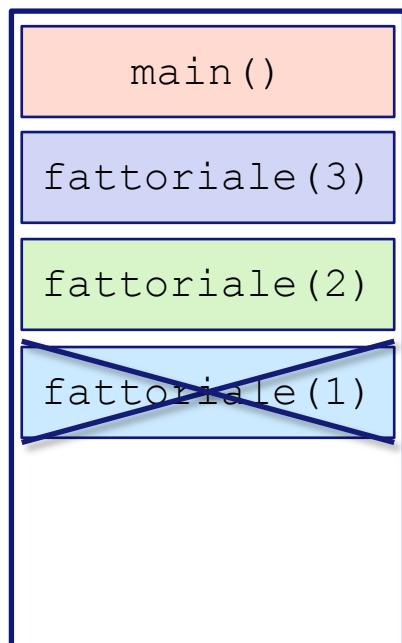


La ricorsione e lo stack

fattoriale(1)
termina restituendo
il valore 1. Il
controllo torna a
fattoriale(2)

fattoriale(2)
effettua la moltiplicazione e termina
restituendo il valore
2. Il controllo torna
a fattoriale(3)

fattoriale(3)
effettua la moltiplicazione e termina
restituendo il valore
6. Il controllo torna
a main()



La ricorsione infinita

Se manca la condizione d'uscita o se ad ogni passo ricorsivo la soluzione non si semplifica (errata gestione delle chiamate ricorsive), la procedura o funzione continua a richiamare se stessa all'infinito dando luogo ad una **ricorsione infinita**.

Il programma termina con l'errore **StackOverflowError** poiché esaurisce la memoria disponibile per tenere traccia delle chiamate.

Svolgimento e riavvolgimento

La fase di **svolgimento** è quando la procedura o funzione ricorsiva continua a chiamare se stessa fino a che non incontra la condizione di terminazione.

Esempio:

```
fattoriale(4)
  4 * fattoriale(3)
    4 * 3 * fattoriale(2)
      4 * 3 * 2 * fattoriale(1)
```

Svolgimento e riavvolgimento

La fase di **riavvolgimento** è quando viene incontrata la condizione di terminazione e le chiamate annidate si chiudono.

Esempio:

```
4 * 3 * 2 * 1  
 4 * 3 * 2  
 4 * 6  
 24
```

Svolgimento e riavvolgimento

È possibile sfruttare la ricorsione sia durante la fase di svolgimento, sia durante quella di riavvolgimento.

```
public class Ricorsione {  
    private static void ricorsiva(int contatore) {  
        if (contatore == 0)  
            return;  
        System.out.println("Prima: " + contatore);  
        ricorsiva(--contatore);  
        System.out.println("Dopo: " + contatore);  
    }  
  
    public static void main(final String[] args) {  
        ricorsiva(5);  
    }  
}
```

Output:

```
Prima: 5  
Prima: 4  
Prima: 3  
Prima: 2  
Prima: 1  
Dopo: 0  
Dopo: 1  
Dopo: 2  
Dopo: 3  
Dopo: 4
```

Ricorsione, particolarità

Se la chiamata alla procedura o funzione avviene dopo aver eseguito tutte le altre operazioni, la ricorsione viene chiamata ***tail recursion***.

Una procedura o funzione con tail recursion può facilmente essere trasformata in una ***procedura o funzione iterativa***.

Invece, se la chiamata ricorsiva è la prima di tutte le operazioni, si parla di ***head recursion***.

Negli altri casi si parla di ***middle o multi recursion***.

Se la funzione X chiama la funzione Y e, rispettivamente, la funzione Y chiama la funzione X, si parla di ***mutual recursion*** o ***indirect recursion***.

Se la funzione richiama più volte se stessa, si parla di ***ricorsione multipla***. Va usata con estrema cautela poiché può portare a programmi molto inefficienti.

Ricorsione o iterazione

Tutto ciò che si risolve con una ricorsione si può anche fare con un'iterazione.

Spesso la ricorsione è più “elegante”.

Spesso l'iterazione è più efficiente.

“To iterate is human, to recurse is divine.”

— Laurence Peter Deutsch

Esempi di problemi risolvibili con ricorsione

I seguenti problemi possono essere risolti utilizzando la ricorsione:

- Riconoscimento di frasi palindrome: angolo bar a Bologna; assalir i mici mi rilassa.
- Successione di Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, ...
- Calcolo del massimo comun divisore:
 1. $a = |a|$, $b = |b|$
 2. Ordina a e b in modo tale che $a > b$
 3. Se $b = 0$ allora $\text{MCD}(a, b) = a$; altrimenti $\text{MCD}(a, b) = \text{MCD}(b, a \bmod b)$
- Il triangolo di Pascal o di Tartaglia:
$$\begin{array}{ccccccc} & & & 1 & & & \\ & & & 1 & 1 & & \\ & & & 1 & 2 & 1 & \\ & & & 1 & 3 & 3 & 1 \\ & & & 1 & 4 & 6 & 4 & 1 \\ & & & 1 & 5 & 10 & 10 & 5 & 1 \end{array}$$
- L'attraversamento di una struttura di dati dinamica (liste, alberi).

