Tree school

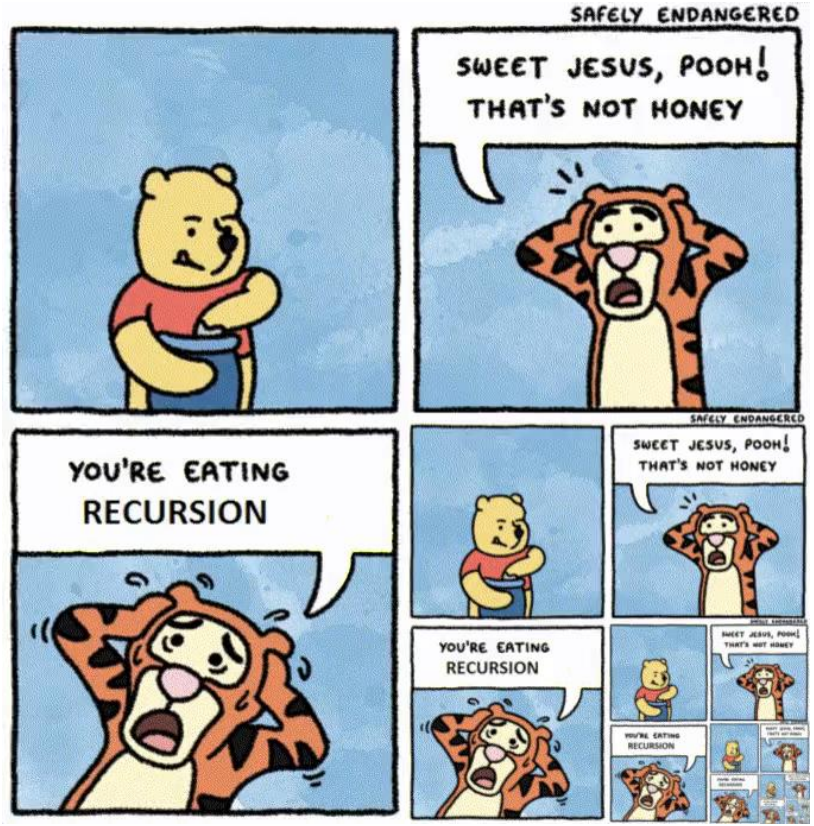
Modulo 2-3

ABC dell'informatica in Java!

Andrea Gasparini

Ricorsione

Ricorsione: come tutto ebbe inizio



Definizione di ricorsione da Wikipedia

In informatica viene detto algoritmo ricorsivo un algoritmo espresso in termini di se stesso, ovvero in cui l'esecuzione dell'algoritmo su un insieme di dati comporta la semplificazione o suddivisione dell'insieme di dati e l'applicazione dello stesso algoritmo agli insiemi di dati semplificati.

Il concetto di ricorsione

- **Il meccanismo di richiamo di una funzione supporta la possibilità, per un metodo, di richiamare sé stesso.**
- **Questa funzionalità è nota anche come ricorsione**
- **Per quanto sia un concetto base dell'informatica, alcuni linguaggi di programmazione possono non offrire questo supporto.**
- **Altri linguaggi invece, non offrono la possibilità di utilizzare cicli ma solo ricorsione.**

Come scrivere una funzione ricorsiva

Una funzione ricorsiva possiede due elementi:

Caso base:

Una condizione di input (nota come condizione di uscita o terminazione) che quando è rispettata restituisce un valore senza effettuare chiamate ricorsive.

Chiamata ricorsiva:

Il caso generico della funzione ricorsiva che per essere risolto ha bisogno di conoscere la risposta del proprio problema allo step precedente.

Esempio: successione di Fibonacci

Come è definita la successione di Fibonacci? (es: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …)

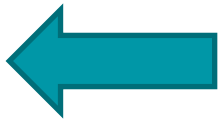
- **fibonacci(0) = 0**
- **fibonacci(1) = 1**
- **fibonacci(x) = fibonacci(x-1) + fibonacci(x-2) se x > 1**

E' una definizione ricorsiva!

Caso base

```
public int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Chiamate ricorsive



Esempio: fattoriale di un numero intero

Come si calcola il fattoriale n! di un intero n?

- **Definizione iterativa:**
  - **n! = n*(n-1)*(n-2)*...*2*1 n > 0**
  - **0! = 1 n = 0**

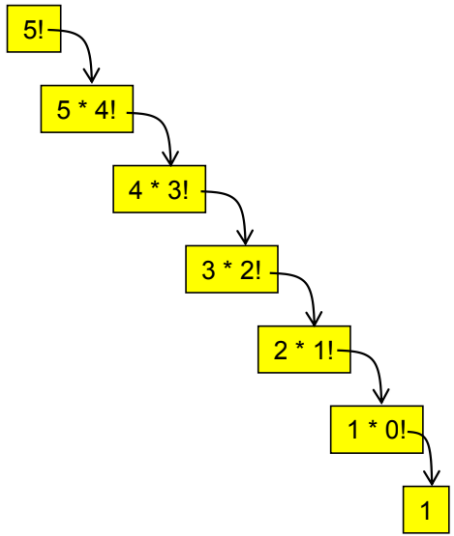
```
public int fattorialeIterativo(int n)
{
    int v = 1;
    for (int k = 0; k <= n; i++) v *= k;
    return v;
}
```

- **Definizione ricorsiva:**
  - **n! = n*(n-1)! n > 0**
  - **0! = 1 n = 0**

```
public int fattorialeRicorsivo(int n)
{
    if (n == 0) return 1;
    return n * fattorialeRicorsivo(n-1);
}
```

Esempio: fattoriale di un numero intero

Sequenza di chiamate ricorsive:



Esempio animato!

