La Ricorsione

Carla Binucci e Walter Didimo

Definizione induttiva di funzioni

Una funzione definita in modo induttivo è una funzione definita in termini di sé stessa

- ad esempio la funzione f(n) = n! (fattoriale di n), con n naturale, può essere definita in modo induttivo come segue:

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

Caso base e caso induttivo

Nella definizione induttiva di una funzione è possibile identificare:

- un caso base (o più di uno)
  - descrive in modo diretto il valore della funzione su alcuni argomenti
- un caso induttivo (o più di uno)
  - descrive in modo indiretto il valore della funzione su altri argomenti

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

caso base

caso induttivo

Esempio: calcolo di 3!

Utilizzando la definizione induttiva della funzione fattoriale, proviamo a calcolare il fattoriale di tre (3!):

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

- 3 è diverso da 0, quindi 3! può essere calcolato come 3 x 2!
- 2 è diverso da 0, quindi 2! può essere calcolato come 2 x 1!
- 1 è diverso da 0, quindi 1! può essere calcolato come 1 x 0!
- ma 0! può essere calcolato in modo diretto, (caso base) e vale 1
- quindi 1! = 1 x 0! = 1 x 1 = 1
- quindi 2! = 2 x 1! = 2 x 1 = 2
- quindi 3! = 3 x 2! = 3 x 2 = 6

Un altro esempio: la funzione potenza

E' possibile definire in modo induttivo anche altre funzioni matematiche

- ad esempio la funzione potenza x^y con x e y naturali, può essere definita in modo induttivo in questo modo:

$$x^y = \begin{cases} 1 & \text{se } y = 0 \quad \text{(caso base)} \\ x * x^{(y-1)} & \text{se } y > 0 \quad \text{(caso induttivo)} \end{cases}$$

Esercizio: utilizzando la definizione induttiva della funzione potenza, prova a calcolare 3^4

La ricorsione come metodologia

La ricorsione è una tecnica di programmazione basata sull'induzione matematica

- consente di scrivere metodi che eseguono operazioni o calcolano funzioni definite in modo induttivo
- consente di scrivere metodi decomponendo un problema in problemi dello stesso tipo (cioè della stessa natura) ma di dimensione più piccola
  - ad esempio, ridurre il problema del calcolo di 3! a quello del calcolo di 2!

Definizione di metodi ricorsivi

Un metodo ricorsivo è un metodo che direttamente o indirettamente, può invocare sé stesso

- ad esempio, la funzione f(n) = n! può essere calcolata mediante il seguente metodo ricorsivo *fattoriale*:

```
/* calcola il fattoriale del naturale n */
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

Struttura dei metodi ricorsivi

Un metodo ricorsivo che calcola una funzione definita in modo induttivo è tipicamente strutturato nel seguente modo:

- dichiara una variabile che rappresenta il valore da calcolare e da restituire (nell'esempio *f*, che rappresenta il fattoriale di n)
- contiene una istruzione condizionale per selezionare il caso (base o induttivo) in cui ci si trova e quindi per assegnare alla variabile un valore sulla base della definizione induttiva della funzione
- termina con la restituzione del valore calcolato

Ulteriori osservazioni

- E’ tipicamente sbagliato tentare di utilizzare istruzioni iterative nell’ambito di metodi ricorsivi

- La ricorsione è una tecnica alternativa all’iterazione!

Un altro esempio di metodo ricorsivo

Ad esempio, utilizzando la definizione induttiva della funzione potenza è possibile scrivere il seguente medodo ricorsivo *potenza*:

```
/* calcola x alla y, x e y sono naturali */
public static int potenza(int x, int y) {
    // pre: x>=0 && y>=0
    int p;
    if (y==0)              // caso base
        p = 1;
    else                   // caso induttivo
        p = x * potenza(x,y-1);
    return p;
}
```

Esecuzione di metodi ricorsivi

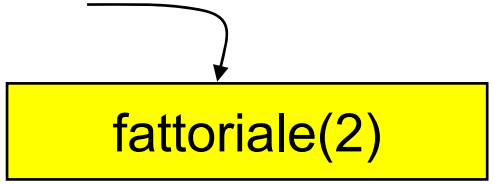
Cosa accade quando viene invocato un metodo ricorsivo?

- come esempio, proviamo a seguire, passo dopo passo, cosa accade quando viene invocato *fattoriale(2)*

