Corso di Programmazione
Anno Accademico 2015–2016
Prof. Giovanni Pighizzini

Note ed esercizi aggiuntivi

Gli esercizi proposti sono utili per rivedere gli esempi riportati, che sono stati sviluppati e discussi in dettaglio a lezione.

15. Metodi e sottoprogrammi ricorsivi

Esempio. Programma ricorsivo che elenca le mosse per la soluzione del problema della Torre di Hanoi, sviluppato nella lezione 29 (non object oriented!).

```
import prog.io.*;

class Hanoi {

  public static void main(String[] a) {
    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    int num = in.readInt("Quanti dischi? ");
    out.println(" 1 = disco piu' piccolo, " + num + " = disco piu' grande");

    spostaTorre(num, 'A', 'C', 'B');
  }

  private static void spostaTorre(int n, char partenza, char destinazione, char supporto) {
    if (n == 1)
      spostaDisco(1, partenza, destinazione);
    else {
      spostaTorre(n - 1, partenza, supporto, destinazione);
      spostaDisco(n, partenza, destinazione);
      spostaTorre(n - 1, supporto, destinazione, partenza);
    }
  }

  private static void spostaDisco(int disco, char p, char d) {
    System.out.println(disco + ": " + p + " -> " + d);
  }
}
```

Esempio. Un metodo aggiuntivo per la classe `Frazione`: dato un intero `k`, fornito tramite il parametro, il metodo restituisce una nuova frazione di valore uguale alla frazione che esegue il metodo elevata alla potenza `k`.

```
public Frazione potenza(int k) {
  if (k == 0)
    return new Frazione(1);
  else if (k > 0) {
    Frazione f = this.potenza(k - 1);
    return this.per(f);
  } else {
    Frazione uno = new Frazione(1);
    Frazione inversa = uno.diviso(this);
    return inversa.potenza(-k);
  }
}
```

Note
Il metodo implementa una soluzione ricorsiva, basata sulla formula $f^k = \begin{cases} 1 & \text{se } k = 0 \\ f \cdot f^{k-1} & \text{se } k > 0 \\ \left(\frac{1}{f}\right)^{-k} & \text{se } k < 0 \end{cases}$

Esercizio 15.1
Scrivete una versione ricorsiva differente, in cui nel caso $k < 0$ si calcoli $\frac{1}{f^{-k}}$ al posto di $\left(\frac{1}{f}\right)^{-k}$.

Esercizio 15.2
Scrivete una versione iterativa del metodo `potenza`.

Esempio. Ordinamento ricorsivo tramite *merge-sort*.

```
import prog.io.ConsoleInputManager;
import prog.io.ConsoleOutputManager;

class Ordinamento {

  public static void main(String[] a) {
    ConsoleInputManager in = new ConsoleInputManager();
    ConsoleOutputManager out = new ConsoleOutputManager();

    int nNumeri = in.readInt("Quanti numeri vuoi ordinare? ");
    int[] numeri = new int[nNumeri];
    for (int i = 0; i < nNumeri; i ++)
      numeri[i] = in.readInt("Numero? ");

    ordina(numeri);

    out.println("Elenco ordinato: ");
    for (int x: numeri)
      out.println(x);
  }
```

```
  public static void ordina(int[] a) {
    if (a.length > 1) {
      int mezzo = a.length / 2;

      //costruisci un array corrispondente alla prima meta'
      //dell'array da ordinare
      int[] b = new int[mezzo];
      for (int i = 0; i < mezzo; i++)
        b[i] = a[i];

      //costruisci un array corrispondente alla seconda meta'
      //dell'array da ordinare
      int[] c = new int[a.length - mezzo];
      for (int i = mezzo; i < a.length; i++)
        c[i - mezzo] = a[i];

      //ordina i due array ottenuti
      ordina(b);
      ordina(c);

      //fondi i due array
      int posB = 0;  //prossima posizione da considerare nel primo array
      int posC = 0;  //prossima posizione da considerare nel secondo array

      for (int i = 0; i < a.length; i++)
        //assegna ad a[i] il minimo tra b[posB] e c[posC]
        if (posC == c.length || posB < b.length && b[posB] < c[posC])
          a[i] = b[posB++];
        else
          a[i] = c[posC++];
    } //else l'array e' gia' ordinato
  }
}
```

Note

- L'esempio (per nulla object-oriented) ha lo scopo di mostrare un uso interessante e non banale della ricorsione.

- Il metodo `ordina` è basato sul seguente schema ricorsivo:

```
if (l'array a contiene almeno due elementi) {
  costruisci due array b e c tali che:
    - b contenga gli elementi della prima meta' dell'array a,
    - c contenga gli elementi della seconda meta' dell'array a;
  ordina ricorsivamente gli array b e c
  fondi i due array ordinati b e c scrivendo il risultato nell'array a
} else non fare nulla (l'array e' gia' ordinato!)
```

- La fusione (detta anche *merge*) avviene selezionando di volta in volta il più piccolo tra gli elementi di `b` e `c` non ancora utilizzati:

  - a tale scopo le variabili `posB` e `posC` indicano, in ciascuno dei due array, la prossima posizione da considerare;
  - la condizione `b[posB] < c[posC]` permette di scegliere il minimo;
  - occorre tenere conto del fatto che a un certo punto verrà raggiunta la fine di uno dei due array (a tale scopo la condizione della selezione contiene ulteriori controlli, utilizzando il meccanismo della lazy evaluation).

