Metodologie di Programmazione

Lezione 25: Ricorsione

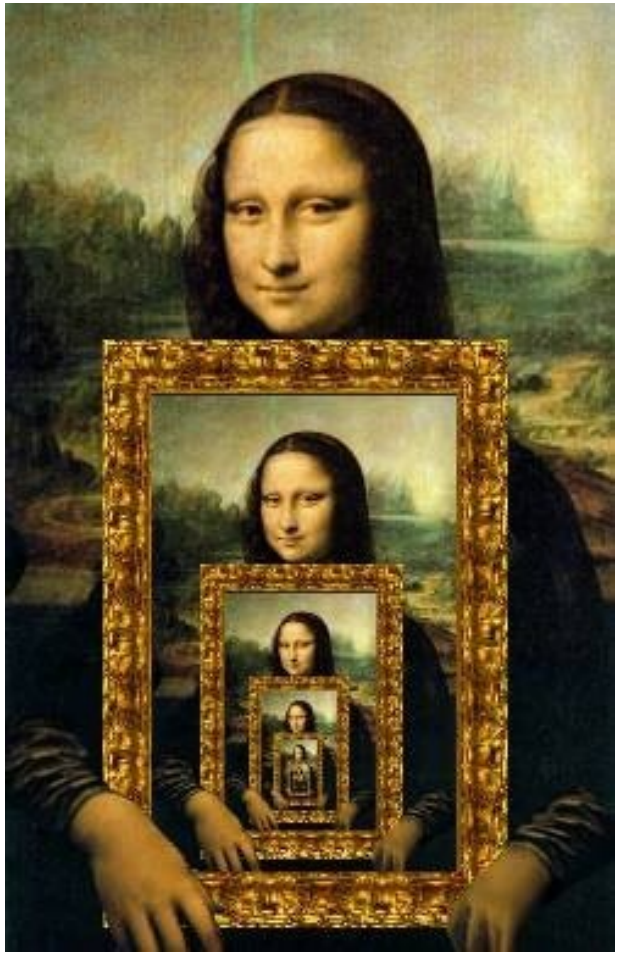
Lezione 25: Sommario

UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- La ricorsione
- I record di attivazione
- Esempi ed esercizi
- Strutture dati ricorsive

Ricorsione vs. Iterazione



Pensare ricorsivamente

- Dato un problema, bisogna identificare:
  - **I casi base**: problemi di dimensione minima e immediatamente calcolabile
  - **Il passo ricorsivo**: passo in cui si riduce l'attuale istanza del problema in uno o più sottoproblemi di dimensione minore combinati mediante una o più operazioni elementari

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

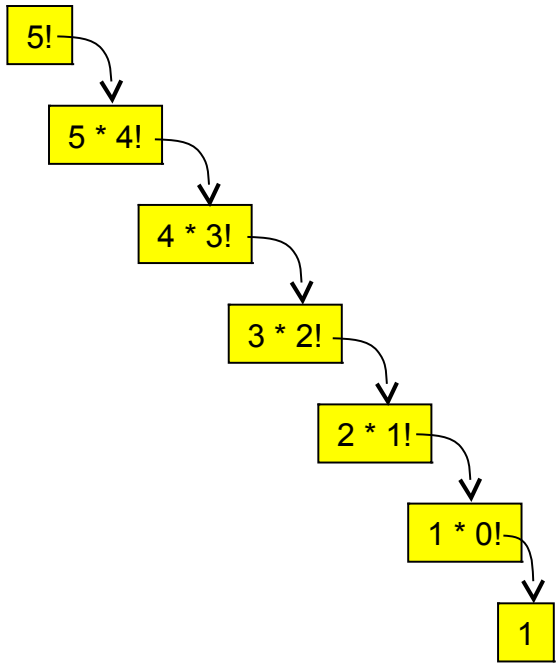
```
public int fattorialeIterativo(int n)
{
    int v = 1;
    for (int k = 1; k <= n; k++) v *= k;
    return v;
}
```