```
factorial( n ):
if n == 1:
    return 1
else:
    return n * factorial(n-1):
```

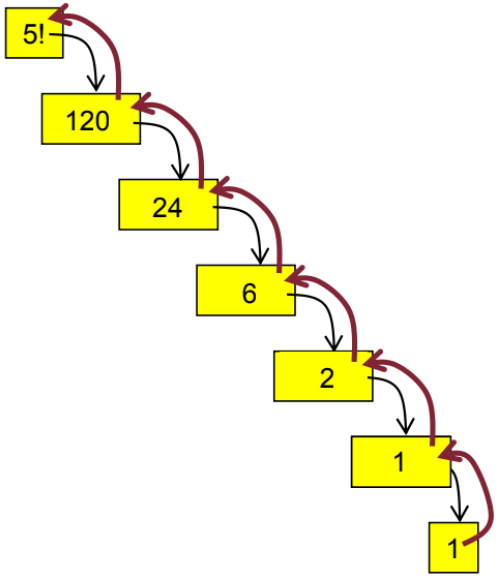
factorial(n) =

www.penjee.com

[Link]

Esempio: fattoriale di un numero intero

Valori restituiti da ciascuna chiamata:

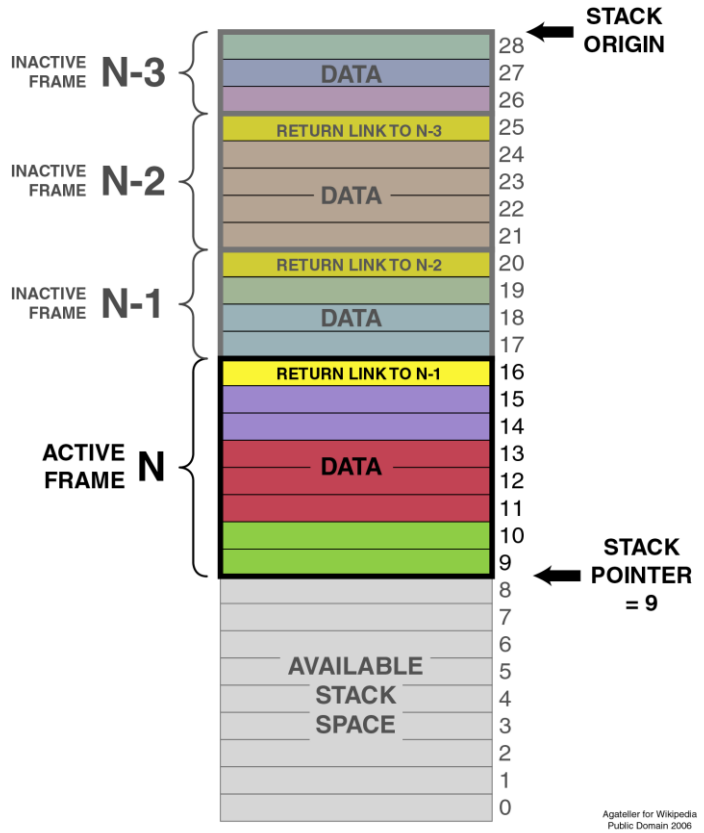


Attenzione!

È possibile avere più di un caso base e più di una chiamata ricorsiva, ovviamente dipende sempre dal problema che state risolvendo!

Dietro le quinte

- **Quando eseguiamo una chiamata a funzione, dietro le quinte il nostro programma va a creare un nuovo ambiente, detto record di attivazione**
- **I record di attivazione si trovano in una porzione di memoria limitata chiamata stack**
- **In ogni record di attivazione sono salvati: la zona di memoria per le variabili locali del metodo e l'indirizzo di ritorno al metodo chiamante**
- **Perciò ogni volta che chiamiamo una funzione consumiamo una parte di stack ma quando la funzione termina la recuperiamo**
- **Questa operazione ha un costo di overhead trascurabile... Ma lo stack è limitato, cosa succederebbe se chiamassimo funzioni senza mai uscirne?**



Stack Overflow

- **Se finisce lo stack otteniamo un errore chiamato Stack Overflow.**

- **È un errore comune quando si è inesperti con la ricorsione, ci siamo passati tutti ☺**

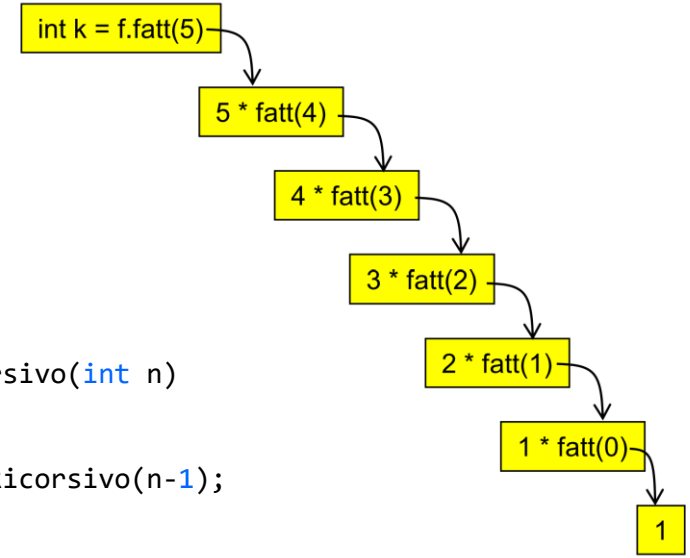
- **Capita soprattutto quando ci dimentichiamo il caso base o non ne avevamo considerato qualcuno.**

- **Purtroppo però, anche se il nostro programma è corretto, è possibile che superi il limite di chiamate ricorsive e che quindi termini in errore.**

