Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

9 Novembre 2022

Definizione di metodi statici

Abbiamo cominciato a scrivere metodi statici per calcolare funzioni o stampare valori che soddisfano una data proprietà.

Quindi, abbiamo metodi che restituiscono un risultato di un dato tipo e metodi di tipo `void` che non restituiscono alcun valore.

In queste lezioni considereremo metodi che calcolano e restituiscono un risultato di un dato tipo.

Riprendiamo i problemi considerati all'inizio del corso e codifichiamo alcuni dei loro algoritmi come metodi statici.

N.B. I problemi definiti su numeri naturali ora verranno definiti su numeri interi.

Massimo di due numeri

Es.1. Scrivere un metodo che, dati due numeri interi m ed n, restituisce il massimo tra m ed n.

Input: $m, n \in \mathbb{Z}$
Output: $max(m, n)$

L'*intestazione del metodo* è ricavata dalle informazioni su input ed output del metodo. Il *corpo del metodo* è costituito da un comando condizionale (o da un'espressione condizionata).

```
public static int max2 (int m, int n) {
  if (m > n)          //return (m>n ? m : n);
    return m;
  else
    return n;
}
```

Massimo di tre numeri

Es.2. Scrivere un metodo che, dati tre numeri interi m, n, p, restituisce il massimo dei tre numeri.

Input: $m, n, p \in \mathbb{Z}$
Output: $max(m, n, p)$

```
public static int max3 (int m, int n, int p) {
  if (m > n)
    if (m > p)
      return m;
    else
      return p;
  else
    if (n > p)
      return n;
    else
      return p;
}
```

Massimo di una sequenza (array) di numeri

In Java una sequenza di elementi (tutti dello stesso tipo) viene implementata come un array.

Quindi, i problemi su sequenze saranno formulati in termini di array (assumiamo che tali array siano non vuoti).

Es.3. Scrivere un metodo che, dato un array di interi a, restituisce il valore massimo in a.

Input: `int[] a`
Output: valore massimo in a

```
public static int maxArray (int[] a) {
  int max = a[0];
  for (int i=1; i<a.length; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
```

Massimo di un array di interi

Confrontando il metodo con il diagramma di flusso dato all'inizio del corso, possiamo notare:

- la variabile *n* è rimpiazzata dall'elemento `a[i]` dell'array;

- si passa all'elemento successivo incrementando l'indice `i`;

- verificare se si è alla fine dell'array (sequenza) si basa sul valore dell'indice `i` (e non più sul trovare un valore particolare che funge da marcatore di fine sequenza).

In modo simile possono essere definiti i metodi per calcolare il *minimo* di due interi, di tre interi e di un array di interi.

N.B. In caso di parametri di tipo array, il riferimento all'array è passato per valore.

Una classe di test per maxArray

Assumiamo di aver definito i tre metodi max2, max3 e maxArray nella stessa classe Max.

In una classe di test MaxTest un semplice metodo main per testare il metodo maxArray è il seguente:

```
public static void main (String[] args) {
  int[] a = new int[args.length];
  for (int i=0; i<a.length; i++) {
    a[i] = Integer.parseInt(args[i]);
  }
  System.out.println("Massimo = " +
Max.maxArray(a));
}
```

Dopo la compilazione con javac MaxTest.java, si ha:

```
java MaxTest 33 -19 4 91 120 -54 76
Massimo = 120
```

Stringa di lunghezza massima in un array

Es.4. Scrivere un metodo che, dato un array di stringhe a, restituisce la stringa di lunghezza massima in a.

Input: `String[] a`
Output: la stringa di lunghezza massima in a

```
public static String maxString(String[] a) {
  String max = a[0];
  for (int i=1; i<a.length; i++)
    if (a[i].length() > max.length())
      max = a[i];
  return max;
}
```

Nel caso in cui esistano più stringhe di lunghezza massima nell'array, questo metodo restituisce la *prima* stringa (i.e., quella più a sinistra).

Un semplice test

Assumiamo di aver inserito la definizione del metodo `maxString` nella classe `Max`.

Per testare la sua correttezza si può semplicemente invocarlo in un `main` passandogli l'array `args` come parametro attuale:

```
public static void main (String[] args) {
  System.out.println("Stringa␣massima␣=␣" +
Max.maxString(args));
}
```

Dopo la compilazione con `javac MaxTest.java`, si ha:

```
java MaxTest alfa csi ro beta gamma
Stringa massima = gamma
java MaxTest alfa csi ro beta
Stringa massima = alfa
```

Contare occorrenze

Es.5-6. Scrivere un metodo che, dati un array di interi a ed un intero n, restituisce il numero delle occorrenze di n in a.

Input: `int[] a`, `int n`
Output: numero di occorrenze di `n` in `a`

```
public static int occorrenze(int[] a, int n) {
  int cont = 0;
  for (int i=0; i<a.length; i++) {
    if (a[i] == n)
      cont++;
  }
  return cont;
}
```

In modo simile possiamo contare numeri (o elementi di un altro tipo, in generale) che soddisfano una data proprietà.

Contare occorrenze di un carattere in una stringa

Scrivere un metodo che, dati un carattere c ed una stringa s, restituisce il numero delle occorrenze di c in s.

Input: `char c`, `String s`
Output: numero di occorrenze di c in s

```
public static int occorrenzeCar(char c,
String s) {
   int cont = 0;
   for (int i=0; i<s.length(); i++) {
      if (s.charAt(i) == c)
         cont++;
   }
   return cont;
}
```

Stesso algoritmo, operazioni diverse per lavorare su stringhe.

Sommare i numeri in un array

Es.7. Scrivere un metodo che, dato un array di interi a, restituisce la somma degli elementi in a.