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

```
public int fattorialeRicorsivo(int n)
{
    if (n == 0) return 1;                    /* caso base */
    return n*fattorialeRicorsivo(n-1);       /* passo ricorsivo */
}
```

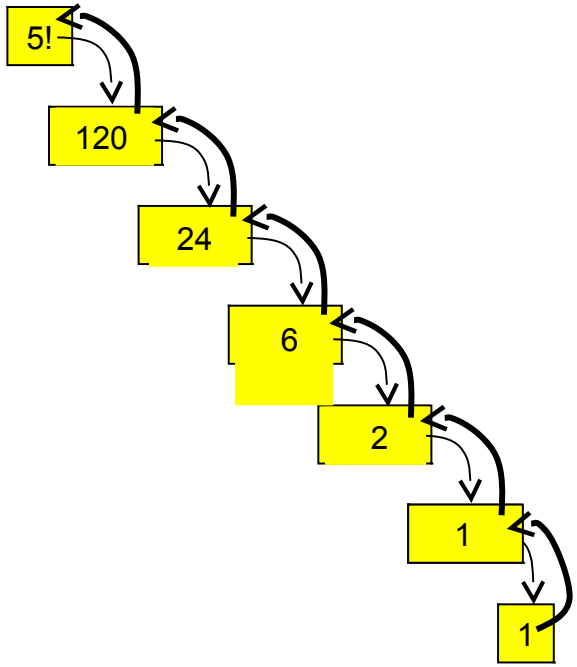
L'idea della ricorsione (1)

- Sequenza di chiamate ricorsive:



L'idea della ricorsione (2)

- Valori restituiti da ciascuna chiamata:

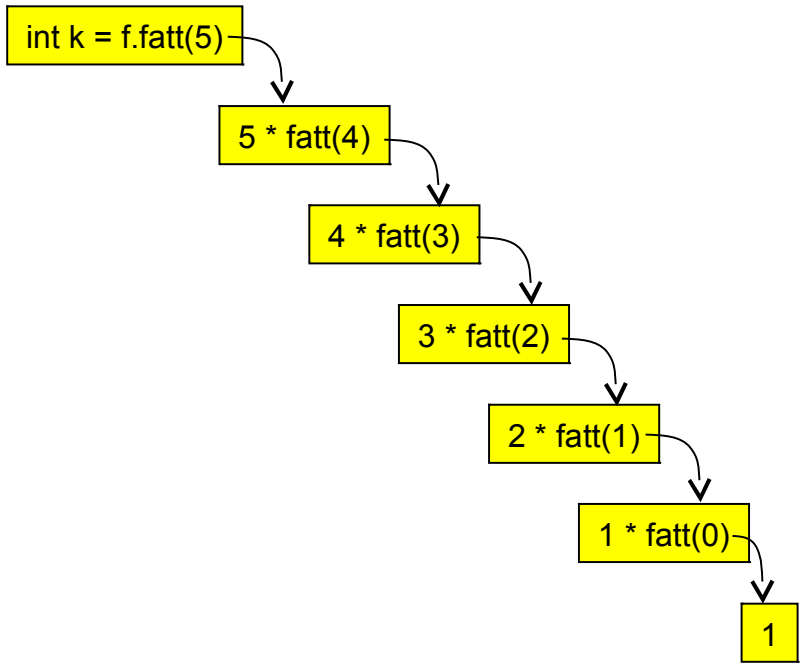


Come funziona la ricorsione? (1)

- Ogni volta che viene effettuata una chiamata a un metodo viene creato un nuovo ambiente, detto record di attivazione
- Un record di attivazione contiene la zona di memoria per le variabili locali del metodo e l'indirizzo di ritorno al metodo chiamante
  - A ogni chiamata, il record corrispondente viene aggiunto sullo stack
- Lo stack è la pila dei record di attivazione delle chiamate annidate effettuate

Come funziona la ricorsione? (2)

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




```
public int fattorialeRicorsivo(int n)
{
    if (n == 0) return 1;                    /* caso base */
    return n*fattorialeRicorsivo(n-1);       /* passo ricorsivo */
}
```