```
Exception in thread "main" java.lang.StackOverflowError Create breakpoint
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
    at SommaArrayRicorsiva.stackOverflow(SommaArrayRicorsiva.java:24)
```

Esempio: fattoriale di un numero intero

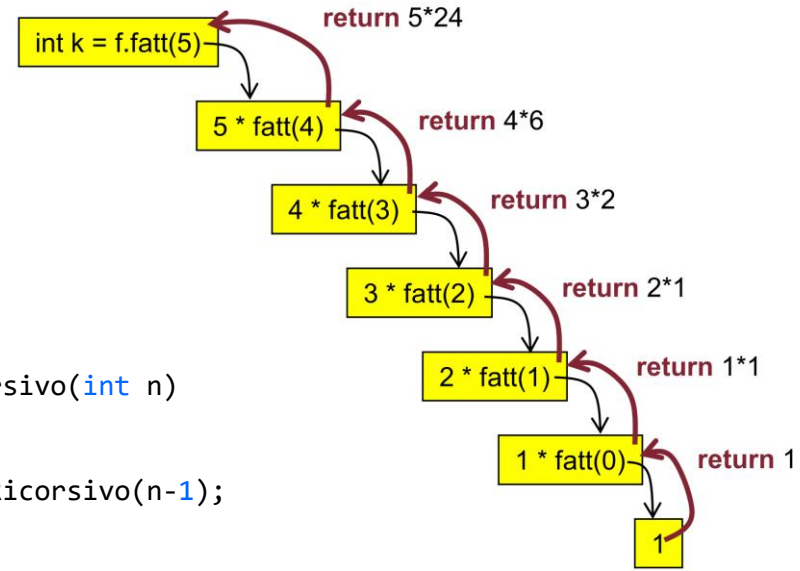
Che succede quando chiamiamo `fattorialeRicorsivo(5)`**?**




```
public int fattorialeRicorsivo(int n)
{
    if (n == 0) return 1;
    return n * fattorialeRicorsivo(n-1);
}
```

Esempio: fattoriale di un numero intero

Che succede quando chiamiamo `fattorialeRicorsivo(5)`**?**