```
/* calcola il fattoriale del naturale n */
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

Calcolo di fattoriale(2)

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

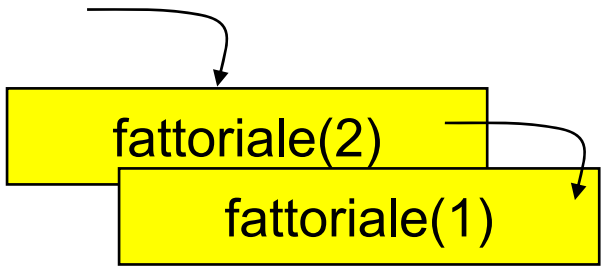


Prima attivazione

- il metodo controlla se *n==0*, ma poiché *n* vale *2*, esegue la parte *else* dell'istruzione condizionale
- per poter assegnare un valore a *f*, deve valutare l'espressione *n*fattoriale(n-1)*: *n* vale *2*, ma non è noto quanto vale *fattoriale(1)*
- il metodo viene temporaneamente sospeso e viene invocato nuovamente il metodo *fattoriale* con argomento *1*

Calcolo di fattoriale(2)

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

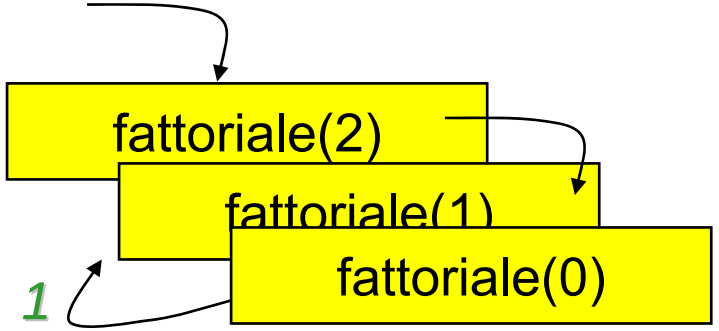


Seconda attivazione

- il metodo controlla se *n==0*, ma poiché *n* vale *1*, esegue la parte *else* dell'istruzione condizionale
- per poter assegnare un valore a *f*, deve valutare l'espressione *n*fattoriale(n-1)*: *n* vale *1*, ma non è noto quanto vale *fattoriale(0)*
- il metodo viene temporaneamente sospeso e viene invocato nuovamente il metodo *fattoriale* con argomento *0*

Calcolo di fattoriale(2)

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

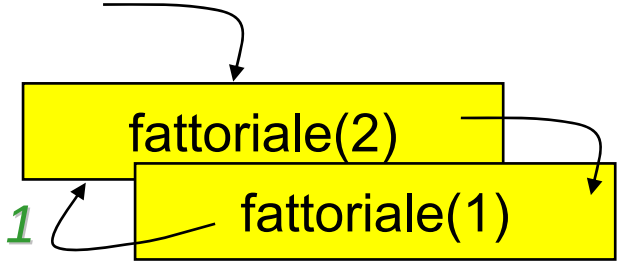


Terza attivazione

- il metodo controlla se *n==0*, e in questo caso *n* vale proprio *0*: deve essere eseguita la parte *if* dell'istruzione condizionale
- viene assegnato a *f* il valore *1*
- questa attivazione del metodo termina restituendo il valore *1*

Calcolo di fattoriale(2)