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

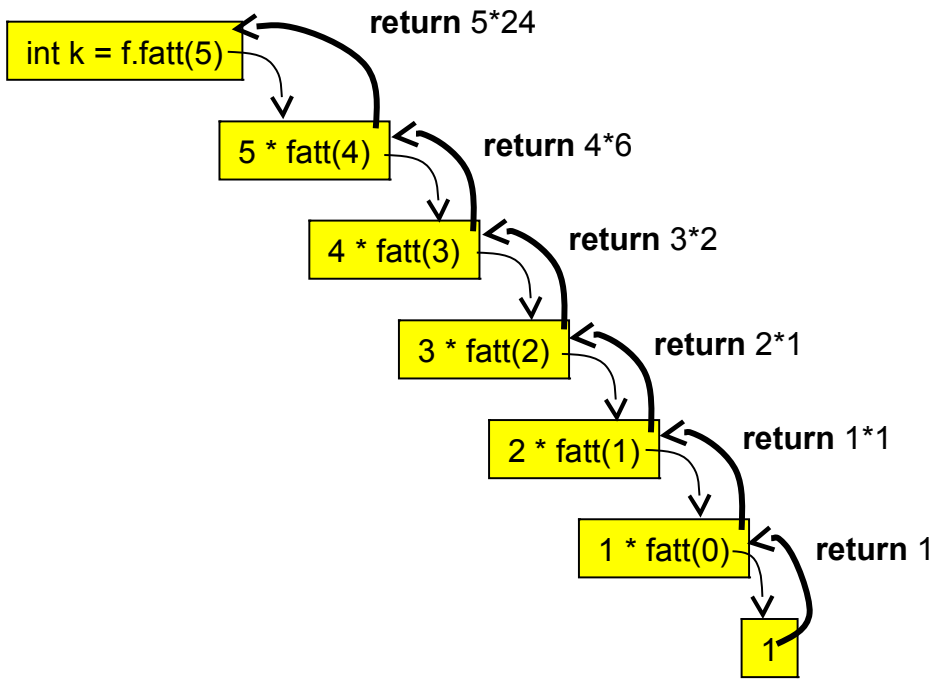
fatt(4)
n = 4
chiamante = Fattoriale.fatt

fatt(5)
n = 5
chiamante = Fattoriale.main

main(new String[] {})
args = new String[] {}
Fattoriale f = new Fattoriale()
chiamante = JVM

Come funziona la ricorsione? (3)

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

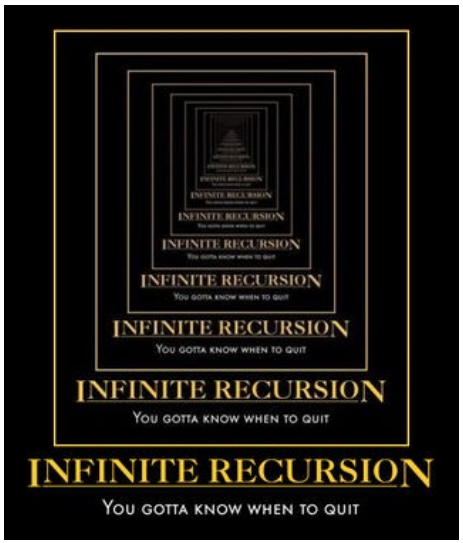



```
public int fattorialeRicorsivo(int n)
{
    if (n == 0) return 1;                    /* caso base */
    return n*fattorialeRicorsivo(n-1);       /* passo ricorsivo */
}
```

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

Non prevedere un caso base

- Che succede se chiamo `fattorialeRicorsivo(-1)`?
- Si ha una ricorsione (potenzialmente) infinita!



- Con quale conseguenza?

Non prevedere un caso base

- Che succede se chiamo `fattorialeRicorsivo(-1)`

```
Exception in thread "main" java.lang.StackOverflowError
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:13)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
	at Fattoriale.fattorialeRicorsivo(Fattoriale.java:14)
```

StackOverflowError, un java.lang.Error!