```
public int fattorialeRicorsivo(int n)
{
    if (n == 0) return 1;
    return n * fattorialeRicorsivo(n-1);
}
```

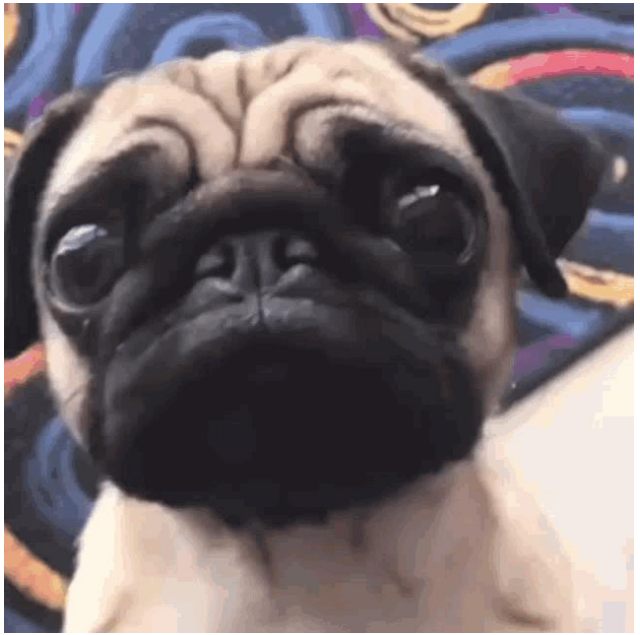
fatt(0)
n = 0 chiamante = Fattoriale.fatt
fatt(1)
n = 1 chiamante = Fattoriale.fatt
fatt(2)
n = 2 chiamante = Fattoriale.fatt
fatt(3)
n = 3 chiamante = Fattoriale.fatt
fatt(4)
n = 4 chiamante = Fattoriale.fatt
fatt(5)
n = 5 chiamante = Fattoriale.main
main(new String[] {})
args = new String[] {} Fattoriale f = new Fattoriale() chiamante = JVM

Ricorsione vs iterazione

- **La ricorsione permette di semplificare il codice e renderlo più pulito e leggibile e soprattutto molto più corto.**
- **Bisogna però tenere sempre conto che può fallire in stack overflow se esegue troppe chiamate ricorsive**
- **Spesso l'approccio iterativo ha performance migliori (non dovendo ogni volta scrivere sullo stack per ogni step)**
- **Se un problema è risolvibile con la ricorsione, allora è risolvibile anche iterativamente e viceversa**

Ma allora a cosa serve la ricorsione?

- **La ricorsione ha l'enorme vantaggio di semplificare codici che altrimenti sarebbero molto complicati da scrivere**
- **Se siete sicuri che la vostra funzione non vada troppo in profondità con i vostri dati, usate la ricorsione se vi fa comodo.**
- **Quello a destra siete voi il giorno che dovrete navigare un grafo senza ricorsione...**

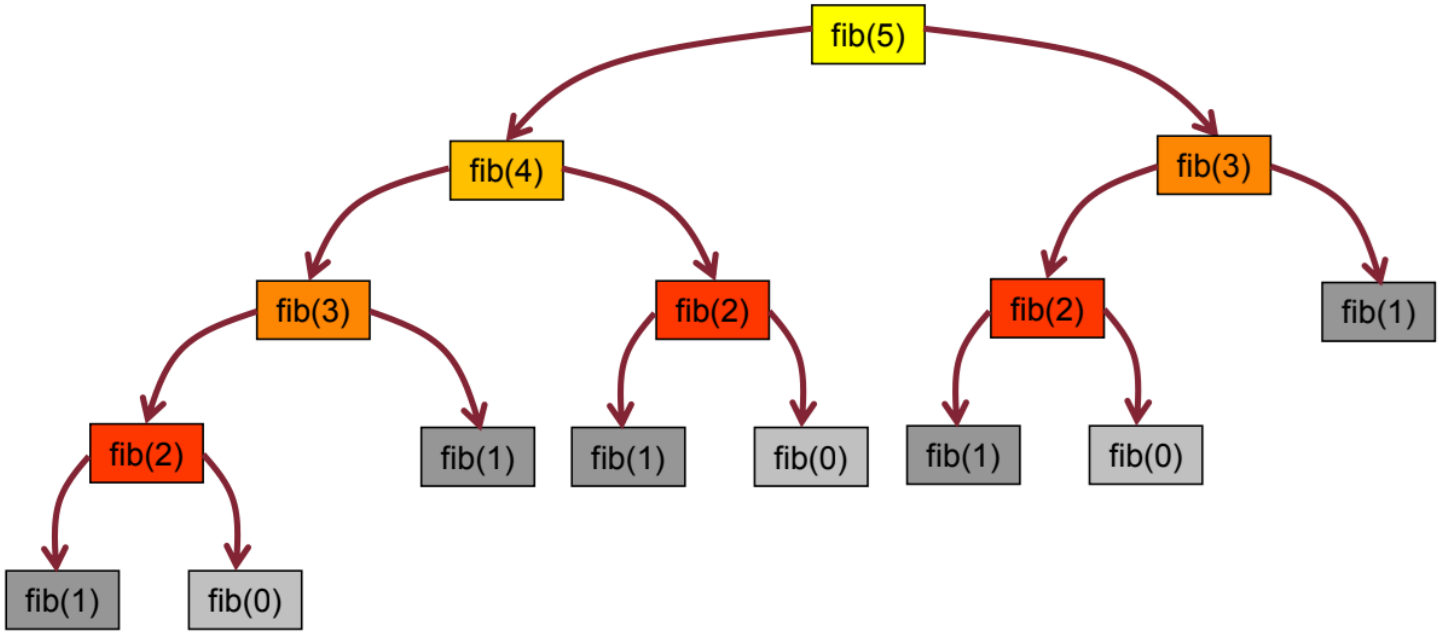


E non finisce qui!

- **Fate anche attenzione a non calcolare più volte la stessa cosa con la ricorsione.**
- **Secondo voi quale è la classe di complessità di questo codice e cosa c'è che non va?**