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

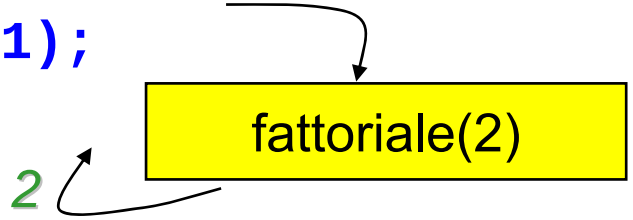


Seconda attivazione

- quando la terza attivazione termina, si tornano ad eseguire le istruzioni del metodo relativamente alla seconda attivazione
- l'invocazione di *fattoriale(0)* ha restituito il valore *1*, ed è possibile eseguire la moltiplicazione con *n* (che vale *1*) e assegnare il risultato a *f* che ora vale *1*
- questa attivazione del metodo termina restituendo il valore *1*

Calcolo di fattoriale(2)

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```



Prima attivazione

- quando la seconda attivazione termina, si tornano ad eseguire le istruzioni del metodo relativamente alla prima attivazione
- l'invocazione di *fattoriale(1)* ha restituito il valore *1*, ed è possibile eseguire la moltiplicazione con *n* (che vale *2*) e assegnare il risultato a *f* che ora vale *2*
- questa attivazione del metodo termina restituendo il valore *2*

Richiamo sull'esecuzione dei metodi

L' esecuzione dei metodi viene gestita dalla macchina virtuale Java mediante il modello basato su *record di attivazione* e *pila di attivazione*:

- ogni volta che viene invocato un metodo, la JVM introduce nella pila di attivazione un nuovo record di attivazione contenente, tra l'altro, queste informazioni:
  - ricevitore
  - punto di ritorno
  - variabili locali – tra cui i parametri formali inizializzati con il valore dei corrispondenti parametri attuali

Esecuzione di metodi ricorsivi

Il meccanismo della *pila di attivazione* permette l'esecuzione di metodi ricorsivi:

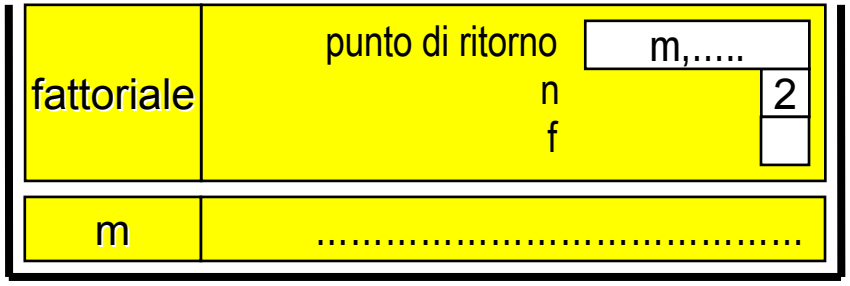
- una *invocazione ricorsiva* può essere vista come l'invocazione di un nuovo metodo, che ha lo stesso nome del metodo che lo invoca
  - ciascuna invocazione ricorsiva di un metodo richiede l'allocazione di un nuovo record di attivazione
  - le variabili locali di un metodo sono locali alle singole attivazioni
  - bisogna fare delle precisazioni su come viene specificato il punto di ritorno

Esecuzione di metodi ricorsivi – fattoriale(2)

```
0 public static int fattoriale(int n) {
1    int f;
2    if (n==0)          // caso base
3        f = 1;
4    else               // caso induttivo
5        f = n * fattoriale(n-1);
6    return f;
7 }
```

Supponiamo che *fattoriale(2)* venga invocato da un metodo *m*

- al momento dell'invocazione viene creato il record di attivazione per tale metodo e inserito nella pila di attivazione
- trascuriamo i ricevitori