Pensare ricorsivamente: il caso delle palindrome

Problema: scoprire se una stringa è palindroma.

Iniziamo a pensare come possiamo semplificare il problema.

Quale di questi sembra essere ***l'approccio più corretto per semplificare il problema?***

- eliminiamo il primo carattere,
- eliminiamo l'ultimo carattere,
- eliminiamo il carattere centrale,
- eliminiamo il primo e l'ultimo carattere,
- dividiamo la stringa a metà.

Palindrome: l'approccio

Ricordiamo che dobbiamo ottenere un problema più semplice, ma identico al problema più complesso.

Una stringa è palindroma se:

- la prima e l'ultima lettera sono uguali (ignorando i segni di interpunkzione):
RADAR
- la stringa ottenuta eliminando la prima e l'ultima lettera è ancora palindroma: **ADA**

Palindrome: condizione d'uscita

La stringa palindroma più semplice è una **stringa di due caratteri uguali**, ma allora si applica ancora il test della pagina precedente (prima e ultima lettera uguale).

Oppure è una **stringa di un carattere**: per definizione è palindroma. Questo caso si presenta se la stringa ha un numero dispari di lettere.

Oppure è una **stringa vuota**. Questo caso si presenta se la stringa ha un numero pari di lettere.



Esempio: la torre di Hanoi

Rompicapo matematico composto da tre paletti ed un certo numero di dischi di grandezza decrescente, che possono essere infilati in uno qualsiasi dei paletti.

Il gioco inizia con tutti i dischi incolonnati su un paletto in ordine decrescente, in modo da formare un cono.

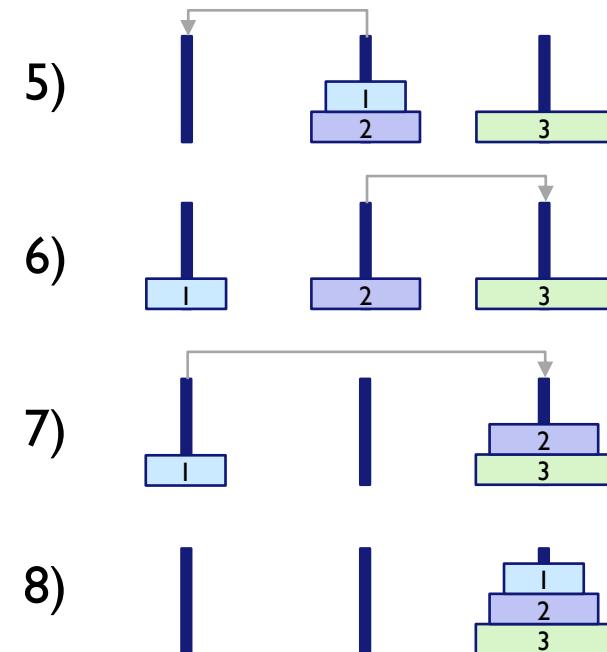
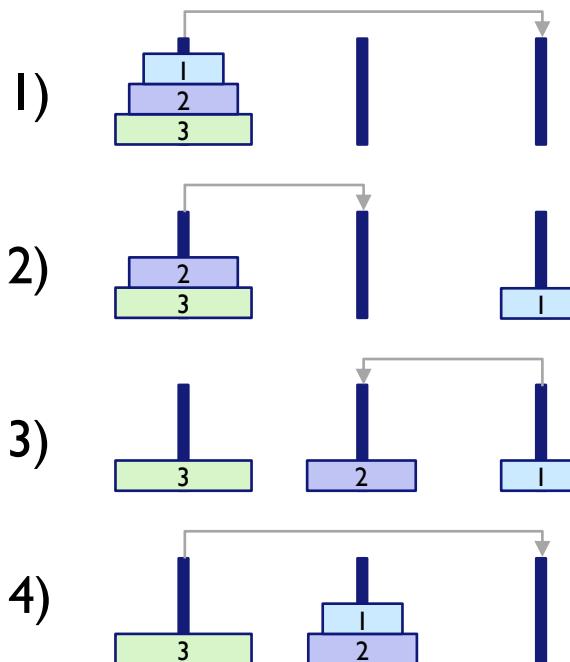
Lo scopo del gioco è portare tutti i dischi su un paletto diverso, potendo spostare solo un disco alla volta e potendo mettere un disco solo su un altro disco più grande, mai su uno più piccolo.