```
public static int fib(int n)
{
    if (n == 0 || n == 1) return n;
    return fib(n - 1) + fib(n - 2);
}
```

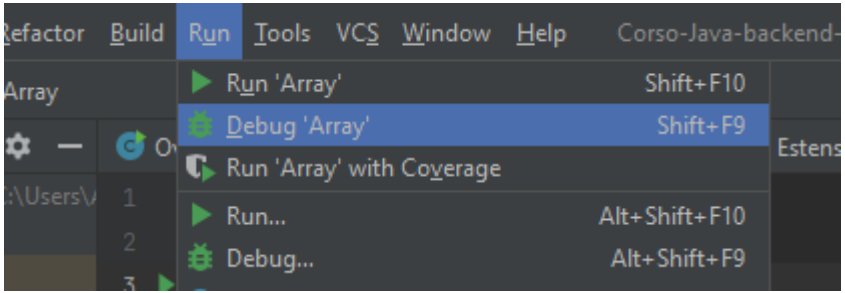
Schema delle invocazioni del metodo ricorsivo *fib*

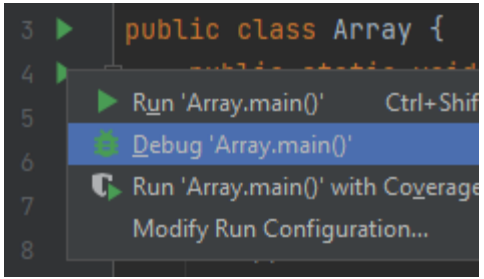


Debugger

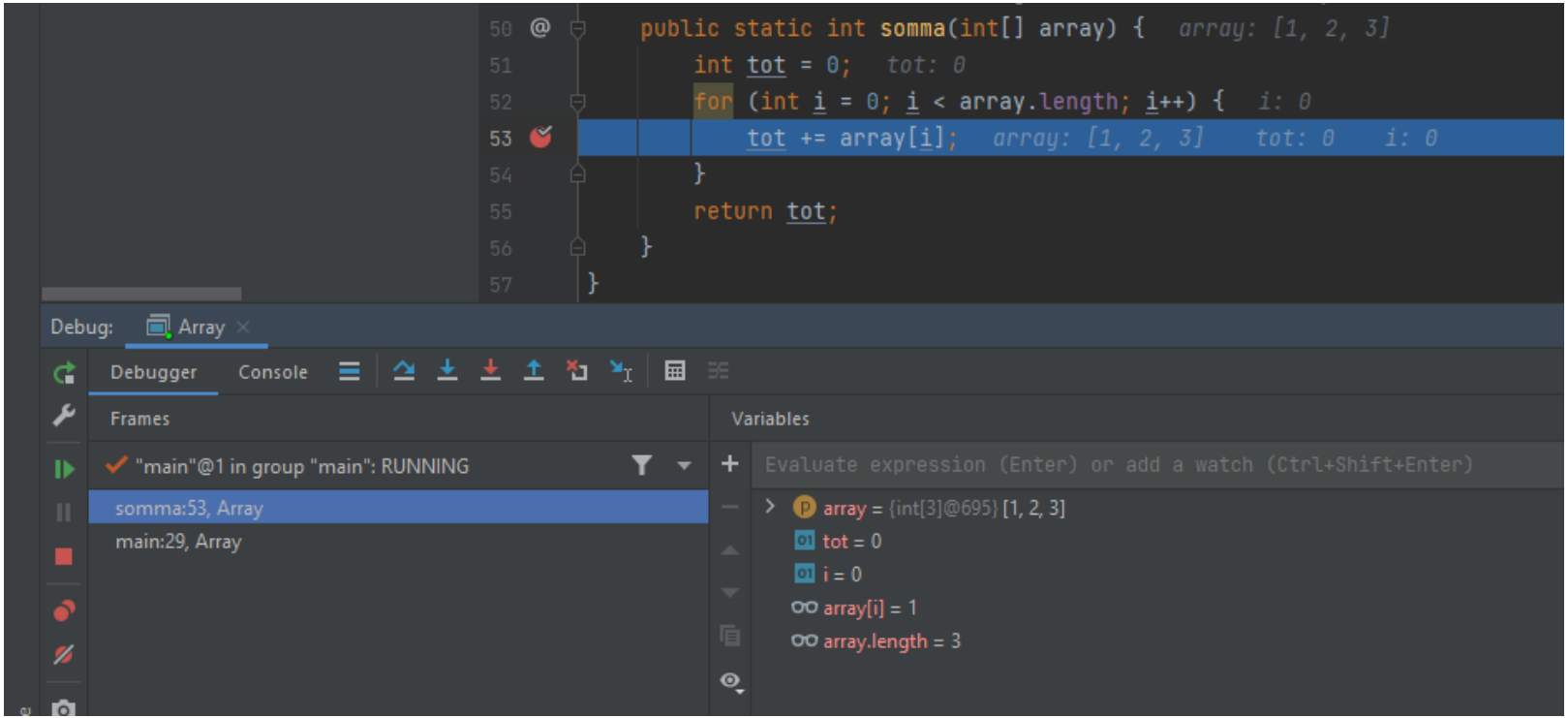
Debugger

- **Programma specificatamente progettato per l'analisi e l'eliminazione dei bug (debugging), ovvero errori di programmazione interni al codice di altri programmi**
- **Assieme al compilatore è fra i più importanti strumenti di sviluppo a disposizione di un programmatore, spesso compreso all'interno di un ambiente integrato di sviluppo (IDE)**
- **Per eseguire il programma in modalità debug, cliccare su Run e poi Debug**





Debugger


50 @ public static int somma(int[] array) { array: [1, 2, 3]
51 int tot = 0; tot: 0
52 for (int i = 0; i < array.length; i++) { i: 0
53 tot += array[i]; array: [1, 2, 3] tot: 0 i: 0
54 }
55 return tot;
56 }
57 }
Debug: Array
Debugger
Console
Frames
Variables
"main"@1 in group "main": RUNNING
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
somma:53, Array
main:29, Array
array = {int[3]@695} [1, 2, 3]
tot = 0
i = 0
array[i] = 1
array.length = 3

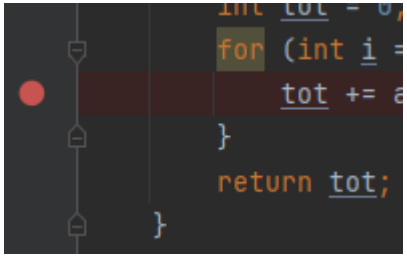
Debugger

Per utilizzare un debugger basta comprendere tre concetti basilari:

- **come impostare alcuni punti di arresto (breakpoint)**
- **come ispezionare il contenuto delle variabili**
- **come far proseguire il programma una riga alla volta (single step debug)**

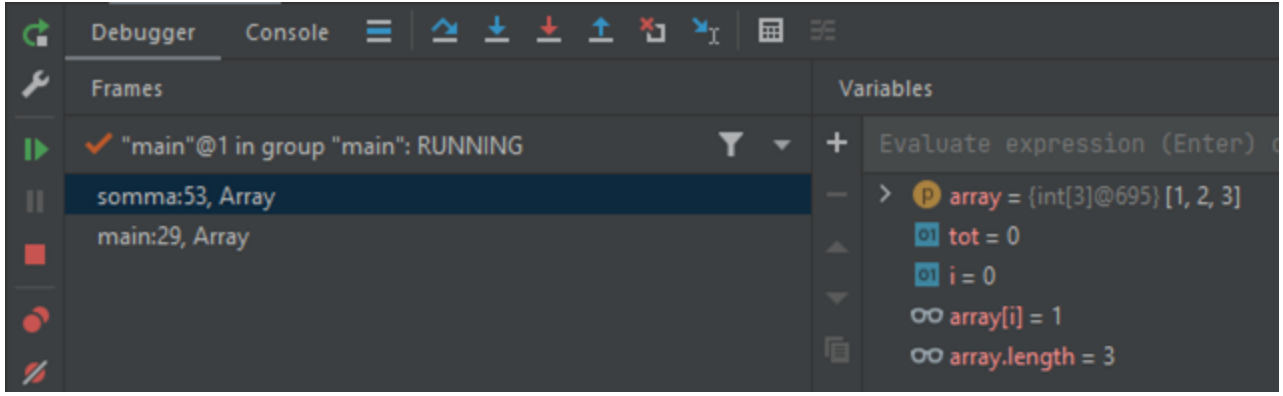
Breakpoint

- **I breakpoint si impostano con un semplice click col pulsante sinistro a lato della riga desiderata (vedrete comparire un pallino rosso su Intellij)**
- **E' sconsigliabile avere più comandi sulla stessa riga (altrimenti non sarà possibile associare breakpoint ad alcuni di essi)**
- **Con un click del pulsante destro sopra il pallino di un breakpoint potrete impostare delle condizioni di break avanzate sul punto di arresto**

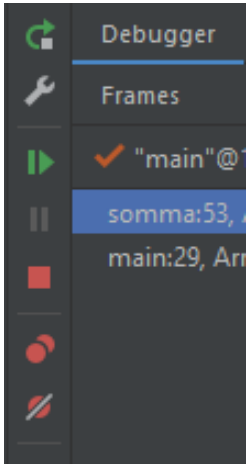


Prospettiva di debug

- **Quando viene raggiunto il primo breakpoint si aprirà la prospettiva di debug, che presenta molte viste interessanti**
- **La vista «Debugger» contiene la pila di esecuzione, dove compaiono tutte le istanze dei vari metodi attualmente sospesi (tipicamente quello in fondo alla pila sarà il main e quello in cima il metodo contenente il breakpoint raggiunto)**
- **La vista «Variables» mostra invece le variabili globali e locali del metodo e permette di ispezionarne il valore**



Step debug



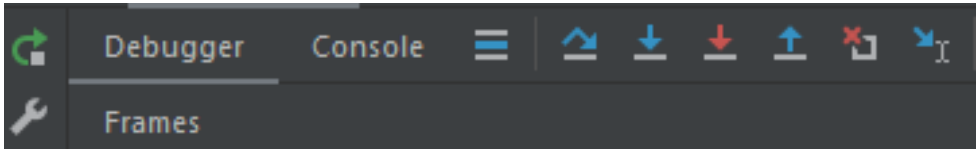
L'esecuzione può essere riattivata con modalità diverse associate alle icone che compaiono in cima alla vista di debug:

Il rettangolino verde permette di riprendere la normale esecuzione fino al prossimo breakpoint (o alla terminazione del programma)

Il quadratino rosso termina il programma

In aggiunta, è possibile eseguire altre azioni "più avanzate":

- **step-into, che riprende l'esecuzione del programma una riga alla volta e che in caso di invocazioni di metodi porta all'interno di essi**
- **step-over, che riprende l'esecuzione del programma una riga alla volta senza arrestarsi all'interno di eventuali metodi invocati**



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Bonus di oggi: valori randomici

Random value

- **Ottenere un valore decimale casuale in Java è possibile tramite il metodo statico *random()* della classe *Math***
- **Questo metodo restituisce un valore decimale maggiore o uguale di 0 e minore di 1**
- **Possiamo facilmente ottenere valori interi o booleani**