Input: `int[] a`
Output: sommatoria degli elementi in a

```
public static int sommaA (int[] a) {
   int sum = 0;
   for (int i=0; i<a.length; i++) {
       sum = sum + a[i];
   }
   return sum;
}
```

In modo simile si può calcolare la produttoria degli elementi.

Concatenare le stringhe in un array

Scrivere un metodo che, dato un array di stringhe a, restituisce la concatenazione delle stringhe in a (da sinistra a destra).

Input: `String[] a`
Output: concatenazione delle stringhe in a

```
public static String concA (String[] a) {
  String s = "";
  for (int i=0; i<a.length; i++) {
     s = s + a[i];
  }
  return s;
}
```

Verificare l'occorrenza di un elemento

Es.8. Scrivere un metodo che, dati un array di interi `a` ed un intero `n`, restituisce `true` se `n` compare in `a`, `false` altrimenti.

Input: `int[] a, int n`
Output: `true` se `n` in `a`, `false` altrimenti

```
public static boolean occorre(int[] a, int n) {
  int i = 0;
  boolean trovato = false;
  while (i<a.length && !trovato) {
    if (a[i] == n)
      trovato = true;
    i++;              // possibile mettere else
  }
  return trovato;
}
```

Soluzione senza la variabile booleana

Un metodo equivalente che sfrutta il costrutto `return` senza introdurre una variabile booleana è il seguente:

```
public static boolean occorre(int[] a, int n) {
  int i = 0;
  while (i<a.length) {
    if (a[i] == n)
      return true;
    i++;
  }
  return false;
}
```

Al posto del `while` è possibile usare un altro comando iterativo (e.g. `for`).

Verificare l'occorrenza di un carattere in una stringa

Scrivere un metodo che, dati un carattere `c` ed una stringa `s`, restituisce `true` se `c` occorre in `s`, `false` altrimenti.

Input: `char c`, `String s`
Output: `true` se `c` occorre in `s`, `false` altrimenti

```
public static boolean occorreCar(char c,
String s) {
  for (int i=0; i<s.length(); i++)
    if (s.charAt(i) == c)
      return true;
  return false;
}
```

Almeno un elemento con una data proprietà

Es.9. Scrivere un metodo che, dato un array di interi a, restituisce `true` se *esiste almeno un* numero negativo in a, `false` altrimenti.

Input: `int[] a`
Output: `true` se almeno un `a[i]` è negativo, `false` altrimenti

```
public static boolean occorreNeg(int[] a) {
  for (int i=0; i<a.length; i++)
    if (a[i] < 0)
      return true;
  return false;
}
```

N.B. Il comando `return false;` è *fuori* dal `for` ed in sequenza ad esso.

Tutti gli elementi con una data proprietà

Es.10. Scrivere un metodo che, dato un array di interi a, restituisce `true` se *tutti* gli elementi di a sono pari, `false` altrimenti.

Input: `int[] a`
Output: `true` se *ogni* `a[i]` è pari, `false` altrimenti

```
public static boolean tuttiPari (int[] a) {
  boolean ok = true;
  int i = 0;
  while (i<a.length && ok) {
    if (a[i] % 2 == 0)
      i++;
    else
      ok = false;
  }
  return ok;
}
```

Soluzione senza la variabile booleana

Un metodo equivalente che sfrutta il costrutto `return` senza introdurre la variabile booleana è il seguente:

```
public static boolean tuttiPari (int[] a) {
  int i = 0;
  while (i<a.length) {
    if (a[i] % 2 == 0)
      i++;
    else
      return false;
  }
  return true;
}
```

Come scrivere questo metodo usando il `for` al posto del `while`?

Almeno k elementi con una data proprietà

Es.11. Scrivere un metodo che, dati un array di interi `a` ed un intero `k`, restituisce `true` se in `a` compaiono *almeno* `k` numeri strettamente positivi, `false` altrimenti.
Si assuma `k>0`.

Input: `int[] a`, `int k`
Output: `true` se esistono *almeno* `k` numeri `a[i]>0`, `false` altrimenti

Non appena si trovano i `k` elementi che soddisfano la proprietà data, si termina restituendo `true`.

Vediamo una soluzione con `return` usato al posto della variabile booleana `trovato` (vedi diagramma di flusso, prima versione).

Almeno k elementi: una soluzione

```
public static boolean almenokPos(int[] a,
int k) {
  int i = 0, cont = 0;
  while (i<a.length) {
    if (a[i] > 0) {
      cont++;
      if (cont >= k) return true;
    }
    i++;
  }
  return false;
}
```

Almeno k elementi: un'altra soluzione

Se mettiamo la condizione cont<k nella guardia booleana del ciclo, possiamo scrivere il metodo come segue (vedere ultima versione del diagramma di flusso):

```
public static boolean almenokPos(int[] a,
int k) {
  int i = 0, cont = 0;
  while (i<a.length && cont<k) {
    if (a[i] > 0)
      cont++;
    i++;
  }
  return (cont >= k);
}
```

Esercizi

Esercizio 1

Scrivere un metodo che, dato un array di interi `a`, restituisce `true` se tutti gli elementi di `a` sono identici, `false` altrimenti.

Esercizio 2

Scrivere un metodo che, dato un array di interi `a`, restituisce `true` se i suoi elementi sono in ordine crescente (`a[0]<a[1]<a[2]<...`), `false` altrimenti.

Esercizio 3

Scrivere un metodo che, dato un array di stringhe `a`, restituisce `true` se `a` è ordinato in modo crescente rispetto alla lunghezza dei suoi elementi, `false` altrimenti.
Esempio: dato `a = {"ad","cbah","wqzxtr"}`, il metodo restituisce `true`.