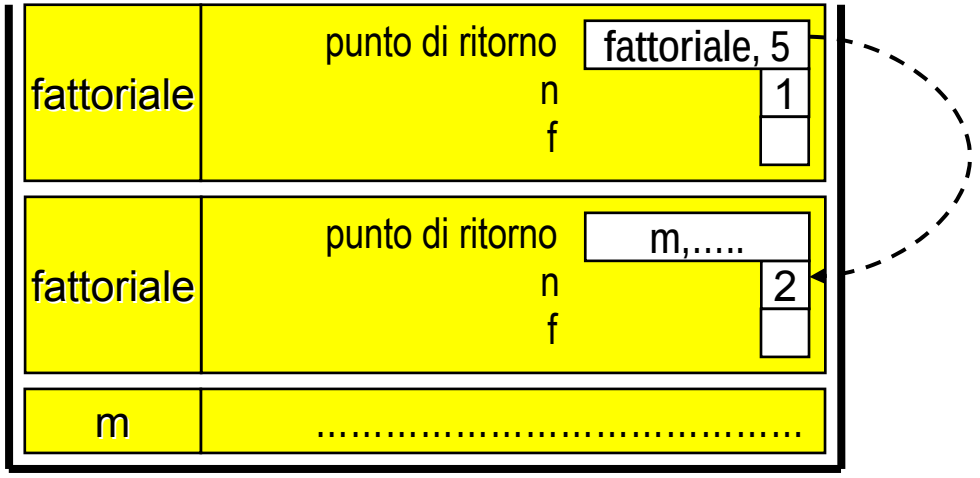


pila di attivazione

Esecuzione di metodi ricorsivi – fattoriale(2)

L'attivazione di *fattoriale(2)* invoca alla linea 5, *fattoriale(1)*

- l'esecuzione di *fattoriale(2)* viene sospesa
- viene invocato *fattoriale(1)* e allocato un nuovo record di attivazione
  - nella pila di attivazione, il punto di ritorno di un record di attivazione fa sempre riferimento all'attivazione descritta nel record di attivazione immediatamente sottostante

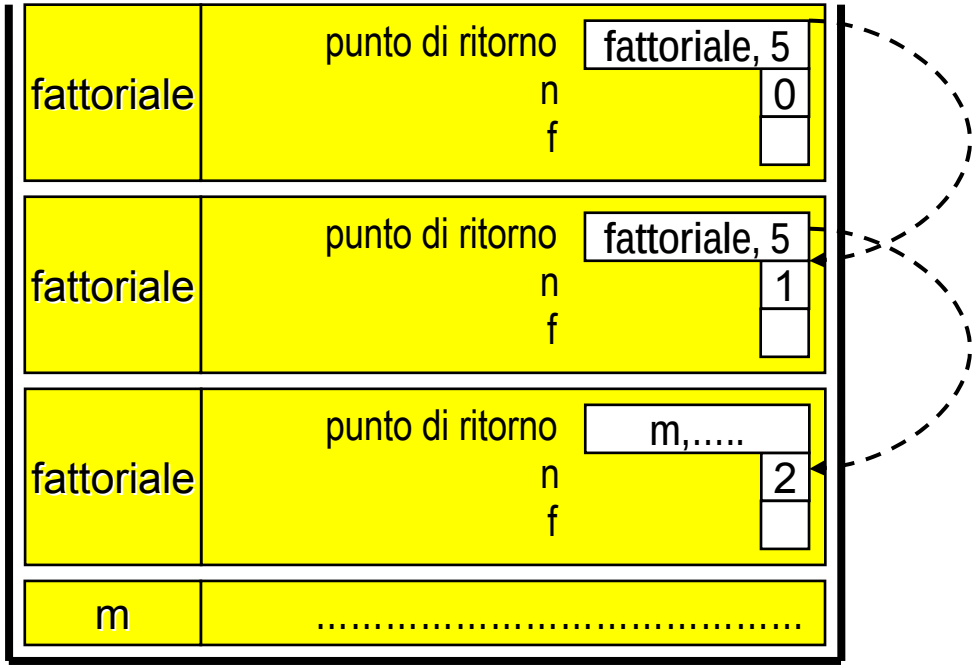


pila di attivazione

Esecuzione di metodi ricorsivi – fattoriale(2)