```java
// Obtain a number between [0.0 - 1.0)
double randomValue = Math.random();

// Obtain a number between [0 - 49]
int randomInt = (int) (randomValue * 50);

// Obtain a boolean value
boolean randomBool = randomValue > 0.5;
```

La classe *Random*

- **Il package *java.util* fornisce inoltre una classe *Random* per la generazione di valori casuali**
- **La classe *Random* consente di creare direttamente valori casuali di tipo *int*, *float*, *boolean* e *double***

```
import java.util.Random;

Random rand = new Random();

// Obtain a number between [0.0 - 1.0)
double randomDouble = rand.nextDouble();

// Obtain a number between [0 - 49]
int randomInt = rand.nextInt(50);

// Obtain a boolean value
boolean randomBool = rand.nextBoolean();
```

Random in un intervallo

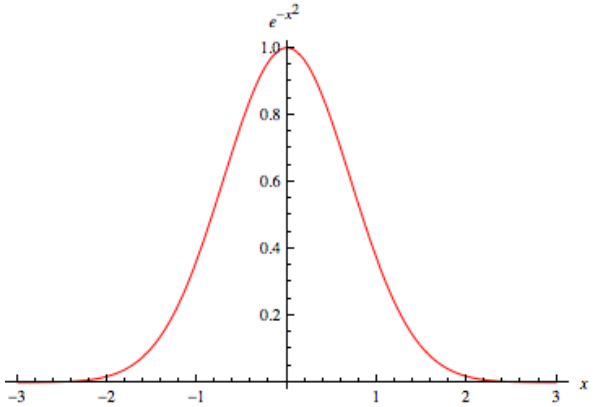
- **Se vi serve un valore casuale in uno specifico intervallo (per esempio compreso tra 34 e 41 per simulare la temperatura corporea di un uomo) potete sommare al valore base un numero casuale che rappresenta il range di distacco rispetto alla base**

```
int min = 34;
int max = 41;