Fattoriale ricorsivo *FATTO BENE*


```java
public class FattorialeFattoBene
{
    public int fattorialeRicorsivo(int n) throws NumeroNonAmmessoException
    {
        if (n < 0) throw new NumeroNonAmmessoException(n);  /* caso non ammesso */
        if (n == 0) return 1;                               /* caso base */
        return n*fattorialeRicorsivo(n-1);                  /* passo ricorsivo */
    }
}
```

```java
public class NumeroNonAmmessoException extends Exception
{
    private int n;

    public NumeroNonAmmessoException(int n)
    {
        this.n = n;
    }

    public String toString()
    {
        return super.toString()+": "+n;
    }
}
```

Un'eccezione con un costruttore!

Esercizio: Riconoscere una stringa palindroma ricorsivamente

- Scrivere un metodo ricorsivo che determini se una stringa in ingresso è palindroma (ad esempio, itopinonavevanonipoti o ailatiditalia)

```
public class Palindroma
{
    public boolean isPalindroma(String s)
    {
        return isPalindroma(s, 0, s.length()-1);
    }

    private boolean isPalindroma(String s, int a, int b)
    {
        if (a >= b) return true;
        return s.charAt(a) == s.charAt(b) && isPalindroma(s, a+1, b-1);
    }
}
```

Esercizio: Concatenazione ricorsiva di stringhe

- Si realizzi un metodo ricorsivo che, dato in input un array di stringhe, ne restituisca la loro concatenazione
- Ad esempio, concatena(new String[] { “abc”, “de”, “f” }) restituisce “abcdef”

Esercizio: Ricerca binaria ricorsiva

- Scrivere un metodo ricorsivo per la ricerca binaria di un elemento all'interno di un array ordinato di interi
  - Restituendo -1 se l'elemento non è presente

```
public class RicercaBinaria
{
    public int cerca(int[] array, int elemento)
    {
        return cerca(array, elemento, 0, array.length-1);
    }

    private int cerca(int[] array, int elemento, int a, int b)
    {
        if (a > b) return -1;

        int meta = (b+a)/2;
        if (array[meta] == elemento) return meta;
        if (elemento < array[meta]) return cerca(array, elemento, a, meta-1);
        return cerca(array, elemento, meta+1, b);
    }
}
```

Strutture dati ricorsive (1)

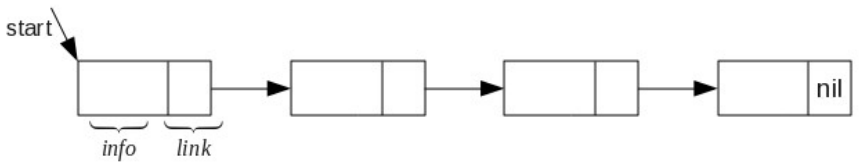
- Ad esempio, una lista "linkata" (ricordate?):

```
public class Lista
{
    private int k;
    private Lista succ;

    public Lista(int k) { this.k = k; }

    public void aggiungi(int k)
    {
        if (succ == null) succ = new Lista(k);
        else
        {
            Lista old = succ;
            succ = new Lista(k);
            succ.succ = old;
        }
    }

    public int getValore() { return k; }
    public Lista getSuccessivo() { return succ; }
}
```



Strutture dati ricorsive (2)

- Ad esempio, un labirinto:

```
import java.util.ArrayList;

public class Corridoio
{
    private ArrayList<Corridoio> corridoi;

    public Corridoio(ArrayList<Corridoio> corridoi)
    {
        this.corridoi = corridoi;
    }

    public boolean getUscita() { return false; }

    public Corridoio getCorridoio(int k)
    {
        return corridoi.get(k);
    }
}

public class CorridoioUscita extends Corridoio
{
    public CorridoioUscita()
    {
        super(null);
    }

    public boolean getUscita() { return true; }
}
```

Esercizio: Visualizzare il contenuto di una cartella

- Si realizzi una classe Cartella, costruita con il percorso di una cartella reale, dotata di un metodo toString() che ne visualizzi ricorsivamente il contenuto
- Avete bisogno della classe java.io.File