L'attivazione di *fattoriale(1)* invoca alla linea 5, *fattoriale(0)*

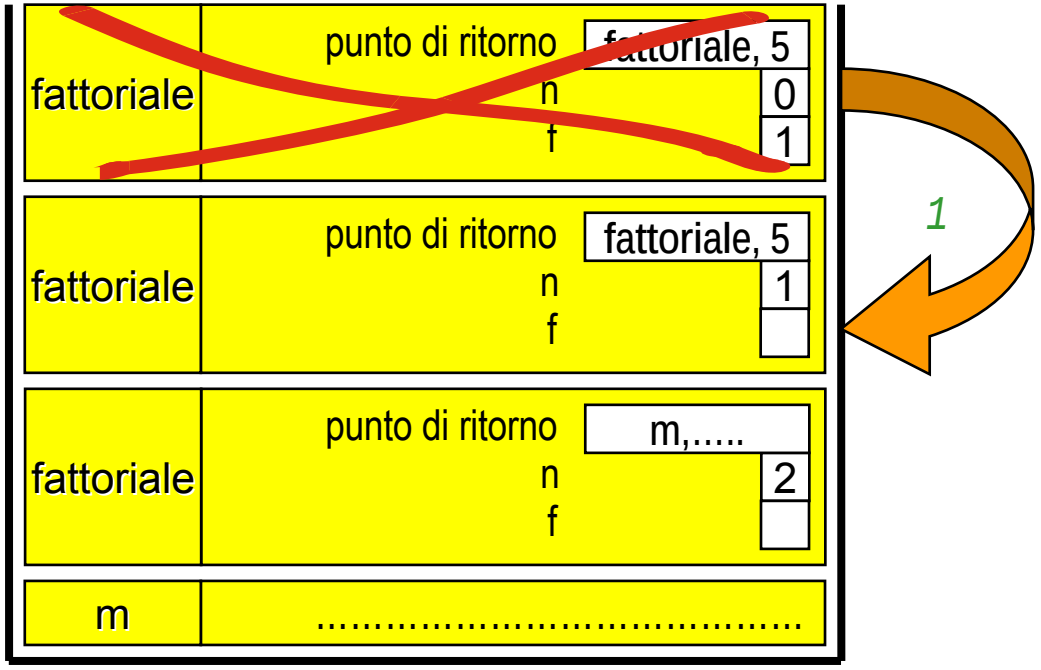
- anche l'esecuzione di *fattoriale(1)* viene sospesa
- viene invocato *fattoriale(0)* e allocato un nuovo record di attivazione



pila di attivazione

Esecuzione di metodi ricorsivi – fattoriale(2)

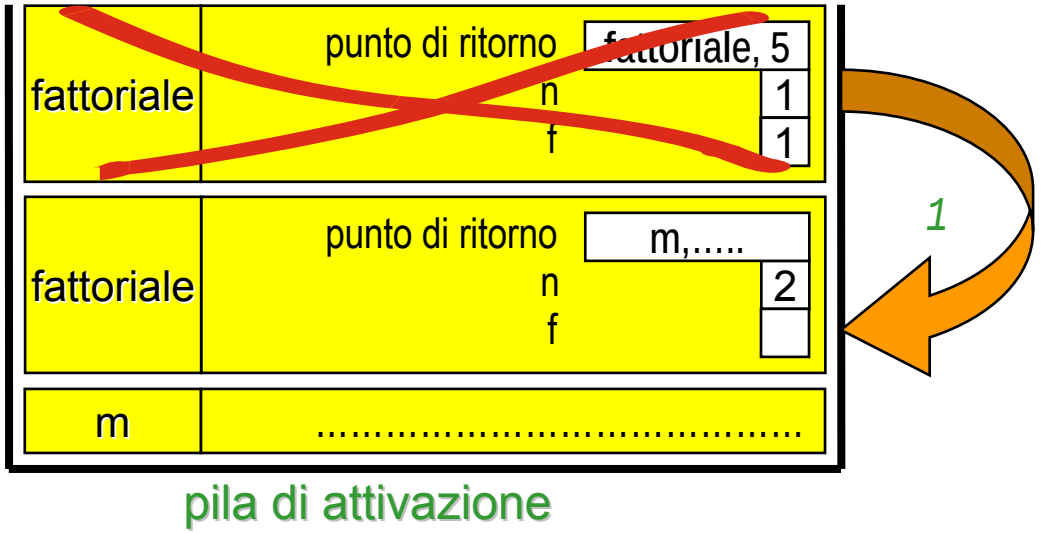
L'attivazione di *fattoriale(0)* valuta il caso base, assegna ad *f* il valore *1* e termina restituendo il valore *1*



pila di attivazione

Esecuzione di metodi ricorsivi – fattoriale(2)