Esempio: la torre di Hanoi – 3 dischi

Soluzione per tre dischi:



Numero minimo di mosse necessarie: $7 (=2^3-1)$

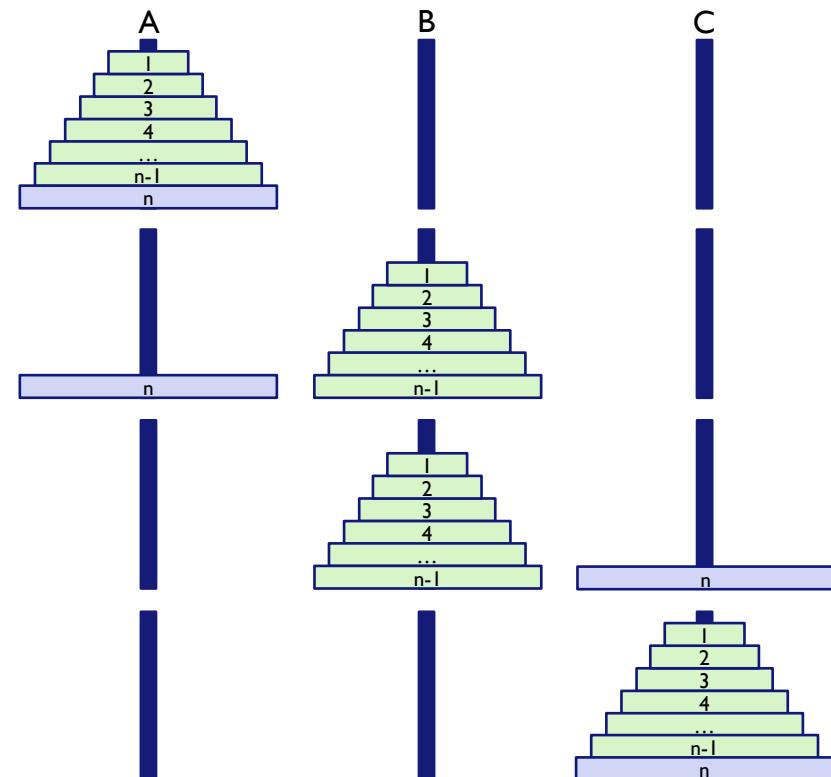


Esempio: la torre di Hanoi – n dischi

Il problema generale consiste nello spostare n dischi da un paletto (A) all'altro (C), usando il terzo paletto (B) come deposito temporaneo.

Soluzione ricorsiva:

1. Spostare i primi $n-1$ dischi da A a B.
2. Spostare il disco n da A a C.
3. Spostare $n-1$ dischi da B a C.





Esempio: la torre di Hanoi – n dischi

```
public class Hanoi {  
    private static void move(String start, String end) {  
        System.out.println(start + " -> " + end);  
    }  
  
    private static void hanoi(int n, String start, String aux, String end) {  
        if (n == 1) {  
            move(start, end);  
        } else {  
            hanoi(n - 1, start, end, aux);  
            hanoi(1, start, aux, end);  
            hanoi(n - 1, aux, start, end);  
        }  
    }  
  
    public static void main(String[] args) {  
        hanoi(3, "A", "B", "C");  
    }  
}
```

Animazione: <http://www.towersofhanoi.info/Animate.aspx>



Esempio: la torre di Hanoi – n dischi

```
hanoi(3, A, B, C)
```

```
    hanoi(2, A, C, B)
```

```
        hanoi(1, A, B, C)
```

```
            move(A, C)
```

```
        end
```

```
        hanoi(1, A, C, B)
```

```
            move(A, B)
```

```
        end
```

```
        hanoi(1, C, A, B)
```

```
            move(C, B)
```

```
        end
```

```
    end
```

```
hanoi(1, A, B, C)
```

```
    move(A, C)
```

```
end
```

...

...

```
hanoi(2, B, A, C)
```

```
    hanoi(1, B, C, A)
```

```
        move(B, A)
```

```
    end
```

```
    hanoi(1, B, A, C)
```

```
        move(B, C)
```

```
    end
```

```
    hanoi(1, A, B, C)
```

```
        move(A, C)
```

```
    end
```

```
end
```

Riepilogo

- Ricorsione
- Esempi di ricorsione: l'elevamento a potenza, il fattoriale, frasi palindrome
- La ricorsione e lo stack
- La ricorsione infinita
- Svolgimento e riavvolgimento
- Head, middle e tail recursion
- Ricorsione vs iterazione