Visualizzare il contenuto di una cartella: soluzione


```
import java.io.File;

public class Cartella
{
    private File cartella;

    public Cartella(String percorso)
    {
        cartella = new File(percorso);
    }

    private String getSpazi(int k)
    {
        StringBuffer sb = new StringBuffer();
        while(k-- > 0) sb.append(" ");
        return sb.toString();
    }

    @Override
    public String toString()
    {
        return getElencoFile(cartella, 0);
    }

    private String getElencoFile(File c, int profondita)
    {
        StringBuffer sb = new StringBuffer();
        for (File f : c.listFiles())
        {
            sb.append(getSpazi(profondita));
            if (f.isDirectory())
            {
                sb.append("<");
                sb.append(f.getName());
                sb.append(">\n");

                sb.append(getElencoFile(f, profondita+1));
            }
            else
            {
                sb.append(f.getName());
                sb.append("\n");
            }
        }

        return sb.toString();
    }
}
```

Esercizio: Cercare file in una cartella

- Si realizzi una classe Cartella, costruita con il percorso di una cartella reale, dotata dei seguenti metodi:
  - Cerca un file all'interno di una cartella
  - Cerca all'interno della cartella tutti i file con l'estensione specificata
  - Cerca all'interno della cartella tutti i file con le estensioni fornite in input
- Si faccia uso al meglio dell'overloading

Esercizio: Somma ricorsiva di numeri

- Supponiamo di non possedere l'operatore di somma ma solo l'incremento (++) e il decremento (--)
- Come si può implementare la somma?

```
public class SommaRicorsiva
{
    public int somma(int k, int j)
    {
        if (j == 0) return k;
        int s = somma(k, --j);
        s++;
        return s;
    }
}
```

- Quanto è efficiente?
- E nella versione iterativa?