// Obtain a number between [min - max]
int randomValueInRange = min + rand.nextInt(max - min + 1);
```

Random in un intervallo particolare 1/2

- **Riprendendo l'esempio della simulazione di temperature, sappiamo tutti che mediamente le persone hanno una temperatura intorno al 36,5 e pochissimi hanno una temperatura superiore al 40 o sotto il 35 (e meno male!)**
- **Il metodo *nextGaussian* della classe *Random* ci restituisce un valore casuale proveniente da una distribuzione gaussiana con valore medio 0 e standard deviation 1**
- **Tradotto in italiano, il 70% dei valori sarà nel range [-1, 1], Il 25% [1,2] o [-2, -1] e il restante 5% [-3,-2] e [2,3]**
- **Con la riga sotto, abbiamo un valore casuale nel range [34,40] con la maggior parte dei valori nel range [36,38]**



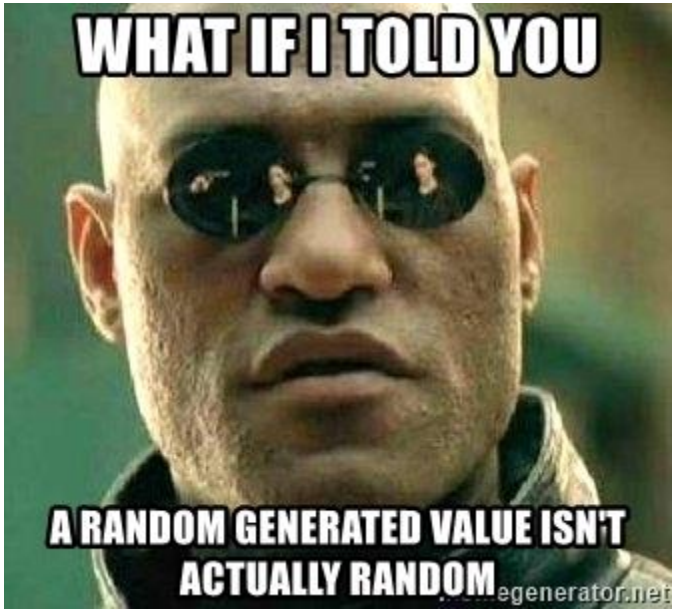

```
System.out.println(37 + rand.nextGaussian()));
```

Random in un intervallo particolare 2/2

- **Per avere invece un valore che può essere al 50% un valore, al 30% un altro e infine al 20% un altro ancora, vi basta prendere un numero random da 0 a 1 e controllare in che intervallo di valori si trova**

- **Un altro simpatico trucco per avere distribuzioni di valori sbilanciate è quello di riempire un array dei valori che volete generare inserendo ripetizioni per avere la vostra distribuzione sbilanciata (in maniera molto simile alle macchinette che distribuiscono pupazzi in cui alcuni pupazzi sono molto più rari di altri)**

Valori pseudo-casuali



Valori pseudo-casuali

- **I valori generati dai metodi visti fino ad ora sono detti pseudo-casuali in quanto sono prodotti da un algoritmo *deterministico* che genera sequenze approssimabili a dei valori casuali**
- **La sequenza di valori generati varia in base al *seed* specificato durante la creazione dell'oggetto *Random*, allo stesso seed corrispondono sempre gli stessi valori**
- **Se non viene specificato alcun seed la classe utilizza come riferimeto l'orologio di sistema, così che il seed sia sempre diverso e di conseguenza anche i valori generati**

```
Random rand = new Random(42);

int randomValue1 = rand.nextInt();
int randomValue2 = rand.nextInt();