Terminata l'esecuzione di *fattoriale(0)* il controllo torna all'attivazione di *fattoriale(1)* che può completare la valutazione dell'istruzione alla linea 5 (assegnando ad *f* il valore *1*) e poi terminare restituendo il valore *1*

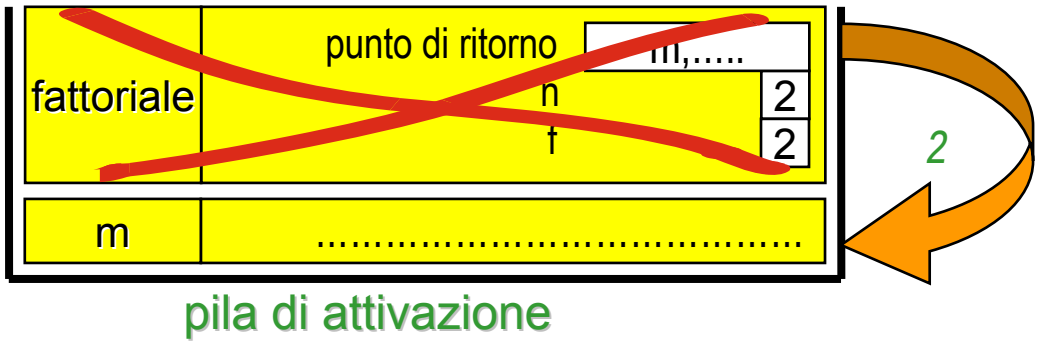


pila di attivazione

Esecuzione di metodi ricorsivi – fattoriale(2)

Terminata l'esecuzione di *fattoriale(1)*, il controllo torna all'attivazione di *fattoriale(2)* che può completare la valutazione dell'istruzione alla linea 5 (assegnando ad *f* il valore *2*) e poi terminare

- l'invocazione di *fattoriale(2)* restituisce il valore *2* al metodo *m* che lo aveva invocato
- ora l'elemento in cima alla pila è il record di attivazione del metodo *m* a cui torna il controllo affinché possa riprendere l'esecuzione dove l'aveva interrotta



Ricorsione e iterazione

Per i metodi ricorsivi esaminati finora si può scrivere anche una versione iterativa equivalente.

Quando conviene usare la ricorsione?

- la ricorsione è più naturale per il calcolo di funzioni definite in modo induttivo (ad esempio, la potenza o il fattoriale)
- la ricorsione permette di generare e controllare processi iterativi senza dover definire delle apposite strutture di memorizzazione
  - la struttura di memorizzazione per i processi ricorsivi è la pila delle invocazioni ricorsive che viene generata automaticamente dalla JVM durante l'esecuzione di un metodo ricorsivo

Altri esempi di funzioni induttive

Per alcune funzioni definite in modo induttivo è più semplice e immediato scrivere un metodo ricorsivo che le calcola piuttosto che scrivere un metodo iterativo

- ad esempio, i *numeri di Fibonacci* sono definiti in questo modo, per *n* naturale

$$F(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

casi base

caso induttivo

Esercizio - Inversione di una sequenza

Vogliamo scrivere un metodo che legge dalla tastiera una sequenza di naturali terminata da uno zero (che non appartiene alla sequenza) e la stampa al contrario (cioè dall'ultimo elemento al primo)

```
Scrivi una sequenza da invertire
10 20 5 8 0
La sequenza invertita è 8 5 20 10
```

E' possibile risolvere iterativamente questo problema usando un array

- da quanti elementi deve essere composto questo array ?