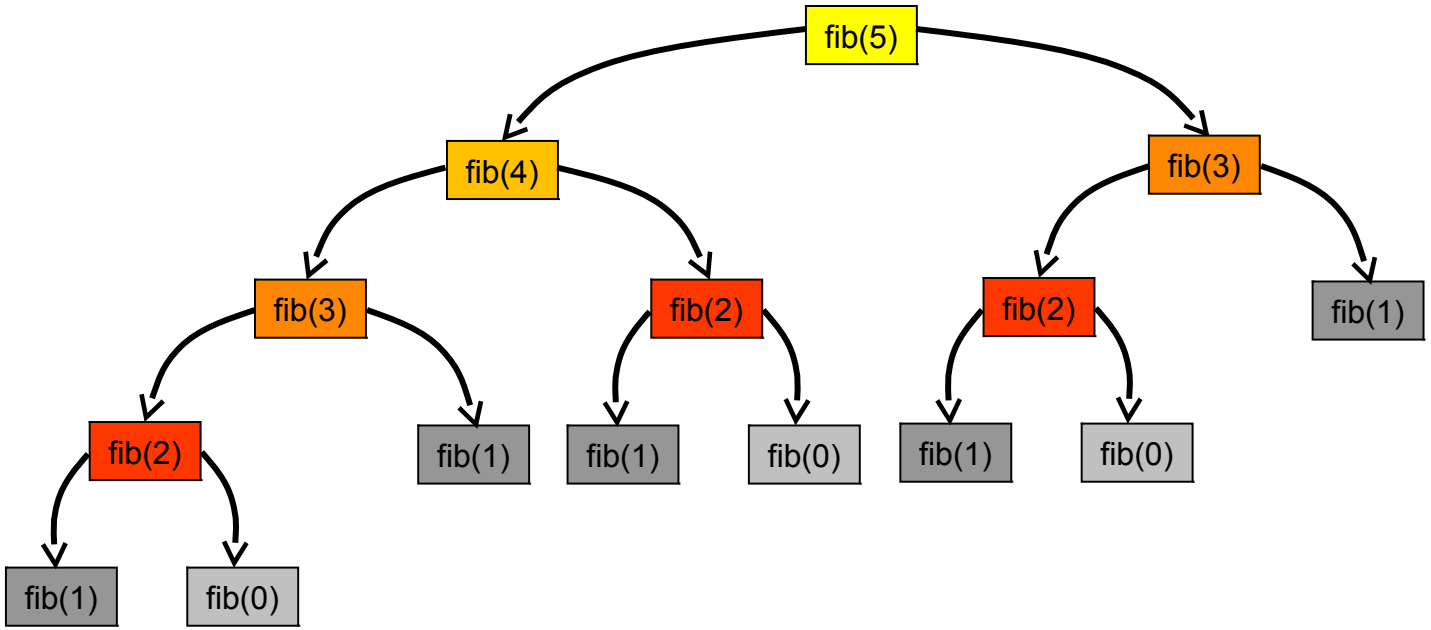
La successione di Fibonacci

- Come è definita la successione di Fibonacci?
  - fib(0) = 0
  - fib(1) = 1
  - fib(x) = fib(x-1)+fib(x-2) se x > 1
- E' una definizione ricorsiva!

```
public int fib(int x) throws NumeroNonAmmessoException
{
    if (x < 0) throw new NumeroNonAmmessoException(x);
    if (x == 0 || x == 1) return x;
    return fib(x-1)+fib(x-2);
}
```

- E' una implementazione efficiente?

Schema delle invocazioni del metodo ricorsivo fib



Esercizio: Permutazioni di una stringa

- Scrivere una classe costruita con una stringa e dotata di un metodo generaPermutazioni che restituisce l'elenco completo delle permutazioni della stringa
  - ad esempio, per la stringa "abc" genera l'elenco ["abc", "acb", "bac", "bca", "cab", "cba"]
- Come procedere? E' necessario scomporre il problema in sottoproblemi
- Fissato un carattere della stringa iniziale, lo poniamo all'inizio della stringa e permutiamo la sottostringa rimanente
  - stesso problema, ma su una sequenza di caratteri più piccola!

Permutazioni di una stringa: soluzione

UNITELMA SAPIENZA
SAPIENZA UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

```
import java.util.ArrayList;

public class GeneraPermutazioni
{
    private String parola;

    public GeneraPermutazioni(String parola)
    {
        this.parola = parola;
    }

    public ArrayList<String> getPermutazioni()
    {
        ArrayList<String> permutazioni = new ArrayList<String>();

        // caso base
        if (parola.length() == 0)
        {
            permutazioni.add(parola);
            return permutazioni;
        }

        for (int k = 0; k < parola.length(); k++)
        {
            // componi una parola più breve eliminando il carattere i-esimo
            String parolaPiuBreve = parola.substring(0, k)+parola.substring(k+1);

            // genera tutte le permutazioni della parola più breve
            ArrayList<String> permutazioniParolaPiuBreve = new GeneraPermutazioni(parolaPiuBreve).getPermutazioni();

            for (String p : permutazioniParolaPiuBreve)
                permutazioni.add(parola.charAt(k)+p);
        }

        return permutazioni;
    }
}
```

Esercizio: Teseo, il Minotauro e il Labirinto di Creta

- Si realizzi una classe Labirinto dotata di un solo ingresso a un corridoio
- Ciascun corridoio fornisce zero o più ingressi ad altri corridoi
- Ogni corridoio è dotato di un metodo contieneMinotauro che specifica se nel corridoio si trova il Minotauro
- La classe Labirinto è dotata di un metodo percorri() che, partendo dal corridoio iniziale, cerca il Minotauro nel labirinto
- **Avanzato:** Teseo, che fa il suo ingresso nel Labirinto, una volta trovato il Minotauro, stampa il percorso a ritroso mediante il "filo di Arianna"…

Variante: Versione con Labirinto a matrice

- Si realizzi una classe LabirintoMatrice in cui il labirinto è rappresentato mediante una matrice NxM (due valori in input al costruttore)
- Ogni cella della matrice è un'istanza di una classe Cella
  - Una cella o è una stanza o è un muro
  - Una stanza può o meno contenere il minotauro
- La classe LabirintoMatrice è dotata di un metodo percorri() che, partendo dalla cella (x,y) specificata in ingresso al metodo, cerca il Minotauro nel labirinto
- **Avanzato:** stampare il percorso a ritroso