// randomValue1 == randomValue2
```


Time to make practice

Esercizio Stack overflow

Scrivete una funzione ricorsiva che vada in stack overflow!

Trova i numeri primi

- **Un numero primo è un numero che è divisibile solo per se stesso e 1.**
- **Scrivete un programma che dato un numero N, scopre tutti i numeri primi fino a N.**
- **Provate a ottimizzare il programma per renderlo il più veloce possibile, ci sono diversi trick che potete fare per velocizzarlo!**

Esercizio Somma i valori di un array ricorsivamente

Dato un array di valori interi, restituire la somma dell'array calcolandola utilizzando la ricorsione.

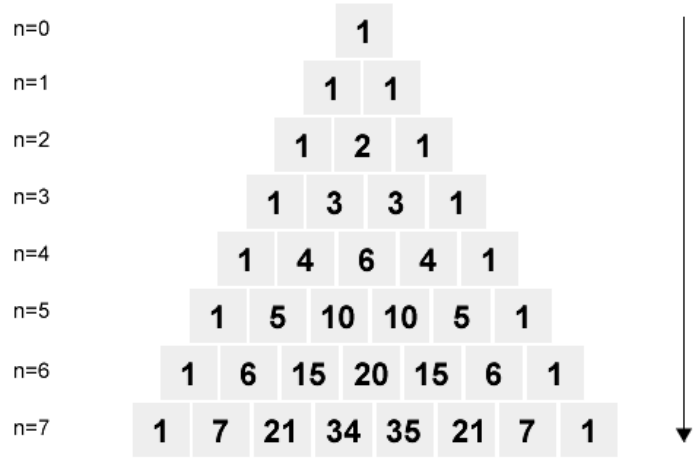
Hint: La classe Arrays di Java potrebbe avere qualcosa che vi serve...

Esercizio Stringa palindroma ricorsiva

Controllare se una stringa data in input è palindroma, ovvero se quando viene letta al contrario rimane invariata (ad esempio, “itopinonavevanonipoti” o “ailatiditalia”) utilizzando la ricorsione.

Esercizio Triangolo di tartaglia

Scrivere un programma (ovviamente ricorsivo) che dato un intero n restituisce l'array corrispondente alla n riga del triangolo di tartaglia.

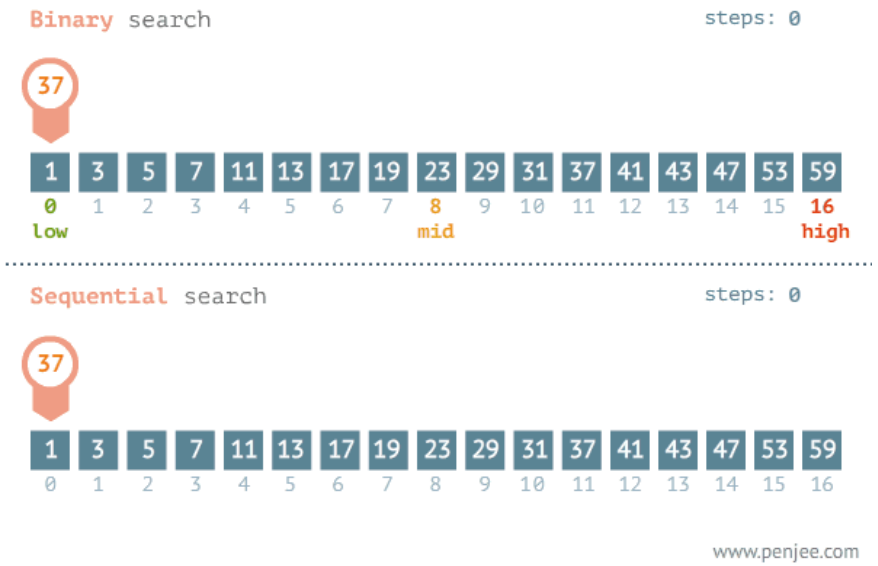


Trova i numeri perfetti

- **Un numero perfetto è un numero che è uguale alla somma dei suoi divisori escluso se stesso.**
- **Per esempio, il 6 è un numero perfetto perché i suoi divisori escluso se stesso sono 3,2 e 1 e sommandoli otteniamo di nuovo il 6**
- **Scrivete un programma che scopre i numeri perfetti fino al 1000.**

Esercizio Ricerca binaria

Scrivete un programma ricorsivo che implementi l'algoritmo della [ricerca binaria] per la ricerca di un elemento all'interno di un array ordinato di interi



Esercizio Mergesort

Scrivete un programma che implementi l'algoritmo di ordinamento del mergesort

6 5 3 1 8 7 2 4

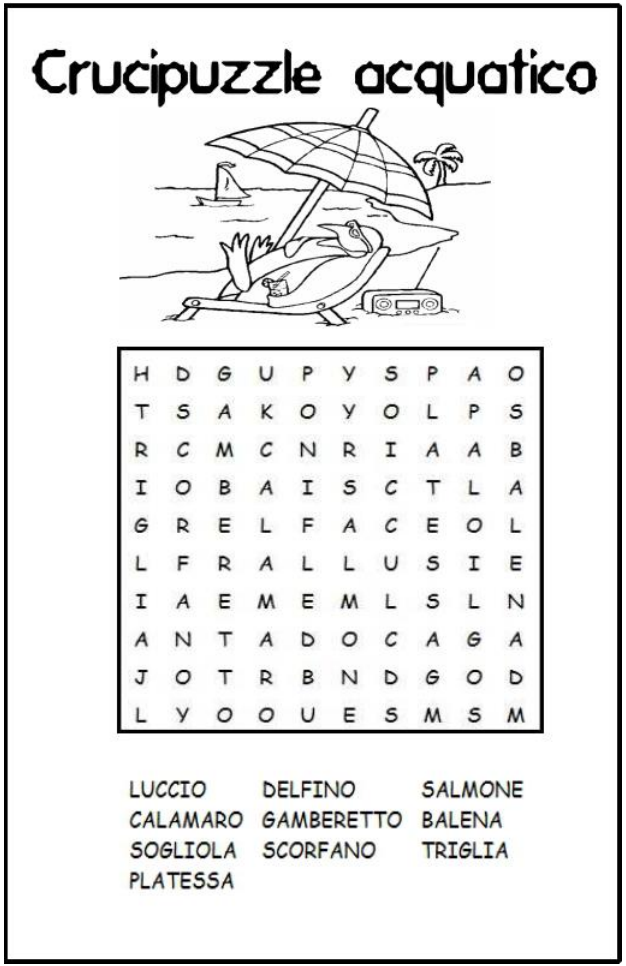
Tombola funzionante!

- **Ora che conoscete il random, potete completare il programma della tombola, facendo in modo che la cartellina venga generata randomicamente e sempre usando il random potete simulare l'estrazione.**
- **Questa volta la funzione che controlla la cartella non dovrà controllare tutti i numeri estratti ma soltanto uno per volta, dovrete pensare anche ad un modo per ricordarvi quali numeri sono usciti precedentemente.**

Cerca le parole

- **Scrivete una funzione che prende in input una matrice di caratteri e una parola e restituisce true se la parola si trova all'interno della matrice sia verticalmente che orizzontalmente, false altrimenti.**
- **Fatto questo fate in modo che il programma controlli anche se la parola è scritta al contrario**
- **Bonus stage: controllate anche se la parola è presente in diagonale!**

- **Suggerimenti: scrivetevi una funzione che stampa a schermo la matrice**
- **Quando trovate una parola stampatevi gli indici della prima lettera**



Experis ManpowerGroup an Opinno company