Mostriamo ora una soluzione ricorsiva

Inversione di una sequenza - soluzione

Soluzione ricorsiva del problema dell'inversione di una sequenza

```
/* punto di ingresso alle chiamate ricorsive */
public static void inverteSequenza() {
    InputWindow in = new InputWindow();
    OutputWindow out = new OutputWindow();
    inversioneSequenza(in,out);
}

/* metodo ricorsivo */
private static void inversioneSequenza(InputWindow in,OutputWindow out){
    int n;
    n = in.readInt("Prossimo numero – 0 per terminare");
    if (n==0)                    // caso base
        out.write("La sequenza invertita è: ");
    else {                       // caso induttivo
        inversioneSequenza(in,out);
        out.write(n + " ");
    }
}
```

Progettazione di metodi ricorsivi

Ecco alcune linee guida per la progettazione di metodi ricorsivi

- determina un *caso base*: un caso base è indispensabile, talvolta ne sono necessari più di uno
- determina un *caso ricorsivo* (cioè *induttivo*), o magari anche più di uno
- verifica che i casi base e induttivi determinati partizionano l'insieme dei possibili dati di ingresso del metodo

..... (continua)

Progettazione di metodi ricorsivi

- usa una istruzione condizionale (solitamente una cascata di *if-else*) per selezionare il caso corrente e per eseguire le azioni corrispondenti
- non usare istruzioni ripetitive!!
  - raramente un metodo ricorsivo contiene istruzioni ripetitive e, se le contiene, esse non hanno lo scopo di controllare la ricorsione
- spesso è utile dichiarare una variabile per memorizzare il risultato del metodo
- talvolta è necessario scrivere anche un metodo (non ricorsivo) per avviare la ricorsione

Errori comuni – caso base

Errore nella definizione del caso base

- ad esempio, supponi che nel calcolo del fattoriale sia stato scelto il caso base *f = 0* se *n = 0*

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 0;         // NO, SBAGLIATO!
    else               // caso induttivo
        f = n * fattoriale(n-1);
    return f;
}
```

questo metodo *fattoriale* restituisce *0* per qualunque valore di n

Errori comuni – caso induttivo

Errore nella definizione del caso induttivo

- ad esempio, supponi che nel calcolo del fattoriale si invochi ricorsivamente *fattoriale(n)* anziché *fattoriale(n-1)*

```
public static int fattoriale(int n) {
    // pre: n>=0
    int f;
    if (n==0)          // caso base
        f = 1;
    else               // caso induttivo
        f = n * fattoriale(n); // NO,SBAGLIATO!
    return f;
}
```

questo metodo *fattoriale* non termina se *n>0*, perché non riconduce mai il calcolo al caso base

Metodi ricorsivi e tipi ricorsivi

La ricorsione è utile in modo particolare nell'implementazione di operazioni ricorsive relative a tipi definiti ricorsivamente

Un tipo ricorsivo è un tipo definito in termini di sé stesso

- ad esempio, il tipo delle stringhe può essere definito in modo induttivo

Stringhe definite in modo ricorsivo

Una stringa non vuota di caratteri alfabetici può essere vista in questo modo:

una stringa non vuota =
- un carattere ← caso base
- un carattere seguito da una stringa non vuota ← caso induttivo

Una stringa (possibilmente vuota) di caratteri alfabetici può essere invece vista in questo modo:

una stringa =
- la stringa vuota ← caso base
- un carattere ← caso base
- un carattere seguito da una stringa ← caso induttivo

Esempio - uguaglianza fra stringhe

Due stringhe s e t sono uguali se

- sono entrambe vuote
- sono entrambe non vuote e
  - s può essere scritta come $c_s\ s'$ — dove c_s è il primo carattere di s e s' è il "resto" di s
  - t può essere scritta come $c_t\ t'$
  - $c_s = c_t$ e s' è uguale a t'

Ad esempio

"alfa" e *"alfa"* sono uguali — infatti

'a' == *'a'* e *"lfa"* e *"lfa"* sono uguali — infatti

'l' == *'l'* e *"fa"* e *"fa"* sono uguali — infatti

'f' == *'f'* e *"a"* e *"a"* sono uguali — infatti

'a' == *'a'* e *""* e *""* sono uguali

Uguaglianza fra stringhe