- La soluzione presentata è dispendiosa in termini di memoria, in quanto ad ogni chiamata ricorsiva vengono creati i due array `b` e `c`. Più avanti è presentato un raffinamento che evita la creazione dei due array.

- Se si vuole che il metodo `ordina` sia a disposizione di altre classi e possa essere richiamato da altri metodi, si può dichiarare la classe `Ordinamento` come `public` (e collocarla poi in un package). In tal caso il metodo può essere richiamato dal codice di altre classi mediante `Ordimento.ordina(...)`. Nella classe `Ordinamento` si potrebbero, mediante overloading, definire altri metodi `ordina` per altri tipi di dati, o un metodo generico, come suggerito negli esercizi successivi.

- Il metodo `main`, estremamente rudimentale, presentato nell'esempio è fornito solo per provare il metodo `ordina`. Esso non è indispensabile se `ordina` è a disposizione di altre classi.

Esercizio 15.3
Utilizzando la stessa tecnica dell'esempio precedente, scrivete un metodo di ordinamento per un array di stringhe e un metodo di ordinamento per un array di frazioni (insieme a metodi di prova).

Esercizio 15.4
Utilizzando la stessa tecnica dell'esempio precedente, scrivete un metodo di ordinamento "generico" che ordini un array di oggetti di un qualunque tipo che implementi l'interfaccia `Comparable`.
(Per risolvere questo esercizio è necessario approfondire alcune tematiche che non sono state trattate nel corso relative a tipi e metodi generici. Alcune idee fondamentali sono presentate, per un differente algoritmo di ordinamento, nel paragrafo 11.3 del libro di testo.)

Esempio. Il metodo `ordina` dell'esempio precedente (con un metodo ausiliario ricorsivo `ord`), implementato senza utilizzare gli array ausiliari `b` e `c`.

```
public static void ordina(int[] a) {
  ord(a, 0, a.length);
}

private static void ord(int[] a, int i, int j) {
   //ordina la porzione di a dalla posizione i alla posizione j esclusa
  if (j > i + 1) {   // la porzione da ordinare contiene alemnto due elementi
    int mezzo = (i + j) / 2;

    //ordina ricorsivamente le due parti di array
    ord(a, i, mezzo);
    ord(a, mezzo, j);

    //fondi le due parti scrivendo il risultato in un array di supporto
    int[] fusione = new int[a.length];
    int posB = i;
    int posC = mezzo;
    for (int k = i; k < j; k++)
      //assegna a fusione[k] il minimo tra a[posB] e a[posC]
      if (posC == j || posB < mezzo && a[posB] < a[posC])
        fusione[k] = a[posB++];
      else
        fusione[k] = a[posC++];
    for (int k = i; k < j; k++)
      a[k] = fusione[k];
  }
}
```

Note

- L'algoritmo di ordinamento ricorsivo è implementato dal metodo `ord` che riceve tramite i parametri un riferimento `a` all'array e due indici `i` e `j`. Il metodo deve modificare l'array riferito da `a` ordinando in modo crescente la porzione `a[i],...,a[j - 1]`.

- Il metodo `ordina` si limita a richiamare il metodo `ord` fornendo come indici `0` e la lunghezza dell'array, in modo che venga ordinato l'intero array.

- Il metodo `ord` segue il medesimo schema ricorsivo del metodo `ordina` della versione precedente. In particolare, se la porzione da ordinare contiene almeno due elementi, essa viene suddivisa in due porzioni, delimitate dall'indice `mezzo`. Le due porzioni vengono ordinate ricorsivamente (chiamate ricorsive a `ord`). Si procede infine alla fusione.

- Per la fusione si utilizza un array ausiliario (riferimento `fusione`) in cui vengono copiati gli elementi durante la fusione (per comodità l'array è dimensionato come quello originario, in realtà se ne utilizzano solo le posizioni da `i` a `j - 1`). Infine, gli elementi vengono ricopiati nella porzione corrispondente di `a`.

- L'array `fusione` serve solo nella fase finale: non è necessario crearlo ad ogni chiamata ricorsiva, ma è sufficiente crearlo una volta sola all'inizio. A tale scopo l'array può essere creato preliminarmente nel metodo `ordina`. Il metodo `ord` riceve l'accesso all'array tramite un ulteriore parametro. Ecco il codice con queste modifiche:

```
public static void ordina(int[] a) {
  int[] supporto = new int[a.length];
  ord(a, 0, a.length, supporto);
}

private static void ord(int[] a, int i, int j, int[] fusione) {
   //ordina la porzione di a dalla posizione i alla posizione j esclusa
  if (j > i + 1) {   // la porzione da ordinare contiene alemnto due elementi
    int mezzo = (i + j) / 2;

    //ordina ricorsivamente le due parti di array
    ord(a, i, mezzo, fusione);
    ord(a, mezzo, j, fusione);

    //fondi le due parti scrivendo il risultato in fusione
    int posB = i;
    int posC = mezzo;
    for (int k = i; k < j; k++)
      //assegna a fusione[k] il minimo tra a[posB] e a[posC]
      if (posC == j || posB < mezzo && a[posB] < a[posC])
        fusione[k] = a[posB++];
      else
        fusione[k] = a[posC++];
    for (int k = i; k < j; k++)
      a[k] = fusione[k];
  }
}
```

- In realtà è possibile evitare anche l'uso dell'array ausiliario per la fusione. Tuttavia l'algoritmo è piuttosto complicato, a meno di non aumentare i tempi di esecuzione.