```
/* Verifica se le stringhe s e t sono uguali */

public static boolean uguali(String s, String t) {
   // pre: s!=null && t!=null
   boolean esito;
   if (s.length()==0 && t.length()==0)
       esito = true;
   else if (s.length()>0 && t.length()>0)
       esito = (s.charAt(0) == t.charAt(0)) &&
                  uguali(s.substring(1), t.substring(1));
   else // una sola delle stringhe è vuota
       esito = false;
   return esito;
  }
```

Una soluzione più efficiente può essere realizzata evitando di creare nuove stringhe — lavorando sempre sulle stringhe *s* e *t*

Una soluzione più efficiente

```
/* verifica se s e t sono uguali */
public static boolean uguali(String s, String t) {
    // pre: s!=null && t!=null
    return ugualiRic(s, t, 0);
}

/* verifica se s e t sono uguali dal carattere di
 * posizione p in poi */
private static boolean ugualiRic(String s, String t, int p){
    // pre: s!=null && t!=null &&
    //      s.length()>=p && t.length()>=p
    boolean esito;
    if (s.length()==p && t.length()==p)
        esito = true;
    else if (s.length()>p && t.length()>p)
        esito = (s.charAt(p) == t.charAt(p)) &&
                ugualiRic(s, t, p+1);
    else // una sola delle stringhe è lunga p
        esito = false;
    return esito;
}
```

Esempio – frequenza di un carattere

La frequenza di un carattere *c* in una stringa *s* è

- 0 - se *s* è vuota
- la frequenza di *c* nel resto di *s* - se *s* non è vuota e il suo primo carattere è diverso da *c*
- 1 più la frequenza di *c* nel resto di *s* - se *s* non è vuota e il suo primo carattere è uguale a *c*

Ad esempio

- la frequenza di *'a'* in *"alfa"* è *2* - infatti
- il primo carattere di *"alfa"* è *'a'* e la frequenza di *'a'* in *"lfa"* è *1* - infatti
- il primo carattere di *"lfa"* è diverso da *'a'* e la frequenza di *'a'* in *"fa"* è *1* - infatti
- il primo carattere di *"fa"* è diverso da *'a'* e la frequenza di *'a'* in *"a"* è *1* - infatti
- il primo carattere di *"a"* è 'a' e la frequenza di 'a' in *""* è *0*

Frequenza di un carattere

```
/* Calcola la frequenza del carattere c nella
stringa (non nulla) s */
public static int frequenza(String s, char c) {
      // pre: s!=null
      int f;
      if (s.length()==0)
          f = 0;
      else if (s.charAt(0)==c)
          f = 1 + frequenza(s.substring(1),c);
      else
          f = frequenza(s.substring(1),c);
      return f;
  }
```

Esercizio

- realizza una soluzione più efficiente evitando di creare nuove stringhe (come nell'esempio precedente)

Confronto lessicografico

Esercizio

- Scrivere un metodo che, date due stringhe s e t, calcola la relazione di ordine lessicografico $s \sim t$ tra s e t, definita come

$$s \sim t = \begin{cases} 0 & \text{se } s \text{ e } t \text{ sono entrambe vuote} \\ -1 & \text{se } s \text{ è vuota e } t \text{ è non vuota} \\ 1 & \text{se } s \text{ è non vuota e } t \text{ è vuota} \\ -1 & \text{se } s \text{ e } t \text{ sono non vuote, e } c_s < c_t \\ 1 & \text{se } s \text{ e } t \text{ sono non vuote, e } c_s > c_t \\ s' \sim t' & \text{se } s \text{ e } t \text{ sono non vuote, e } c_s = c_t \end{cases}$$

- dove

  0 significa che s è uguale a t

  -1 significa che s precede t

  1 significa che s segue t

Glossario dei termini principali

Termine	Significato
Funzione definita induttivamente	Una funzione definita in termini di se stessa
Metodo ricorsivo	Un metodo che può invocare se stesso
Tipo ricorsivo	Un tipo definito in termini di se stesso